Ivan Porres, Tommi Mikkonen, Adnan Ashraf (Editors). Developing Cloud Software: Algorithms, Applications, and Tools

Book · October 2013

CITATIONS
0

READS
1,081

3 authors:

Ivan Porres
Åbo Akademi University
185 PUBLICATIONS **3,245** CITATIONS
SEE PROFILE

Tommi Mikkonen
University of Jyväskylä
442 PUBLICATIONS **4,865** CITATIONS
SEE PROFILE

Adnan Ashraf
Åbo Akademi University
45 PUBLICATIONS **1,123** CITATIONS
SEE PROFILE

Ivan Porres | Tommi Mikkonen
Adnan Ashraf (Editors)

DIGILE

Developing Cloud Software

Algorithms, Applications, and Tools

TURKU CENTRE *for* COMPUTER SCIENCE

TUCS General Publication
No 60, October 2013

Developing Cloud Software

Algorithms, Applications, and Tools

Edited by

Ivan Porres
Tommi Mikkonen
Adnan Ashraf

TUCS General Publication
No 60, October 2013
ISBN 978-952-12-2952-7
ISSN 1239-1905

Preface

Executed during years 2010-2013, Cloud Software is a Finnish research program, whose goal has been to significantly improve the competitive position of Finnish software intensive industry in global markets in the field of cloud computing. More than 30 Finnish IT companies and research organizations have participated in the program, and the output of the program has been considerable both in terms of scientific publications as well as industry transformation towards using cloud computing techniques.

This book contains selected contributions by the Cloud Software Program partners focusing on innovative algorithms, applications, and tools to develop new services and applications to be deployed in public and private cloud infrastructures. These are needed to fully utilize the concept computing as a utility as promised by cloud computing. In addition to the actual contributions, the first chapter of the book contains an introduction to cloud computing from the point of view of software development. For the rest of the book, each chapter tackles a concrete problem relevant to software development for cloud infrastructures that originated in the program company partners.

We want to thank all the participants in the Cloud Software program for their efforts, shared experiences, and insights during these four years that have been materialized in the contents of the book. Specially, we want to thank Dr. Janne Järvinen, focus area director, Professor Veikko Seppänen, academic coordinator, Dr. Tua Huomo, program coordinator, and Dr. Pauli Kuosmanen, Digile CTO, for leading the program. In addition, we wish to thank Hans Ahnlund and Mika Laurila (ECE Ltd.), Marc Englund (Vaadin Ltd.), Sami Helin (Steeri Oy), Jarno Kallio (Packet Video Finland), Risto Laurikainen (CSC), and Heikki Nousiainen (F-Secure) for performing reviews for the different chapters that constitute the book. Finally, we also wish to thank all the contributors of this book for their dedicated efforts both in the program itself as well as in reporting the results.

Ivan Porres, Tommi Mikkonen, Adnan Ashraf
October 2013

Acknowledgments

This work was supported by TEKES, the Finnish Funding Agency for Technology and Innovation, as part of the Cloud Software program of DIGILE, the Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT and digital business.

Contents

1 Introduction to Cloud Computing Technologies 1
- 1.1 Technology Drivers and Adoption Problems . . . 2
- 1.2 Background Technologies . . . 7
  - 1.2.1 Cloud Hardware Virtualization Technologies . . . 7
  - 1.2.2 Sandboxing . . . 9
  - 1.2.3 Virtualized Network Block Storage . . . 10
  - 1.2.4 Network Virtualization . . . 11
- 1.3 Scalable Cloud Technologies . . . 12
  - 1.3.1 Scalable Datastore . . . 13
  - 1.3.2 Scalable File Storage . . . 17
  - 1.3.3 Scalable Batch Processing . . . 18
  - 1.3.4 Approaches to Fault Tolerance in Cloud Infrastructure 20
  - 1.3.5 Latency and Clouds . . . 21
- 1.4 Cloud from Application Programmer View . . . 23
  - 1.4.1 Infrastructure as a Service . . . 23
  - 1.4.2 Platform as a Service . . . 27
  - 1.4.3 Software as a Service . . . 31
  - 1.4.4 Discussion . . . 36
- 1.5 Conclusions . . . 36

2 The Social Devices Platform: An Infrastructure for Social Devices in a Mobile Cloud 43
- 2.1 Introduction . . . 44
- 2.2 Social Devices . . . 45
  - 2.2.1 Motivation and Background . . . 45
  - 2.2.2 Running Example . . . 47
  - 2.2.3 Requirements and Characteristics . . . 49
- 2.3 The Social Devices Platform . . . 52
  - 2.3.1 Overall Architecture . . . 53
  - 2.3.2 ProximityServer: Managing Proximity Information . . 54
  - 2.3.3 StateServer: Managing Device States and Properties . 54

2.3.4 Controller: Triggering and Scheduling of Actions 55
2.3.5 CaaS: Configuration of Actions 56
2.3.6 Orchestrator: Execution of Actions 57
2.3.7 SocialDevicesClient: The Client for Social Devices . . 58
2.4 Related Work 59
2.5 Discussion and Results 62
2.6 Conclusions 66

3 Prediction-Based Virtual Machine Provisioning and Admission Control for Multi-tier Web Applications 71
3.1 Introduction 72
3.2 Related Work 75
3.2.1 VM Provisioning Approaches 77
3.2.2 Admission Control Approaches 78
3.3 Architecture 80
3.3.1 Load Balancer 80
3.3.2 Global Controller 82
3.3.3 Admission Controller 83
3.3.4 Cloud Provisioner 84
3.3.5 Entertainment Server 84
3.3.6 Application Server 84
3.3.7 Application Repository 85
3.4 Algorithms 86
3.4.1 Load Prediction 89
3.4.2 The Server Tier 90
3.4.3 The Application Tier 92
3.4.4 Admission Control 93
3.5 Experimental Evaluation 94
3.5.1 VM Provisioning Experiments 95
3.5.2 Admission Control Experiments 100
3.6 Conclusions 106

4 Proactive Virtual Machine Allocation for Video Transcoding in the Cloud 113
4.1 Introduction 114
4.2 Video Bit Stream Structure 116
4.3 System Architecture 118
4.4 Video Segmentation 120
4.5 Proactive VM Allocation Algorithms 121
4.5.1 VM Allocation Algorithm 122
4.5.2 VM Deallocation Algorithm 124

4.6 Load Prediction . . . 126
4.7 MPI-Based Distributed Video Transcoder . . . 127
4.8 Simulation Results of VM Allocation . . . 128
4.8.1 Experimental Design and Setup . . . 128
4.8.2 Results and Analysis . . . 131
4.9 Evaluation of MPI-Based Distributed Video Transcoder . . . 132
4.9.1 Experimental Design and Setup . . . 133
4.9.2 Results and Analysis . . . 134
4.10 Related Work . . . 136
4.11 Conclusion . . . 138

5 Hadoop in Large Scale Data Analytics for Bioinformatics 145
5.1 Introduction . . . 146
5.2 MapReduce . . . 149
5.2.1 Execution Model . . . 150
5.2.2 Distributed File System . . . 154
5.3 Hadoop . . . 156
5.3.1 Pig . . . 158
5.3.2 Hive . . . 159
5.3.3 HBase . . . 162
5.4 Hadoop in Bioinformatics . . . 166
5.4.1 Hadoop-BAM . . . 167
5.4.2 SeqPig . . . 176
5.5 Closing Remarks . . . 176

6 Performance Testing in the Cloud Using MBPeT 191
6.1 Introduction . . . 191
6.2 Related Work . . . 193
6.3 The Performance Testing Process . . . 195
6.3.1 Model Creation . . . 195
6.3.2 Model Validation . . . 195
6.3.3 Test Setup . . . 195
6.3.4 Load Generation . . . 196
6.3.5 Monitoring . . . 196
6.3.6 Test Reporting . . . 196
6.4 MBPeT Tool Architecture . . . 197
6.4.1 The Master Node . . . 197
6.4.2 The Slave Node . . . 202
6.4.3 Graphical User Interface . . . 204
6.5 Model Creation . . . 204
6.5.1 Workload Characterization . . . 205

6.5.2 Workload Modeling Using PTA . . . 206
6.6 Performance Testing Process . . . 207
6.6.1 Test Setup . . . 207
6.6.2 Load Generation . . . 209
6.6.3 Test Reporting . . . 211
6.7 Experiments . . . 211
6.7.1 YAAS . . . 211
6.7.2 Test Architecture . . . 212
6.7.3 Load Models . . . 212
6.7.4 Experiment 1 . . . 215
6.7.5 Experiment 2 . . . 219
6.8 Conclusions . . . 222

7 Cloud Communication Service 227
7.1 Introduction . . . 228
7.2 Business Scenarios . . . 228
7.3 Past work . . . 230
7.3.1 Framework . . . 231
7.3.2 Quality of an Ontology . . . 232
7.3.3 Cloud Communication System as a Big Data Problem . 233
7.4 Human Interaction . . . 237
7.4.1 Information Filtering and Recommender Systems . . . 238
7.4.2 Content-based Information Filtering . . . 238
7.4.3 Collaborative Filtering . . . 239
7.4.4 Knowledge Based Recommendation . . . 239
7.5 Implementation and User Interfaces of the Cloud Communication System . . . 240
7.6 Conclusion . . . 245

8 HTML5 in Mobile Devices – Drivers and Restraints 249
8.1 Introduction . . . 249
8.2 Theoretical Background . . . 252
8.2.1 Technology Evolution . . . 252
8.2.2 Research Framework . . . 253
8.3 Technology Overview . . . 254
8.3.1 HTML5 . . . 254
8.3.2 Firefox OS . . . 256
8.4 Analysis . . . 257
8.4.1 Added Value . . . 257
8.4.2 Ease of Experimentation . . . 259
8.4.3 Complementary Technologies . . . 260

8.4.4 Incumbent Role . . . 260
8.4.5 Technological Performance . . . 262
8.5 Summary of Results and Discussion . . . 263
8.5.1 General Results . . . 263
8.5.2 Practical Examples . . . 265
8.6 Conclusions . . . 267

9 Collaborative Coding Environment on the Web: A User Study 275
9.1 Introduction . . . 275
9.2 Collaborative Development: The CoRED Approach . . . 277
9.3 Experiment . . . 280
9.3.1 Research Questions . . . 280
9.3.2 Proof-of-concept Session . . . 281
9.3.3 Experiment Setup . . . 282
9.3.4 Surveys . . . 283
9.4 Results . . . 283
9.4.1 Survey . . . 284
9.4.2 Log Data . . . 285
9.4.3 Interviews . . . 287
9.5 Developer Expectations for Real-time Collaborative IDEs . . . 290
9.6 Future Work . . . 293
9.7 Related Work . . . 294
9.8 Conclusion . . . 296

1 Introduction to Cloud Computing Technologies

Adnan Ashraf[1], Mikko Hartikainen[2], Usman Hassan[3], Keijo Heljanko[3], Johan Lilius[1], Tommi Mikkonen[2], Ivan Porres[1], Mahbubul Syeed[2], and Sasu Tarkoma[4]

[1]Åbo Akademi University, Turku, Finland
Email: firstname.lastname@abo.fi

[2]Tampere University of Technology, Tampere, Finland
Email: firstname.lastname@tut.fi

[3]Aalto University, Espoo, Finland
Email: firstname.lastname@aalto.fi

[4]University of Helsinki, Helsinki, Finland
Email: firstname.lastname@helsinki.fi

Abstract–This chapter presents the main technologies currently used in cloud computing, what are the main commercial offerings and what are their programming models. We discuss hardware virtualization technologies used in datacenters, three different service abstraction levels: infrastructure, platform and application and the main driver and adoption problems in cloud computing.

Keywords-Cloud computing, virtualization, scalability, IaaS, PaaS, SaaS.

The authors are listed in alphabetical order.

1.1 Technology Drivers and Adoption Problems

The cloud computing paradigm is a new framework for purchasing computing as a utility (Utility Computing) instead of using traditional datacenters to provide data processing capability. Perhaps the best overview of the technology drivers behind cloud computing is given in the report "Above the Clouds" [7, 6] from Berkeley, which also discusses the main obstacles and opportunities in cloud computing.

Cloud computing contains a number of technologies that are required to realize the "computing as utility" promise made by cloud computing vendors. Many of the key technologies have existed already before the term "cloud computing" was invented, while others have been born out of the Internet-scale deployment of computing in distributed data centers by several big companies such as Google, Amazon, Yahoo, and Salesforce.com. Consequently there are many definitions of cloud computing but the main distinguishing features of cloud computing are:

- Computing resources can be purchased *on-demand* from a virtually unlimited supply.
- The capital expenses needed to purchase computing resources up-front are changed to operational expenses, shifting the capital investment risk for under/overprovisioning to the cloud computing vendor.
- Computing is priced with a *pay-as-you-go* pricing model where capacity can be scaled up and down on a short term basis.

The National Institute of Standards (NIST) in the US has also decided to emphasize the elasticity of computing resources in their definition of cloud computing [40], which is a definition we can largely agree with. One frequently cited article defines the cloud as follows:

> Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs [54].

One of the main drivers for cloud computing are the economics of scale in datacenter building and operations costs. The pricing of hardware, electricity, cooling, and especially global network capacity is much more competitive

when bought for data centers with tens or hundreds of thousands of servers instead of small scale datacenter operations with maybe a hundred to a thousand servers.

This chapter focuses on the technical aspects of the available cloud computing platforms and mostly overlooks financial and business aspects of cloud computing. The key technologies we are discussing in this chapter are:

1. *Hardware virtualization*: In order to allow for maximum flexibility of offering customers the illusion of dedicated computing, storage, and networking infrastructure on a computing infrastructure shared with other clients, virtualization technologies are heavily used. Section 1.2.1 discusses the commonly used virtualization technologies, for example Xen is used by the Amazon cloud offering.

2. *Sandboxing*: Sometimes the overhead per (Linux or Windows) virtual machine can be quite significant, as typically each virtual machine is running its own kernel instance. Another commonly used approach is to use high-level programming languages with sandboxing (Python, Java, etc.) virtual machines to do the isolation between clients on a multi-tenant cloud infrastructure. Examples of this are the Google App Engine that uses Python and Java runtime environments that have been sandboxed to disallow things such as writing to files and opening of network sockets. Sandboxing is discussed in more detail in Section 1.2.2.

3. *Virtualized network block storage*: This is similar to having a virtualized network file server with redundancy available to mount storage to a virtual machine. A typical product is the Amazon Elastic Block Store, which can be used to mount network attached storage to a cloud server instance. The technologies employed here are similar to traditional file serving technology with a virtualization layer on top. This technology is discussed in more detail in Section 1.2.3.

4. *Scalable datastore*: One of the key technologies in scalable web services is the scalable datastore, a database component to manage the data behind the web application. There is a large interest in *NoSQL* (for *No SQL* or *Not Only SQL*) datastores. This is a quite central topic that we will discuss at length in Section 1.3.1.

5. *Scalable file storage*: Another basic cloud building blocks is that of geographically replicated hashed file storage. Examples of this service include the Amazon S3 storage system, and the recently announced

Google storage system. The data in replicated hashed file storage is accessed through a REST (via HTTP) based interface, and can thus be directly linked into in web sites. Replicated hashed file storage is often used to store the virtual machine images that are loaded to virtual machines at startup, to store backups of block storage, as well as to store large binary blobs (images, software, etc.) of data served through Web servers, as well as the seeds of data to be distributed through content distribution networks. These will be discussed in Section 1.3.2

6. *Scalable batch processing*: One of the key new technologies used in the cloud are scalable batch processing systems. The main reference implementation here is Google MapReduce and its open source implementation Apache Hadoop. This will be discussed in detail in Section 1.3.3.

7. *Cloud controller*: All of the cloud offering provide either a command line or a Web based interface to deploy and administer cloud computing, including not only computing but also storage and networking. Examples include the Eucalyptus Cloud Controller (CLC), the OpenNebula Virtual Infrastructure Manager, and the Web based Google App Engine Administration Console. These technologies will be discussed in Chapter 1.4. The different types of approaches (Infrastructure as a Service, Platform as a Service, Software as a Service) are addressed in individual Sections 1.4.1, 1.4.2, and 1.4.3.

In addition to technologies listed above most of the traditional Web application and web content serving technologies such as web services using load balancing and caching are quite predominant in Cloud application development, as one of the main drivers of scalable technologies are Web applications deployed at the massive Internet scale. For example, clouds are used to do the distribution of static Web content globally. Several providers such as Akamai and Amazon with its Cloudfront service offer caching services for static content using globally distributed network of datacenters to minimize the Internet data transfer fees for providing Web-based services. As these are basically Web serving content delivery networks, they will not be discussed further in this chapter.

Figure 1.1 presents an overview of the central elements in cloud computing. The lowest layer pertains to datacenters, clusters, and networking. Flexibility is achieved by using virtualization, and creating and moving platform instances at runtime. On the client-side, the Web browser is becoming a key platform for applications. Various Web application frameworks are then used through the browser. On a higher level, the aim of the cloud infra-

structure is to support on-demand service creation, management, and access. Open APIs are key components for interoperable cloud-based systems.

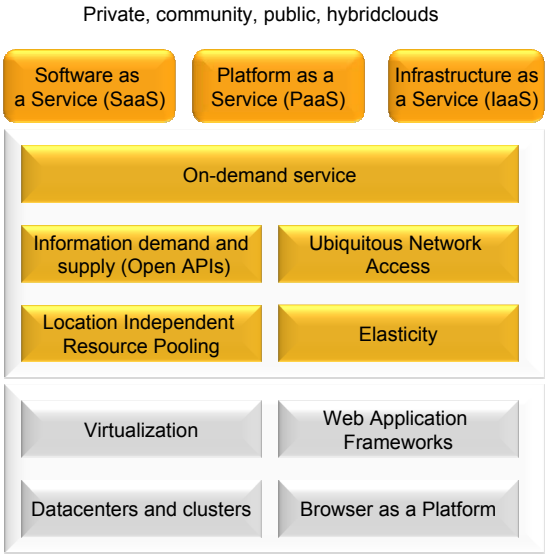


Figure 1.1: Overview of cloud services

The figure highlights the different roles found in cloud computing, namely the service consumer, provider, and developer. The service provider is a central element of cloud computing, because it facilitates the deployment and execution of various building blocks. The service provider achieves flexibility through virtualization and dynamic configuration of the runtime system. This flexibility that takes the supply and demand of content into account can be seen as a central feature of clouds. Virtualized resources include networks, CPUs, and storage. In order to implement and manage a cloud platform, a number of management features are needed. The necessary management features include reporting, Service Level Agreements, capacity planning, and billing. The software layer providing the virtualization is called a virtual machine monitor or hypervisor. A hypervisor can run on bare hardware (Type 1 or native VM) or on top of an operating system (Type 2 or hosted VM). The service developer uses APIs exposed by the cloud platform and the software it is executing. The service developer needs to have tools for service creation, deployment and publishing, and analyzing the service at runtime.

The service consumer uses various APIs and user interfaces to access the deployed services.

The services of Cloud computing can be divided into three categories: Software-as-a-Service (SaaS), in which a vendor supplies the hardware infrastructure, the software product, and interacts with the user using a portal. Platform-as-a-Service (PaaS), in which a set of software and development tools are hosted by a provider on the provider's infrastructure, for example, Google's AppEngine. Infrastructure-as-a-Service (IaaS), which involves virtual server instances with unique IP addresses and blocks of on-demand storage, for example, Amazon's Web services infrastructure. Figure 1.2 shows layer architecture of cloud computing.

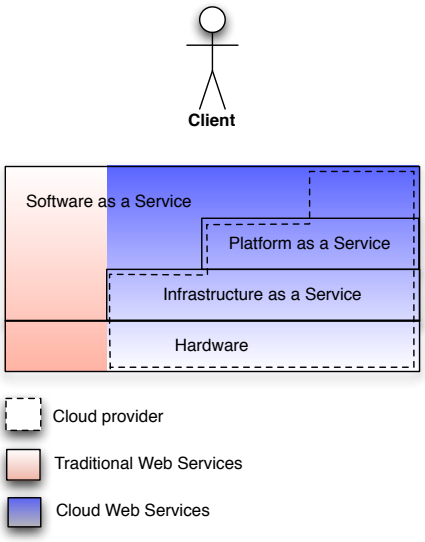


Figure 1.2: Layer architecture of cloud software

This chapter focuses on the cloud computing technologies from two different perspectives. The first one is the view of the application programmer: What kinds of application frameworks do the different cloud providers provide and what are the benefits and drawbacks on the frameworks in question. The second perspective is that of a cloud service provider: What technologies are employed in the cloud platforms, especially concerning the scalability of services. Many of the available implementations share similar (or even the same) components, and they are often composed by mixing and matching

proprietary and open source components.

1.2 Background Technologies

There are some commonly used technologies that have already existed before the introduction of cloud computing, but which have lately experienced a renaissance due to the introduction of cloud computing. Such technologies include in particular the following:

- hardware virtualization technologies
- virtualized network block storage
- sandboxing

These technologies play such a big role in current approaches to cloud computing that we have decided to address them separately.

1.2.1 Cloud Hardware Virtualization Technologies

Virtualization, or the capability to make one computer appear as several computers or a totally different computer, is a 4 decades old idea, introduced by IBM in its 7044 computer together with the *Compatible Time Sharing System* (CTSS) developed by MIT.

Virtualization is key component in Cloud computing as it allows one to distinguish the underlying hardware from the operating system, and allows the cloud hardware provider to easily let the client run any operating system that is needed.

The report *Secure Virtualization and Multicore Platforms State-of-the-Art report*, by Heradon Douglas and Christian Gehrmann [20] provides a good overview of the underlying techniques and issues.

Virtualization techniques can be split into two main approaches:

1. *System virtualization* in which the entire system is virtualized. This enables multiple virtual systems to run concurrently totally isolated from each other. The *hypervisor* or *virtual machine monitor* provides access to memory, devices, network, including the CPU. As a consequence, the Guest operating system thinks it has the machine for itself.

2. *Para-virtualization* in which the guest operating system is modified to cooperate with the hypervisor. The guest is modified to use interfaces that are safer or more efficient to use than the original guest operating system interfaces.

The two main hypervisor types are the following:

- Type 1 runs directly on the host's hardware and executes the guest operating system.
- Type 2 (or hosted) runs within an operating system.

System virtualization typically requires hardware support from the processor. Such support is provided e.g. by Intel Virtualization technology Intel-VT [33], AMD's support for virtualisation AMD-V [4], or ARM's TrustZone [53] . Thus, system virtualization is typically only supported for newer IA-32, Xeon, Itanium, Athlon, and Phenom families of processors.

System virtualization can also be achieved by *pre-virtualization*, in which the guest operating system code is scanned before execution by the hypervisor and then modified at run-time, thus resembling a Just-In-Time compiler. This approach naturally comes at a premium performance wise.

A comprehensive list of different virtualization solutions is available on wikipedia [15]. The list contains over 70 different solutions, with either open-source, or commercial licensing. Below, we focus on 3 solutions, KVM, XEN, and VMWare. This choice is motivated by the fact that XEN is the open-source market leader, while VMWare can be considered the commercial market leader. KVM has been included in the list as it is gaining quite a lot of interest in the Linux community.

XEN

The Xen hypervisor was created at the University of Cambridge at the end of the 1990's as part of the Xenoserver research project. The first open-source release of the Xen hypervisor was done 2002, and the current version of the hypervisor is 4.0. It can thus be considered a mature and stable product. Commercial support is provided by XenSource Inc. Xen is also provided as the virtualization solution by solution providers like Red Hat, Novell, and Sun. Xen is currently marketed by Citrix (`http://www.citrix.com/`).

Xen is usually considered a para-virtualization solution, although version 4.0 adds capabilities for system virtualization. Xen handles device drivers by running a special operating system in a special high-privilege Xen domain (dom0). This operating system handles all device driver requests and has optimized device drivers for the available hardware. The guest operating system then has to be modified to work against these interfaces.

VMWare

VMware (`http://www.vmware.com/`) offers a commercial hypervisor ESX [58]. ESX runs on "bare metal" and does not require a separate operating system. Instead it comes with an included Linux kernel that is booted first and used to load special device drivers and other features required by the hypervisor. The Linux kernel provides access to all devices of the system to the guest operating system using these drivers. In principle, VMWare is thus a para-virtualization solution. However "Scan-Before-Execution" the VMWare marketing term for its run-time pre-virtualization technology, allows the guest operating system to run unmodified on VMWare.

KVM

KVM is a newcomer among virtualization solutions. What KVM provides is a solution to make a Linux kernel into a hypervisor by loading a module. Each guest operating system is now a process in user-mode of the KVM hypervisor. KVM assumes that it is running on a processor with hardware support for virtualization. Thus it is not possible to run it on older processors, nor is any such support planned.

KVM consists of two parts: the KVM module that is used to virtualize memory and QEMU [46], an emulator, for virtualization of I/O.

Summary

Figure 1.3 presents a summary of well-known hypervisors including the three hypervisors mentioned above. The key properties of hypervisors include whether or not the system is open source, on what level does it operate (type 1 or 2), what hardware is supported and what hardware can be virtualized, and additional features such as live nested virtualization and live migration.

1.2.2 Sandboxing

Sandboxing is a commonly used security mechanism that separates programs and resurces from one another. By including the different applications in separate sandboxes, the infrastructure used to host them can be shared by numerous applications, some of which may have different trust levels. Moreover, it is easy to use experimental software in the same infrastructure as the production software, since by encapsulating the different systems into sandboxes of their own they can not cause harm to each other.

	Xen	KVM	VMWare	Hyper-V
Open source	Yes	Yes	No	No
Type 1/2	1.5 (Dom0 privileged guest)	1.5 (Linux kernel, Qemu)	2	2
Hardware	x86 / x86_64	x86 / x86_64	x86 / x86_64	x86_64
Virtual hardware	Qemu: x86 / x86_64	Qemu: X86 / x86_64	x86 / x86_64	x86 / x86_64
Features	Nested virtualization, live migration	Nested virtualization, live migration		
Association	Citrix	Red Hat / Intel	VmWare	Microsoft

Figure 1.3: Comparison of hypervisors

In the simplest form, sandbox systems are really isolating applications from each other and the hosting operating system in full. They have no way to interact, exept indirectly in terms of processor time, which they must share, provided that the same computing infrastructure is used. However, it is also common that not everything is isolated to such a degree, but different privileges can be offered to applications in exchange for e.g. providing reasonable evidence that the developer is authorized to use some services. Such a fine-grained sandboxing system can be implemented using so-called capabilities, which can be used to provide an access to different resources based on more detailed definitions.

Since sandboxing has been proven a really useful technology in many fields of computing, it is not uncommon to find different implementations that have been geared towards some particular area of application. In the realm of cloud computing, the most common use of sandboxing is together with a virtualization system, where the goal of sandboxing is to provide an illusion of a single computer, dedicated to the developer, which is isolated from the rest of the applications run in the same server farm or datacenter.

1.2.3 Virtualized Network Block Storage

The goal of storage virtualization is to abstract the physical location of the data from users and developers. Provided with adequate implementation,

this leads to location independence. The role of the virtualization storage is to provide a mapping from the perceived data to the actual physical location.

For obvious reasons, the actual form of the mapping is implementation dependent. For example, there may be limitations on granularity of the mapping, where different implementations provide a scale from a full, physical single disk residing physically in a certain computer to small subsets of the disk, provided in e.g. megabytes or gigabytes.

Commonly available implementations allow heterogeneous management of multi-vendor storage systems. Consequently it is possible to build a virtualized system out of best-suited component subsystems, which may provide different quality of service in terms of e.g. access speed.

The benefits of virtualized storage systems in general are many. They include at least the following:

- *Non-distruptive data migration.* With the virtualized system, data can be migrated to different locations even if it is being used. Consequently, there is more freedom on organizing the data in the network in accordance to the services that are being provided and computers and data storages that are currently available.

- *Improved utilization.* As is general with cloud computing, one of the gains of using a virtualized storage is the ability to use the available resources in a more optimized fashion.

- *Simplified management.* Another common gain of cloud computing is the fact that one needs less management points when relying on virtualized storage. This in turn simplifies management.

1.2.4 Network Virtualization

OpenFlow is an open standard that allows to run experimental protocols in production networks [39]. OpenFlow is a feature added to switches, routers, access points (APs) and basestations, allowing these datapath devices to be controlled through an external, standardized API. Major switch vendors are now implementing the system and it is used by universities to deploy innovative networking technology. Basically, OpenFlow is a software-defined Ethernet switch. Software-defined networking is expected to be one of the new emerging research topics in computer networking.

Figure 1.4 presents an overview of the OpenFlow protocol. Routers and switches implement the open API that allows administrators and control components to create, modify, and remove flows in the flow table. The protocol is a step towards software-defined networks.

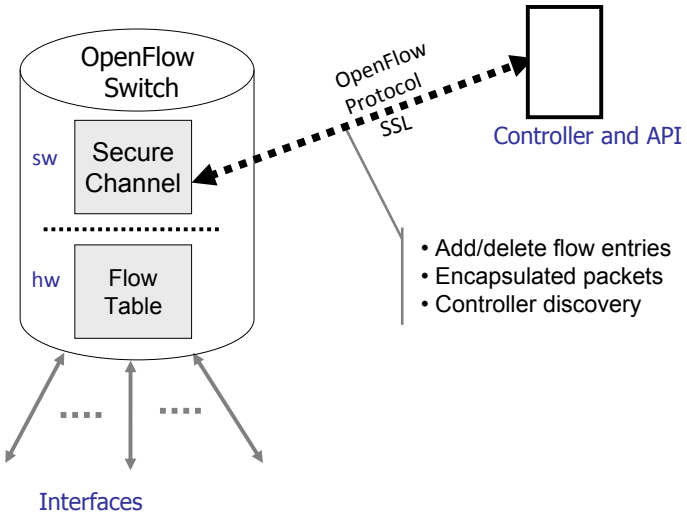


Figure 1.4: The OpenFlow protocol

NOX is an open source network control platform that can control all connectivity on the network including forwarding, routing, which hosts and users are allowed. NOX is a control plane element that can be used with OpenFlow switches [28].

1.3 Scalable Cloud Technologies

A key driver behind cloud computing are Web applications deployed at the Internet scale. One of the problems with these applications is the variability of demand in the capacity needed to serve users. An application that proves successful on the Internet, for example a game, can have its load increased dramatically in a very short time period. For example, when released as a Facebook plugin, Animoto (`http://animoto.com/`) traffic doubled every 12 hours for 3 days [6]. If the application serving the load is not constructed from scalable building blocks, such scaling to heavy Internet scale loads is not possible. Another way to see these scalable cloud computing technologies is that they are software based approaches to horizontally scale data processing to be run on a large number of small machines instead of a few very powerful machines. This can have potential in cost reduction for hardware as well as potential for energy consumption reduction. As an example, Google discusses

the high energy consumption of typical servers at idle, and are suggesting architectures that are more energy-proportional, that is, the server power should be proportional to its application performance level [5, 8]. Thus the cloud should also be able to scale not only up but also down: when application load decreases, the number of active servers needed to serve the load should also be automatically decreased as well. Another example is the paper [52], which discusses the use of virtualization technology to power off idle machines for better power efficiency during hours of low load.

The scalable cloud computing technologies presented here can also be seen as a preview of technologies to be employed on a smaller (company private datacenter) scale in the future, as the need of scaling of systems to a "private cloud" of small commodity servers in an economical fashion becomes an issue.

1.3.1 Scalable Datastore

One of the key components to scalability on the Internet scale is that of a scalable datastore. The datastore is needed as a backend to Web applications serving dynamic Web pages built on top of the Representational State Transfer (REST) architectural style [23]. These datastores can be seen as databases with often very limited functionality but they are able to scale to even tens of thousands of servers in a single datastore instance. For example, all of the Google App Engine applications are sharing a single datastore instance making up a massively distributed datastore system.

Example scalable cloud datastores include Google Datastore (based on Google Bigtable [12, 13]), Amazon SimpleDB (`http://aws.amazon.com/simpledb/`), Amazon Dynamo [19] (based on Chord [48, 49]), the Yahoo! PNUTS (also called Sherpa) [16] system (which internally uses MySQL (`http://www.mysql.com/`) for ordered storage), the open source Apache Cassandra [36] (`http://cassandra.apache.org/`) system, the open source Tokyo Tyrant (`http://1978th.net/tokyotyrant/`) system, and the open source Project Voldemort (`http://project-voldemort.com/`), just to name a few examples. As can be seen from the list of systems above the field is still quite fragmented but also in very active development.

NoSQL Datastores

One of the current hot topics is *NoSQL* (for *No SQL* or *Not Only SQL*) datastores. A good short overview of the topic can be found in [50, 51]. In traditional database systems literature [29] the guarantees given to the user by most traditional database systems are denoted with the term ACID

(Atomicity, Consistency, Integrity, Durability). A part of the NoSQL movement is instead insisting on that a much more limited datastore functionality called BASE [24] (Basically available, soft state, eventually consistent) is to be adopted. The main idea of datastores implementing BASE is to favor availability (and often also low write latency) over consistency.

An example given by the Amazon Dynamo paper [19] is that of a shopping cart: A customer would prefer to have the shopping cart application available even though some datacenters of the Amazon infrastructure would not be currently available due to e.g., disk failures or network connection problems to other datacenters which might also manipulate the same shopping cart. In the rare occasion that a shopping cart update should manage to write its data only to a subset of the servers storing the database, and another update for the cart comes in, the database would contain two conflicting versions of the shopping cart data. This is conceptually very similar to merge conflicts in distributed revision control systems. When such inconsistency is noticed, the shopping cart application would eventually be presented with two conflicting versions of the contents of the shopping cart.

The approach taken by Amazon was to use the union of the two shopping carts as the "true contents" of the shopping cart in the case of such rare failure, and to handle the exceptional cases of missing deletes of items from the shopping cart by the user being presented with a slightly wrong contents of the shopping cart. This allowed the Amazon system to scale easier but at the expense of having each of the applications handle the inconsistent versions of the datastore contents in an application dependent fashion. Amazon also employs a lot of ACID systems in billing and order manipulation, but these are often not customer facing applications, and thus can tolerate the longer latencies and lower availability than the applications the customers are directly interacting with. So a good system design might employ a mixture of BASE and ACID datastores, depending on the application and its inherent requirements.

So one of the main debates in scalable datastores is BASE vs. ACID [45]. The debate is based on a theoretical result on distributed databases: Brewer argued in his CAP conjecture that it is impossible to build a database system that is:

- *Consistent*: The client perceives that a set of operations has occurred all at once.
- *Available*: Every operation must terminate in an intended response.
- *Partition tolerant*: Operations will complete, even if individual components are unavailable.

This was later proved to indeed be true [26], and without additional assumptions it is indeed impossible to create a distributed database system satisfying all the three properties at the same time. To overcome this impossibility result, one has to drop one of these three guarantees by either:

- Discarding partitions: CA: The database has to be centralized which immediately leads to scalability problems.
- Discarding availability: CP: A database server must disallow writes in case the update can not be written to a majority of the database servers due to network partition.
- Discarding consistency: AP: The database must allow for inconsistency of writes in case of network partition of the database servers.

Brewer gives in his PODC 2000 keynote some examples of systems in the different classes:

- CA systems include: Single-site databases, LDAP, and NFS. These systems are often based on two-phase commit algorithms, or cache invalidation algorithms.
- CP systems include: Distributed databases, distributed locking systems, and majority protocols. The systems often use pessimistic locking or majority (aka quorum) algorithms such as Paxos [37].
- AP systems include: Filesystems allowing updates while disconnected from the main server such as Coda, Web caching, and DNS. The employed mechanisms include cache expiration times and leases.

One of the key observations here is that caching of Web applications have already discarded consistency as caches might contain stale data not available anymore in the database. So in a way data staleness of data already exists in Web applications to some extent.

When we look at the BASE vs. ACID debate, it is interesting to note that the Google App Engine Datastore is by default ACID. The BASE proponents are the Amazon Dynamo and Yahoo! PNUTS (Sherpa) system, as well as the Cassandra datastore that can be configured either in BASE or ACID mode on a per table basis.

The Google App Engine Datastore as well as most other NoSQL datastores support a very limited notion of transactions, and most systems do not support table joins used in relational databases. The Google Datastore stores its key values in a sorted fashion by the primary key, and thus range queries

are still efficient. Also Google itself notes that Datastore writes are expensive compared to reads that can be usually served from a cache. This is not surprising given the fact that the written data has to be replicated (using GFS) to ensure durability. Thus a common scalable application programming strategy Google proposes to be used is to batch datastore writes using a memory cache and only periodically flush the required writes to the datastore "backup". So the Google Datastore can be seen as quite database-like datastore system (without complex transactions or table joins) engineered for massive scalability.

Key/Value Datastores

The other interesting approach to datastores is that of pure key/value stores. These are systems built on top of massive distributed hash-tables, and the only thing one can do is lookup a value based on a key (md5 checksums of key material are often used). One of the ideas underlying these systems was presented in the peer-to-peer system Chord [48, 49]. It uses a technique called *consistent hashing* to allow for nodes to enter and leave a peer-to-peer content lookup network with minimal overhead. Basically if a network contains n nodes each containing a set of values, if one node leaves the system, only $\mathcal{O}(\frac{1}{n})$ data items have to be reallocated to the remaining nodes. In other words only the data that went missing by the node leaving needs to be reallocated while other nodes do not have to move the data they still store around to other nodes. Similar result also holds for new nodes entering the system. The Amazon Dynamo and its descendant the Cassandra system uses this idea to implement a scalable key/value store with replication, and it is used in internal Amazon infrastructure (to implement parts of Amazon S3, most likely file location lookup). Many of the other key/value stores use the consistent hashing primitive as well to do load balancing in a way that minimizes the effects of adding or removing servers to the servers remaining to serve the application load. For an overview on the performance of Cassandra, which copies heavily from Dynamo for its load balancing and configurable consistency levels, as well as from Bigtable for its on-disk storage design, see [35].

Also in-memory caching systems based on key-value lookup such as the open source `memcached` (`http://www.memcached.org/`) are heavily used in cloud computing offerings and usually used as very high performance caches to persistent databases. However, as `memcached` is just a least-recently-used cache (forgetting data is a feature, not a bug), we do not discuss it at length here. It should be noted that `memcached` seems to be one of the most widely deployed pieces of infrastructure, though. For example, the combination of

`memcached` with MySQL database is often used datastore solution for applications with small load, and for applications where the load is almost exclusively read-load and MySQL replication is used to improve the read capacity by having lots of read-only MySQL replicas of a single master MySQL database. However, once write load is significant, the replication overhead becomes problematic with a straightforward MySQL replication solution.

Scalable Consensus Datastores

One of the interesting pieces of technology Google uses in their system is the Chubby system [10, 11]. It is a highly fault tolerant database, where the contents of the database are replicated over a number of servers. If more than half of the servers are available, the database can serve requests. The system is used to mainly maintain configuration data for other Google services, for example BigTable uses Chubby to store the master node identity, servers that are up, and the location of the root table in GFS. Bigtable (as well as other Google services) has been designed to reboot quickly if needed, and it fetches the initial configuration data from Chubby. Google also uses Chubby internally to store the DNS records of internal services, which allows for atomic updates of DNS data. Chubby is thus a piece of the infrastructure that needs to be kept running 24/7. The design is based on the classic Paxos algorithm [37]. In Paxos writes are very expensive, but read performance can be made quite good by using extensive client caching and cache invalidation technologies [10, 11]. An open source implementation of the Paxos algorithm is Keyspace by Scalien (http://scalien.com/keyspace/). Also the Cassandra datastore can be configured to implements a majority algorithm on a table basis.

1.3.2 Scalable File Storage

Another key component in scalable cloud computing technologies is that of scalable file storage. The main reference here is the Amazon S3 system that implements a file storage that is geographically replicated for durability. Amazon says that the data should still be available even if two different datacenters are affected. In practice this means that all of the data must be available on at least three geographic location. In practice storing data in many different geographically disjoint locations means high latency for updating data. The Amazon S3 system uses standard HTTP methods to read and write data, and also data stored on S3 can be directly linked into on Web pages as standard URLs. Google has a new competing product with almost identical interface called Google storage. Scalable file storage

can use many methods in data storage inside the cloud: it can compress the customer data, it can deduplicate (detect identical data blocks and only store one copy of the data), and it can replicate the data not only for storage but also for caching purposes on geographically disjoint locations, trying to minimize latencies in accessing data stored on the file storage.

These scalable file storages can be used for example to: Store virtual machine images and disk images for servers, backups, serving static content to the Web, seeding content distribution networks. For example Amazon S3 can be used to seed BitTorrent feeds of data, thus feeding peer-to-peer distribution of content on the Internet.

For filesystem type interface to data, see the Google filesystem (GFS) [25], and the Hadoop HDFS filesystem (`http://hadoop.apache.org/common/docs/current/hdfs_design.html`) whose design is heavily influenced by GFS.

1.3.3 Scalable Batch Processing

In addition to scaling up datastores and storage horizontally to a large number of machines, also asynchronous batch processing of data needs to horizontally scale to a large number of computers. The main cloud reference here is the Google MapReduce system [18, 17] and its open source clone Apache Hadoop (`http://hadoop.apache.org/`) originally developed at Yahoo!.

The Google MapReduce is an implementation of MapReduce, a distributed batch programming paradigm based on functional programming techniques. It consists of a framework for automatically distributing batch jobs onto a large number of worker machines, and the framework takes care of the scheduling and synchronization between the jobs. An overview of the system is presented in Figure 1.5. The input data to MapReduce is given in a very large file, usually split into large (64MB is typical) chunks of data to be processed in parallel by n mapper tasks. Each one of the chunks contains number of records (for example lines of text), which are fed into a user provided *map* function record at a time. The map function processes each line at a time and decides for each record to produce some number of records consisting of a pair $(key, value)$. Next the MapReduce framework does what is called a *shuffle* operation, which groups all the data from all the parallel mappers using a (user provided) hash function to distribute them over m reducer tasks. This is a very network bandwidth heavy operation, as each one of the n mappers has to communicate the values it has for each of the m reducers in parallel, amounting to $\mathcal{O}(n \cdot m)$ pairs of file transfers. After the temporary files have been transfered by the shuffle, they are next sorted locally by the reducers in order to give the user provided *reduce* function,

which is presented with the list of values attached to each key, for example as $(key, (value_1, value_2, \ldots))$.

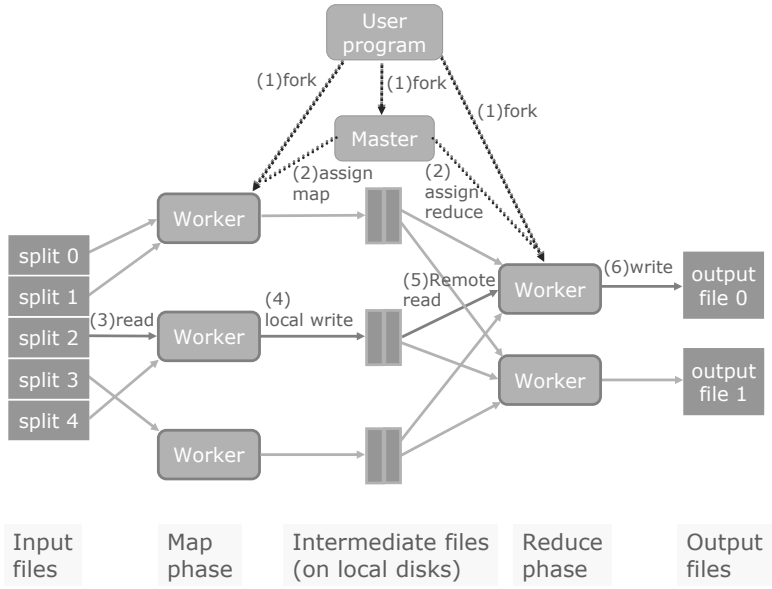


Figure 1.5: Overview of MapReduce

A really nice feature of the MapReduce framework is that it automatically parallelizes the work among the user provided number of computer nodes. Furthermore, the programs can be easily debugged, as the output of the program (following the rules of the framework) run in parallel is exactly the same as running the same program sequentially. This is the guarantee given by the functional programming framework. Also the functional programming paradigm gives very good fault tolerance: all the data produced by any number of mapper or reducer tasks can be lost and the framework can still continue making progress by rescheduling the re-execution of the lost jobs.

Google has been using the MapReduce framework to compute the production Web indexes Google uses to index the Web. For example, one of the heavy processing task is to compute the reverse Web link graph. That is, for each indexed site, collect all the sites linking to it. This perfectly matches the MapReduce framework. Also things like processing and doing statistics from log files matches the MapReduce framework nicely. Interestingly Google is not providing the full MapReduce feature to its customers. Also unfortu-

nately the MapReduce programming paradigm has been recently patented by Google.

The Apache Hadoop system is directly based on the design of the Google MapReduce and the Google Filesystem for its own distributed filesystem HDFS. The Hadoop system is written fully in Java and many cloud providers, for example Amazon, provides support for it in their service offering. Hadoop is used by many high volume production sites (`http://wiki.apache.org/hadoop/PoweredBy`), for example Facebook is running Hadoop clusters with 15 PetaBytes of storage using mainly the Hive (`http://hadoop.apache.org/hive/`) system implementing a SQL-style query language on top of the Hadoop MapReduce engine. Another Hadoop HDFS based database used mainly for batch processing is HBase (`http://hbase.apache.org/`), which is heavily inspired by Google Bigtable. The main applications for Hadoop seem to be log analysis, web indexing, and various data mining and customer analysis applications.

If a batch processing task fits the MapReduce framework, the framework gives good parallelization. In addition, MapReduce and Hadoop do not offer control of multiple parallel frameworks. In practice, there is a requirement to execute several different data processing frameworks, such as different versions of MapReduce and Hadoop, in parallel. The Nexus system presents a framework for running multiple frameworks in the same cluster [31]. The key idea of Nexus is to multiplex resources across frameworks and decouple job execution management from resource management. Figure 1.6 gives an overview of the Nexus framework.

1.3.4 Approaches to Fault Tolerance in Cloud Infrastructure

When providing scalable computing infrastructure on the cloud level the systems must be very fault tolerant and self-healing. When running datacenters with tens of thousands of computers there will always be some number of machines and hard disks that are broken. Most of the scalability solutions in the cloud space are built to have redundancy in a shared-nothing infrastructure. The aim of these systems is to provide an environment where one can power off any individual server in the datacenter, and the cloud infrastructure will recover automatically from the loss of the server without any manual intervention. The key techniques to achieve this is to have high redundancy in the datastore system for fault tolerance, and to never store any persistent data on the servers themselves: all data on the servers is just cached/preprocessed contents of the real application data that is stored in

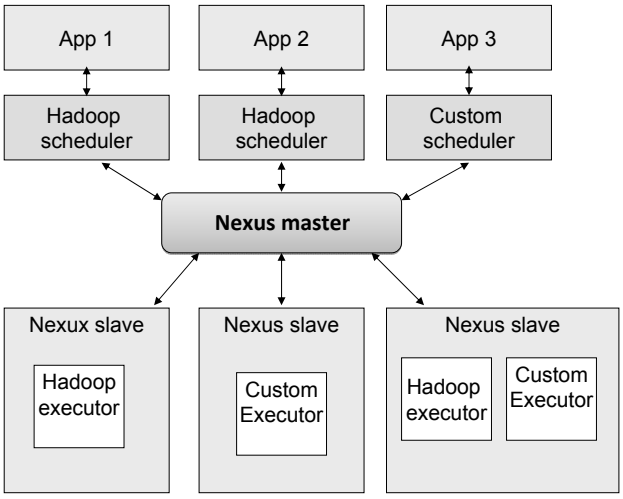


Figure 1.6: The Nexus system

the datastore. By doing so, all that is lost by powering down a server is the contents of a cache, that can be repopulated, and the server load can be redistributed among the remaining servers. Such a design also allows for powering off servers at off-peak hours, allowing the minimization of overall power usage of computing.

1.3.5 Latency and Clouds

On a much more philosophical level a key observation affecting the design decisions of future looking scalable cloud computing technologies is that computing capacity, memory capacity, storage capacity, and network bandwidth all improve at a much higher rate than latency improves. This is because ultimately latency is fixed by the speed of light, and especially when dealing with cloud computing systems where different computers of the cloud can be geographically very far from each other, latency certainly is an important issue. This has been observed by David Patterson in [42]. Maybe of much more practical interest are the three classical solutions to latency mentioned in the paper [42]: *Caching*, *Replication*, and *Prediction*.

If we look at the three above mentioned techniques in the scalable cloud computing technologies context, we observe all of the technologies are in use:

- *Caching* – Caching is one of the key components to cloud scalability, and is one of the reasons why cloud computing is heavily based on Web technologies that are themselves engineered with efficient caching in mind. In the cloud context especially memory based caches such as `memcached` have been introduced as additional write-through caches to lower back-end datastore read load.

- *Replication* – Replication is heavily used by all scalable cloud computing infrastructures. Many of the storage systems such as Amazon S3 and Google File System are using replication not only to provide improved data durability across geographically redundant replication, but also to improve the serving of "hot" files by replicating the highly requested files on a much higher number of file server nodes than the infrequently requested "cold" files. One of the implications of potentially very high replication ratios is that modifying data in-place becomes very expensive as basically all the (geographically distant) replicas have to be modified. The solution to this problem employed is to heavily emphasize write-once-read-many storage patterns with quite big block sizes in application design, see for example the Google BigTable design [13], as once written the files cannot be efficiently modified. Thus BigTable has been designed from the ground up to be run on a replicated file storage system to eliminate most of the random access write traffic to the replicated filesystem.

- *Prediction* – Quite a common architectural style these days is the precomputation of potential application database queries asynchronously beforehand and populating caches such as the main memory `memcached` beforehand with contents that the application might require in the future. This is once more done to hide latency from the user and to serve the common case queries from cache instead of a datastore. To implement this background processing one can employ asynchronous job queueing systems such as Amazon Simple Queue Service and scalable batch processing systems such as Hadoop to pre-populate the caches in advance.

As a practical consideration, if one develops applications that need to scale to large numbers of users on the cloud infrastructure, ACID datastore writes are going to become more and more expensive compared to the datastore reads. Thus applications need to be designed in a manner that tries to minimize the number of datastore writes per time unit to ensure application scalability, or to use a datastore that does not use ACID but BASE, and deal with the

potentially significant application complexity increase that is needed to deal with the inconsistent datastore.

1.4 Cloud from Application Programmer View

In terms of engineering, web application development is still in its infancy for obvious reasons – as long as the primary purpose of web development was the creation of web pages, there was no need to apply established software engineering principles to web development. However, cloud computing forces one to treat web development in the same fashion as software development in general. These include aspects like reusability, interoperability, and security, just to give some examples.

In the following, we will study different cloud computing approaches in terms of the type of cloud service they offer. The viewpoint is that of a programmer that might be interested in benefitting from the available cloud platform, not that of a potential cloud service developer that might wish to build a similar system.

1.4.1 Infrastructure as a Service

Infrastructure as a service (IaaS) is about providing a hosted computing infrastructure. A typical way to implement such a system is platform virtualization, where the user of the system can consider that the service corresponds to a piece of hardware and associated system software. One common approach seems to be that Linux type of a system is provided, where the developers can deploy their own software stack.

Examples of IaaS type systems include the following:

- Amazon EC2 (`http://aws.amazon.com/ec2/`)
- Eucalyptus project [41] (`http://open.eucalyptus.com/`)
- Ubuntu Enterprise Cloud (`http://www.ubuntu.com/cloud/`)

Next, we address these systems in more detail.

Amazon Elastic Computer Cloud

Amazon, which is better known from its Internet bookstore, was one of the first companies to enter into the cloud computing business. Amazon's Elastic Computer Cloud (Amazon EC2) was designed to make it easier for developers to use web-scale computing power [3]. Amazon promises 99.95% availability

for each EC2 region. Amazon EC2 offers a virtual computing environment where developer can launch multiple instances with variety of operating system or with custom application environment. Amazon offers a web service interface to launch instances.

One of key advantages in Amazon EC2 is that it allows a freedom for developers to choose their tools as they want. Amazon offers Amazon Machine Images (AMIs), which are preconfigured with several Linux distributions, Microsoft Windows Server, or OpenSolaris. Developers can customize AMI by choosing Amazon provided software (e.g. Java Application server, Database server). If offered operating systems or software do not meet developers' needs, they are always free build their own custom AMI. Since each developer can have the root access and can manage network access permissions, therefore the developer has total control over the software stack they use.

Amazon has three different kinds of pricing models: On Demand Instances, Reserved Instances, and Spot Instances. In On Demand Instances, the user pays for the capacity which is actually used. This approach is good when system needs to scale. In Reserved Instances, the user pays one time fee for each instance user wants reserve. With Spot Instances, Amazon is selling unused capacity of EC2. The price fluctuates based on supply and demand.

Amazon Elastic Block Store (EBS) offers persistent storage for Amazon EC2 instances [2]. EBS allows the user to create 1GB to 1TB volumes for use of EC2 instances. Instances see volumes as raw unformatted block devices and they can be used as any other block device (e.g. hard drive). EBS volumes are automatically replicated to prevent dataloss and they can be used as boot partitions.

Amazon divides locations into regions, e.g. US and Europe. These regions are divided into smaller areas - Availability Zones. EC2 instances can be placed into multiple locations in case of failure in an Availability Zone. Availability Zone are designed to be insulated from other AZs and they have inexpensive, low latency network connectivity to other Availability Zones in the same region.

Amazon EC2 uses Elastic IPs, static IPs that has been design for dynamic cloud computing. Elastic IP points to users account instead of EC2 instances. User controls the IP until the user chooses to release it. Elastic IP allows the user to remap IP to point instance, which the user has chosen. In case of a failure in Availability Zone, user can start new instance from other Availability Zone and keep running the service.

Amazon offers Virtual Private Cloud (VPC) technology for enterprises to extend their existing infrastructure [21]. VPC works as a secure isolated portion of the Amazon cloud. So far, Amazon VPC integrates EBS, EC2 and

Cloud Watch and rest of the Amazon cloud features are under development. Amazon provides enterprise a VPN connection and one cloud, where user can define up to 20 subnets.

Cloud Watch is the Amazon's cloud monitoring system. Cloud Watch enables user to monitor EC2 instances and Elastic Load Balancing (ELB) in real-time via Web Service interface. ELB distributes traffic automatically across multiple Amazon EC2 instances. From Cloud Watch, user can also enable Auto Scaling, which automatically scales capacity up or down depending upon the conditions that the user defines. This suits well in application that has high variability in usage.

Eucalyptus

Eucalyptus is an open source software used to implement cloud computing on compute clusters. Eucalyptus implements Infrastructure as a Service (IaaS) while giving the user the ability to run and control virtual machine instances deployed across a variety of physical resources [9]. Some aspects of underlying protocol and interface design for Eucalyptus are similar to Amazon Web Services (AWS). For example, the external interface to Eucalyptus is based on the API provided by Amazon [41]. The system is relatively easy to set-up, even on a local environment with limited resources. It is already a part of the Ubuntu Enterprise Cloud (UEC) installation.

Figure 1.7 presents an overview of the Eucalyptus system that follows a hierarchical design. The primary high-level components that comprise the Eucalyptus architecture are as follows:

- Node Controller (NC): An NC makes queries to discover the node's physical resources - the number of cores, the size of memory, the available disk space as well as to learn about the state of VM instances on the node (although an NC keeps track of the instances that it controls, instances may be started and stopped through mechanisms beyond NC's control) [41].

- Cluster Controller (CC): The CC schedules the distribution of virtual machines to the NC and collects resource capacity information [9]. Each CC may manage one or more NC(s). The Cluster Controller (CC) generally executes on a cluster front-end machine, or any machine that has network connectivity to both the nodes running NCs and to the machine running the Cloud Controller (CLC) [41].

- Storage Controller (Walrus): To put it simply, Walrus is a put/get storage service that implements Amazon's S3 interface, providing a

mechanism for storing and accessing virtual machine images and user data [41]. Walrus implements the REST (via HTTP), sometimes termed the "Query" interface, as well as the SOAP interfaces that are compatible with S3. Walrus provides two types of functionality. Firstly, users that have access to EUCALYPTUS can use Walrus to stream data into/out of the cloud as well as from instances that they have started on nodes. In addition, Walrus acts as a storage service for VM images. Root filesystem as well as kernel and ramdisk images used to instantiate VMs on nodes can be uploaded to Walrus and accessed from nodes [41].

- Cloud Controller (CLC): It is the entry-point into the cloud for users and administrators. It queries node managers for information about resources, makes high level scheduling decisions, and implements them by making requests to cluster controllers [41].

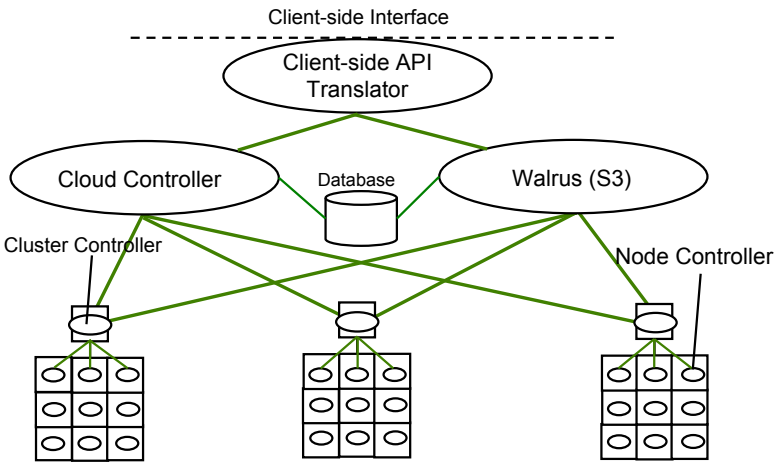


Figure 1.7: Overview of Eucalyptus

Each of these components runs as a web service on the respective machines; we have seen above that the machines distribution is separated for CLC/Walrus, CC(s) and NC(s). However it is also possible to set-up a test system with lesser resources. A minimal working "cloud" structure as mentioned in UEC set up guide is to have the CLC, Walrus and CC running on one machine and an NC service running on another.

Several performance measurements with Eucalyptus compared to the original AWS have been made. The result of these measurements is that a

private cloud can provide almost the same functionality and possibly better performance compared to the AWS. The performance can be easily improved with storage area networks (SAN) for storing the data in the S3/EBS directories (e.g. image files) in a better performing environment. This approach also helps a lot to reduce the time needed to start the virtual server instances. On the other hand, AWS offers unlimited scalability and thus it would be beneficial to combine both, private and public resources in a hybrid cloud [9].

Eucalyptus system has filled an important niche in the cloud-computing design space by providing a system that is easy to deploy atop existing resources, that lends itself to experimentation by being modular and open source, and that provides powerful features out-of-the-box through an interface compatible with Amazon EC2 [41]. Eucalyptus is an interesting product to build private cloud infrastructures for R&D. The performance of such an installation with commodity hardware is satisfying for most common scientific applications. Already now a large number of open source tools exist for cloud management. As also other cloud software providers start to implement Amazon AWS as a cloud computing interface AWS has the potential to become a de facto standard for cloud computing infrastructure services [9].

Ubuntu Enterprise Cloud

Company Canonical claims that Ubuntu is only Linux distribution to position itself as true Cloud OS. Ubuntu Enterprise Cloud(UEC) is one of the three components in Canonical's cloud strategy [56]. Other two components are Ubuntu Server edition and UbuntuOne. UEC and Ubuntu Server are aimed as IaaS and UbuntuOne is aimed as SaaS. Canonical's cloud project started with offering official AMI for Amazon EC2. Later, Canonical took Ubuntu Server Edition and integrated enchanted version of KVM [34] based Eucalyptus into the distribution. As a result Canonical got UEC – a user deployable cloud, which matches the API that AWS provides.

The architecture of UEC closely follows the architecture of Eucalyptus [41]. It has the same controllers (node controller, cluster controller and cloud controller) as Eucalyptus. One difference is that UEC includes also ESB Controller, which runs in the same machine as CC and is configured automatically when CC is installed. EBS Controller follows the same ideology as Amazon ESB [2].

1.4.2 Platform as a Service

Platform as a Service (PaaS) is about providing a computing platform that contains a complete solution stack, hosted as a service. Applications devel-

oped on top of PaaS can commonly be run as a part of the service. The biggest difference between PaaS and IaaS seems to be that PaaS usually assumes a certain kind of application model, together with associated libraries and system software, whereas in IaaS the developers have more freedom to select the systems they want to use. PaaS usually offers additional features such as load balancing, automatic provisioning and geographic replication.

Sample PaaS systems include the following:

- Google AppEngine (`http://code.google.com/appengine/`)
- Microsoft Azure (`http://www.microsoft.com/windowsazure/`)
- Heroku (`http://heroku.com/`)

Google AppEngine

Google App Engine is a service developed by Google to allow developers to run web applications on Google's infrastructure [27]. The reason for wanting to run web applications on Google's infrastructure is the access to their immense computing power and storage capabilities. The promise is that developers can start out small and then scale when the need arises.

One of the key advantages with App Engine is that there are no servers that need to be maintained by the developer, one can simply use part of the infrastructure Google already has in place. The need to have your own servers has traditionally been an issue for smaller projects, since they represent a considerable investment in hardware. By using App Engine the developer can simply use more processing power when the service usage grows.

App Engine allows developers to simply upload their code and have it deployed automatically, ready to be used by consumers. App Engine supports applications written in multiple languages. A Java runtime environment supports development with standard Java technologies, including JVM, Java servlets, and the Java programming language. It also supports any other language that uses a JVM-based interpreter or compiler such as JavaScript or Ruby. Python is also supported and App Engine offers a dedicated Python runtime environment, which includes a fast Python interpreter and the Python standard library. Both the Java and Python runtime environments are built to ensure that web applications run quickly, securely and without any interference from other applications running on App Engine.

App Engine provides its own storage solution called Datastore which utilizes BigTable as the method for storage. BigTable is not a relational and SQL compatible database and requires a different approach for storing and

retrieving data compared to a conventional SQL database. The advantages of BigTable is that it is fast and can scale to large tables and loads.

Google Datastore authors describe it as being "a sparse, distributed multi-dimensional sorted map", sharing characteristics of both row-oriented and column-oriented databases. It performs queries over data objects, known as entities. An entity has one or more properties, named values of one of several supported data types. A property can be a reference to another entity. The datastore supports executing multiple operations in a single transaction, and roll back the entire transaction if any of the operations fail. This type of feature is especially useful for distributed web applications.

Google currently supplies two standard Java interfaces for interacting with the Datastore, JDO (Java Data Objects), JPA (Java Persistence API), a low-level API is also available to allow developers direct access and the possibility to develop new interfaces. These interfaces allow developers to manage relational data in applications and include mechanisms that are meant to be used when trying to define classes for data objects and for performing queries.

JDO uses annotations on Java classes (POJO's) to describe how instances of the class are stored in the Datastore as entities, and how entities are recreated as instances when retrieved from the datastore. At the moment Datastore supports the use of JPA version 1.0. JPA also requires the use of annotations as in JDO but it also requires a `persistence.xml` file to be added which indicates to App Engine how to use the Datastore with this specific application. In order to make use of the App Engines persistence capabilities all objects used must be serializable.

All these features make App Engine a very attractive solution for deploying web applications that can scale without manual provisioning of computing resources.

Microsoft Azure

Azure is a Platform as a Service (PaaS) cloud offering from Microsoft [14]. It provides a platform for running mainly Windows applications and storing data in the cloud. These applications could be existing Windows applications that have been modified to run on cloud, or brand new ones written specifically for Microsoft Azure.

Developers can create applications for Microsoft Azure using familiar tools such as Visual Studio 2010. Azure applications are usually written using the .NET libraries, and are compiled to the Common Language Runtime (CLR). However, there is also support for the Java, Ruby and PHP languages.

Developers get a choice of language, but cannot control the underlying operating system or runtime. The platform provides a degree of automatic network configuration failover and scalability, but requires the developer to specify some application properties in order to do so.

The main components of the Windows Azure platform are:

- Windows Azure: Provides a Windows-based environment for running applications and storing data on servers in Microsoft data centers.
- SQL Azure: Provides data services in the cloud based on SQL Server.
- Windows Azure platform AppFabric: Provides cloud services for connecting applications running in the cloud.

Microsoft Azure may be a good solution to deploy existing applications for Windows and .NET platform. However, it raises the question on how well applications that have not been designed for the cloud can be scaled in the first place.

Heroku

Heroku [30] is a cloud application platform for the Ruby programming language. It was founded in 2007 by Orion Henry, James Lindenbaum, and Adam Wiggins. Heroku architecture consists of six components: HTTP reverse proxy, HTTP cache, routing mesh, dyno grid, SQL database with replication, and memory cache.

HTTP reverse proxy is the entry point for all requests coming into the platform. These front-end servers are used to manage DNS, load balancing, and fail-over. Heroku uses Nginx (`http://wiki.nginx.org/Main`) as its proxy.

All requests go through a HTTP cache (Varnish) [55]. If the requested content is available in the cache (a hit), the cache responds immediately and the request never reaches the application servers.

The routing mesh is a distributed pool of dynamic HTTP routers that balances requests across applications, dynos described below, tracks load, and intelligently routs traffic to available resources. The mesh easily handles frequent and instantaneous registration and de-registration of dynos. It is implemented using Erlang.

Actual application logic runs inside a dyno process. The number of dynos running for an application can be increased or decreased dynamically. According to the platform provider, it takes less than two seconds to start a

dyno for most applications. The dyno grid is spread across a large pool of server instances. The size of this pool varies with load.

Heroku provides a fully featured SQL database (PostgreSQL) for every application and an in-memory cache (Memcached). A dyno is a single process running Ruby code on a server in the dyno grid. In terms of computing power, four dynos are equivalent to one CPU-core on other systems. Each dyno is independent and includes the following layers:

- POSIX environment (Debian Linux).
- The Ruby VM, which then loads the application.
- The Thin application server.
- The Rack web server interface.
- Rails web framework. It is also possible to use other Rack-compliant frameworks.

We consider Heroku to be conceptually similar to the Google Application Engine (GAE) from a technical point of view. However there are several important differences worth mentioning:

- Heroku supports the Ruby programming language, while GAE supports Python and JVM languages.
- Heroku applications can use a SQL database.
- Heroku allocation of application threads is performed manually by the application provider. Thread allocation is automatic in GAE.

1.4.3 Software as a Service

Software as a Service (SaaS) can be considered as the most service oriented way to use cloud. So far, SaaS has proven useful as a business model among the cloud approaches. In SaaS, complete software systems that are ready-to-use is hosted in the software providers' servers and is offered to users to use over internet. The end-user willing to use this system pays the software provider a subscription fee for the service. This is completely different form the traditional way of software distribution and use, in which end-users need to purchase the license from the software provider and then install and run the software directly form on-site servers. Thus, SaaS allows some serious cost cutting for the companies (end-users) as they can avoid maintenance costs, licensing costs and the costs of the hardware required to run servers on-site.

Examples of such systems include the following:

- Salesforce (`http://salesforce.com`)
- Facebook (`http://www.facebook.com/`)
- Zynga farmwille game (`http://www.farmville.com/`)

Salesforce

Salesforce.com offers Customer Relationship Management (CRM) application services in the Software as a Service (SaaS) Industry [47, 57]. For end users, it offers services (applications) to industries and businesses of all sizes through online access, with minor implementation and no on-premise installation or maintenance of software or physical servers. These applications can be used to systematically record, store business data and to optimize different aspects of a company's business including sales, marketing, partnerships, and customer service. But due to the fact that CRM solutions should differ from one company to another, customization is obvious. That's why the cloud platform Force.com came into existence. Force.com [44] provides developers a platform to create data-oriented business applications which run against the salesforce.com database. This platform uses its own programming language called Apex. It has the following platforms [44]:

- Collaboration Platform: Chatter is a real time collaboration platform that brings together people, data, and content in a secure, private, trusted social framework. It facilitates creating customized profile across multiple business applications; provide real-time monitoring of business activities; secured content sharing; APIs to create new collaboration applications as add-ons; integration with other social networks like facebook and twitter.
- Development Platform: Development platform provides following services,
  - Database customization: Users can create and customize database relationships, formula fields, validation rules, reporting, tagging, auditing, and searches using the Web-based environment or the Eclipse-based IDE. It also generates user interface based on the data model defined which can be edited with page layout editor.
  - Programmable UI: User can build interfaces with customized behavior and look & feel. Force.com pages use a standard model-view-controller (MVC) design, HTML, and web technologies such

as CSS, AJAX, and Adobe Flash and Flex. It also has 60 predefined components that can be assembled with minimal coding in building-block fashion. Force.com also provides an IDE to implement cloud-based RIAs which can be deployed through the browser via the Adobe flash player, or directly to the desktop using the Adobe AIR runtime. It runs seamlessly online or offline while taking full advantage of the security, scalability, and reliability of Force.com.

- Programmable Cloud logic: Force.com code has similar syntax to Java and C#. Its Eclipse-based IDE can be used to create, modify, test and deploying applications.
- Visual process manager: Visual Process Manager, along with workflow and approvals, enables users to rapidly design and run any business process in the cloud without infrastructure, software, or code.
- Mobile deployment: Users can create complex mobile applications with point-and-click ease that work across BlackBerry, iPhone, Windows Mobile, and others with offline mobile access to key Salesforce CRM data.

- Cloud Infrastructure: Force.com is based on a multitenant architecture that makes it secure, reliable, and elastic. It has ISO 27001 security certification and used by nearly 60,000 companies including Cisco, Japan Post Network, and Symantec. It provides real time query optimization, upgradation, and scalability.

Facebook

Facebook is a social networking website that provides following services as SaaS to its users: Publisher (used to post information and messages which appear on the user's own Wall), Wall (space on each user's profile page that allows friends to see and post messages), Photo and video uploads, Gifts (virtual gift shop), Marketplace (allows users to post free classified ads in different catagories), Status updates, Events (to organize and notify the community with new events), Networks, groups and like pages, Chat, Pokes (to attract the attention of another user).

Facebook also provides platform that consists of a set of APIs and tools for developing social networking applications [38]. It is possible to develop Facebook applications using external server capacities from Cloud computing service providers like Amazon or Joyent [38].

In general, there are two types of applications on Facebook [22]:

- FBML Canvas applications: This kind of applications are rendered by Facebook using FBML (Facebook Markup Language). The application is hosted by the developer on their own server.

- IFrame Canvas applications and websites using Facebook: This kind of applications are usually rendered by the developer's server without using Facebook as an intermediary. IFrame Canvas applications are architecturally very similar to websites which incorporate data and widgets from Facebook.

Although the two types of applications do roughly the same things, they differ in the way user data is retrieved from Facebook, display static content and perform optimization.

The list of API's and SDK's currently provided by facebook platform [22] to develop applicaiton on Facebook are as follows:

- Core APIs:
  - Graph API: The Graph API can be used to read and write objects and connections in the Facebook social graph. Objects can be for example, album, photo, event, link, note, status message, video and so on. Whereas connections can be friend relationships, shared content, and photo tags. Every object in the social graph has a unique id and the data associated with the object can be fetched using that id.
  - Social plugins: Social plugins are the extensions of Facebook. These plugins are designed for not to share personal data with the sites on which they appear, but to show users with their activities on the facebook. These social plugins can be added to a site with a line of HTML code.

- Advanced APIs:
  - Facebook Query Language (FQL): FQL provides a SQL-style interface to query the data exposed by the Graph API. Batching multiple queries into a single call is possible. Query response format can be specified as either XML or JSON with the format query parameter.

  - Facebook Markup Language (FBML): FBML is used to build Facebook applications that can be hooked into several Facebook integration points, including the profile, profile actions, and canvas. FBML is HTML extension and is used in traditional FBML Canvas applications, and is rendered by Facebook directly.
  - XFBML is also a HTML extension provided by Facebook platform that can be used to incorporate FBML into an HTML page on a Facebook Connect (a set of API's to provide trusted connection between facebook and developer site) site or an iframe application.

- Facebook SDKs:

  - Android SDK (unofficial).
  - JavaScript SDK: JavaScript is used to access features of the Graph API. It also provides client-side functionality for authentication and sharing. Its recommended to load the SDK asynchronously in the site for better efficiency.
  - PHP SDK: It also supports access to Graph API.
  - Python SDK: This client library is designed to support the Facebook Graph API and the Facebook JavaScript SDK, which is the canonical way to implement Facebook authentication.
  - iPhone SDK: This mobile SDK is a Objective-C code that can be used to connect users' Facebook accounts with a mobile application. User authorization is required to fetch user profile data from the Graph API and to publish messages on user's wall. Facebook uses OAuth 2.0 protocol for authentication and authorization. There are also releases for PHP and Python SDKs for the Graph API to support mobile application development.

All the above Facebook SDKs are open source and are available on GitHub.

Zynga

Zynga is a social game developer, which develops games to play on social networks such as Facebook and MySpace, on mobile devices like the iPhone, on MSN games and my yahoo. Zynga makes some of the most popular social networking games that run on Facebook which includes Mafia Wars, Farmville, and Cafe World. The company's games attract 235 million users a month, which is more than half of Facebook's worldwide total of 400 million users [32].

1.4.4 Discussion

Since Cloud Computing has become a buzzword only relatively recently, there are no dominant technology designs yet that would be used predominantly. However, there are certain emerging trends, especially when approaching the development from programming perspective.

To begin with, although the paradigm shift from client-side programs to running the majority of applications in the cloud per se does not prescribe any new programming techniques, there have been some recent trends that deserve attention. However, in practice the trend seems to be that developers are increasingly using web technologies and scripting, as pointed out in [43]. In part, we believe that this is a consequence of the availability of increasing computing resources, but also the fact that modifiability, extendability and the access to ready-made web-enabled implementations simply are more important than the development of the most optimized implementation.

Finally, we fundamentally believe that it still remains a challenge to compose reliable software systems that are distributed and scalable in nature. Consequently, there is a need for improved understanding and better tools and methods for developing applications for any cloud programming platform.

1.5 Conclusions

This chapter has looked at cloud computing from both the application developers perspective as well as from looking at the technologies employed inside the cloud.

Virtualization is nowadays a mature technology that is heavily used both in clouds and in datacenters to remove the dependency of a server from the physical hardware it is running on, easing maintenance including hardware replacement and fail-over.

On the application development side it can be seen that the Web application development methodologies such as RESTful services and high level programming frameworks similar to the Google App Engine and Ruby on Rails are key building blocks for building Web applications hosted on the cloud.

Cloud computing can be used as a direct drop-in replacement for traditional applications using technologies such as Amazon EC2 and Amazon EBS. They basically provide virtual (Linux/Windows) machines and virtual (NAS) storage with traditional database products used as data storages. If the application does not have to scale to very large user numbers this is a viable alternative. It basically just uses clouds as virtualized hardware to

run traditional applications. If scalability to large user counts is needed, the other approach is to use scalable datastores such as, for example, Google App Engine Datastore, Cassandra, or Amazon SimpleDB to create scalable Web applications. In this framework the value add is that the cloud computing provider provides many of the pieces of the infrastructure to monitor and scale the applications to very large numbers of users but this requires some re-architecting the applications for scalability. The use of Web application development frameworks is a good start as it uses a similar division of work between the application itself and the datastore (database) used to store the data for the application.

To make cloud application deployment and administration cost-effective for large scale applications, scalable cloud based systems are generally architected in a shared-nothing architectural style where loosing any single server due to hardware failures will not effect the behavior of the cloud applications, as the cloud infrastructure will reconfigure the load of the failing server to the remaining servers. The key techniques to achieve this is to have high redundancy in the datastore system for fault tolerance, and to never store any persistent data on the servers themselves: all data on the servers is just cached or preprocessed contents of the real application data that is stored in the datastore.

When selecting if and how to employ the cloud technologies one should consider the scalability needs of the developed application as this gives requirements on the employed development and datastore methodologies. Many of the advanced cloud technologies are still in active development, and clear suggestions on which technologies to employ for which applications are not yet straightforward. Also the financial issues of cloud computing need to be considered, as common cloud pricing structure (pay-as-you-go) is at its best for handling computing loads that are bursty and hard to predict in advance and therefore carry investment risks in server and datacenter capacity planning.

References

[1] *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*. USENIX Association, 2006.

[2] *Amazon Elastic Block Store*. `http://aws.amazon.com/ebs/`.

[3] *Amazon Elastic Compute Cloud*. `http://aws.amazon.com/ec2/`.

[4] *AMD Virtualization (AMD-V^{TM}) Technology*. http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx.

[5] H. Amur et al. "Robust and Flexible Power-Proportional Storage". In: *ACM Symposium on Cloud Computing (ACM SOCC)*. 2010.

[6] M. Armbrust et al. "A view of cloud computing". In: *Commun. ACM* 53.4 (2010), pp. 50–58.

[7] M. Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report UCB/EECS-2009-28. Available from: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf. University of California at Berkeley, Electrical Engineering and Computer Sciences, 2009, p. 23.

[8] L. A. Barroso and U. Hölzle. "The Case for Energy-Proportional Computing". In: *IEEE Computer* 40.12 (2007), pp. 33–37.

[9] C. Baun and M. Kunze. "Building a private cloud with Eucalyptus". In: *E-Science Workshops, 2009 5th IEEE International Conference on*. 2009, pp. 33 –38. DOI: 10.1109/ESCIW.2009.5408006.

[10] M. Burrows. "The Chubby Lock Service for Loosely-Coupled Distributed Systems". In: *OSDI*. USENIX Association, 2006, pp. 335–350.

[11] T. D. Chandra, R. Griesemer, and J. Redstone. "Paxos made live: An engineering perspective". In: *PODC*. Ed. by I. Gupta and R. Wattenhofer. ACM, 2007, pp. 398–407. ISBN: 978-1-59593-616-5.

[12] F. Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *OSDI*. USENIX Association, 2006, pp. 205–218.

[13] F. Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *ACM Trans. Comput. Syst.* 26.2 (2008).

[14] D. Chappell. *Introducing The Windows Azure Platform*. Tech. rep. Microsoft Corporation, 2009.

[15] *Comparison of platform virtual machines*. http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines.

[16] B. F. Cooper et al. "PNUTS: Yahoo!'s hosted data serving platform". In: *PVLDB* 1.2 (2008), pp. 1277–1288.

[17] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI*. 2004, pp. 137–150.

[18] J. Dean and S. Ghemawat. "MapReduce: Simplified data processing on large clusters". In: *Commun. ACM* 51.1 (2008), pp. 107–113.

[19] G. DeCandia et al. "Dynamo: Amazon's highly available key-value store". In: *SOSP*. Ed. by T. C. Bressoud and M. F. Kaashoek. ACM, 2007, pp. 205–220. ISBN: 978-1-59593-591-5.

[20] H. Douglas and C. Gehrmann. "Secure Virtualization and Multi-core Platforms State-of-the-Art report". In: *SICS Technical Report* T2009:14A (2010), pp. 1–71. URL: `http://soda.swedish-ict.se/3800/`.

[21] *Extend Your Virtual IT Infrastructure With Amazon Virtual Private Cloud*. Tech. rep. Amazon Web Services, 2010.

[22] *Facebook Developers*. `http://developers.facebook.com/docs/`. 2010.

[23] R. T. Fielding and R. N. Taylor. "Principled design of the modern Web architecture". In: *ACM Trans. Internet Techn.* 2.2 (2002), pp. 115–150.

[24] A. Fox et al. "Cluster-Based Scalable Network Services". In: *SOSP*. 1997, pp. 78–91.

[25] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google file system". In: *SOSP*. Ed. by M. L. Scott and L. L. Peterson. ACM, 2003, pp. 29–43. ISBN: 1-58113-757-5.

[26] S. Gilbert and N. A. Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". In: *SIGACT News* 33.2 (2002), pp. 51–59.

[27] *Google App Engine*. `http://code.google.com/appengine/`.

[28] N. Gude et al. "NOX: towards an operating system for networks". In: *SIGCOMM Comput. Commun. Rev.* 38.3 (2008), pp. 105–110. ISSN: 0146-4833. DOI: `http://doi.acm.org/10.1145/1384609.1384625`.

[29] T. Härder and A. Reuter. "Principles of Transaction-Oriented Database Recovery". In: *ACM Comput. Surv.* 15.4 (1983), pp. 287–317.

[30] *Heroku Homepage*. `http://heroku.com/`.

[31] B. Hindman et al. *Nexus: A Common Substrate for Cluster Computing*. Tech. rep. UCB/EECS-2009-158. EECS Department, University of California, Berkeley, 2009. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-158.html`.

[32] *Information Week*. http://www.informationweek.com/. 2010.

[33] *Intel® Virtualization Technology*. http://www.intel.com/technology/virtualization/technology.htm?iid=tech_vt+tech.

[34] *KVM Homepage*. http://www.linux-kvm.org/page/Main_Page.

[35] J. Laine. *Cloud Storage Systems in Telecom Services*. Master's Thesis, Aalto University, School of Science and Technology, Degree Programme in Computer Science and Engineering. 2010.

[36] A. Lakshman and P. Malik. "Cassandra - A Decentralized Structured Storage System". In: *LADIS 2009: The 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware*. 2009.

[37] L. Lamport. "The Part-Time Parliament". In: *ACM Trans. Comput. Syst.* 16.2 (1998), pp. 133–169.

[38] A. Lenk et al. "What's Inside the Cloud? An Architectural Map of the Cloud Landscape". In: *ICSE '09: Proceedings of the Workshop on Software Engineering Challenges in Cloud Computing*. Available from: http://www.icse-cloud09.org/cloud-dashboard. 2009.

[39] N. McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (2008), pp. 69–74. ISSN: 0146-4833. DOI: http://doi.acm.org/10.1145/1355734.1355746.

[40] P. Mell and T. Grance. *The NIST Definition of Cloud Computing v15*. Version 15 available from: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc.

[41] D. Nurmi et al. "The Eucalyptus Open-Source Cloud-Computing System". In: *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. ISBN: 978-0-7695-3622-4. DOI: http://dx.doi.org/10.1109/CCGRID.2009.93.

[42] D. A. Patterson. "Latency lags bandwith". In: *Commun. ACM* 47.10 (2004), pp. 71–75.

[43] L. D. Paulson. "Developers Shift to Dynamic Programming Languages". In: *Computer* 40.2 (2007), pp. 12–15. ISSN: 0018-9162. DOI: http://dx.doi.org/10.1109/MC.2007.53.

[44] force.com platform. In: http://www.salesforce.com/platform/cloud-platform/, 2010.

[45] D. Pritchett. “BASE: An ACID Alternative”. In: *ACM Queue* 6.3 (2008), pp. 48–55.

[46] *QEMU Open Source Processor Emulator*. http://wiki.qemu.org/Index.html.

[47] “SalesForce products”. In: http://www.salesforce.com/crm/products.jsp, 2010.

[48] I. Stoica et al. “Chord: A scalable peer-to-peer lookup protocol for internet applications”. In: *IEEE/ACM Trans. Netw.* 11.1 (2003), pp. 17–32.

[49] I. Stoica et al. “Chord: A scalable peer-to-peer lookup service for internet applications”. In: *SIGCOMM*. 2001, pp. 149–160.

[50] M. Stonebraker. “SQL databases v. NoSQL databases”. In: *Commun. ACM* 53.4 (2010), pp. 10–11.

[51] M. Stonebraker et al. “MapReduce and parallel DBMSs: Friends or foes?” In: *Commun. ACM* 53.1 (2010), pp. 64–71.

[52] N. Tolia et al. “Delivering Energy Proportionality with Non Energy-Proportional Systems - Optimizing the Ensemble”. In: *HotPower*. Ed. by F. Zhao. USENIX Association, 2008.

[53] *TrustZone*. http://www.arm.com/products/processors/technologies/trustzone.php.

[54] L. M. Vaquero et al. “A break in the clouds: towards a cloud definition”. In: *SIGCOMM Comput. Commun. Rev.* 39.1 (2009), pp. 50–55. ISSN: 0146-4833. DOI: http://doi.acm.org/10.1145/1496091.1496100.

[55] *Varnish Homepage*. http://varnish-cache.org/.

[56] S. Wardley, E. Goyer, and N. Barcet. *Ubuntu Enterprise Cloud Architecture*. Tech. rep. Canonical, 2009.

[57] “wikinvest”. In: http://www.wikinvest.com/wiki, 2010.

[58] *Wikipedia page for VMWare ESX*. http://en.wikipedia.org/wiki/VMware_ESX.

2 The Social Devices Platform: An Infrastructure for Social Devices in a Mobile Cloud

Timo Aaltonen[1], Varvana Myllärniemi[2], Niko Mäkitalo[1],
Tomi Männistö[2,3], Jari Pääkkö[2], and Mikko Raatikainen[2]

[1]Department of Pervasive Computing
Tampere University of Technology, Tampere, Finland
Email: firstname.lastname@tut.fi

[2]Department of Computer Science and Engineering
Aalto University, Espoo, Finland
Email: firstname.lastname@aalto.fi

[3]Department of Computer Science
University of Helsinki, Helsinki, Finland
Email: firstname.lastname@helsinki.fi

Abstract–Mobile devices are becoming the primary access point for users for various services and mobile cloud computing has emerged as a means of utilizing the advantages of cloud computing within mobile devices. Cloud computing, in general, hides the location of services, regardless of whether they are provided by the mobile devices or by backend services. Thus, mobile cloud computing is especially suitable in situations in which several mobile devices participate in synchronized behavior as the cloud can provide a unifying infrastructure for such a scenario in addition to content. In this chapter, we present an approach to how distributed, collaborative, and coordinated services among multiple mobile devices can be supported with a

The authors are listed in alphabetical order. The chapter is extended and combined from the authors' earlier publications in the WICSA workshop [30], the MUM conference [27], and the APSEC conferences [31][1].

cloud-based infrastructure. In particular, our contribution outlines an exemplar infrastructure called the Social Devices Platform, which delivers social, co-located multi-device services to users. The contribution combines research from different areas, such as mobile computing, service-oriented computing, pervasive computing, artificial intelligence, and human-computer interaction. The Social Devices Platform utilizes the cloud to store contextual data and to select, compose, and coordinate services provided by mobile devices, thus extending the cloud to mobile devices and making devices' resources available to the cloud.

Keywords-Social devices, mobile devices.

2.1 Introduction

Technological advances are redefining the ways that people behave and communicate, making digital communication increasingly important. Social media services, such as Facebook and Flickr, have created a new means for people to find like-minded friends and to communicate with others regarding their everyday activities. While various devices support remote connectivity, the communication of such devices still largely overlooks specific co-located social situations. While there are obvious advantages in supporting remote social interactions, we argue that it is important to support the face-to-face, co-located interactions between people. In practice, support then relies on mobile phones or other personal devices that are in proximity to each other. Similarly, as smart spaces are emerging to provide services between humans and computers in specific places, similar kinds of smart spaces could be formed in a mobile and ad-hoc manner between various nomadic devices.

While in cloud computing, applications, and computing resources are provisioned and released dynamically to address dynamic needs, mobile cloud computing extends the cloud paradigm to mobile devices thus offering a means for connecting mobile devices to web-based services and to each other. The concept of a so-called mobile cloud in its simplest form refers to accessing cloud computing resources from a mobile device [21], but often the ability to share more advance services among mobile devices in the same cloud is assumed [36], even to the extent of delivering s unique user experience by shared and pooled resources. Mobile clouds are becoming a means for a technology platform for implementing social interactions and service sharing between mobile devices.

In this chapter, we describe the concept of *Social Devices* along with a prototype implementation called the *Social Devices Platform* (SDP). The

concept of Social Devices focuses on enriching local interaction by a means of technology such as audio or picture relevant to the specific context. The interaction can be between devices and between humans and technology, whereas the enrichment can rely almost solely on the autonomous interaction within technology such as between mobile phones. Therefore, humans and proactive, context-sensing mobile devices form a new kind of socio-digital system where the mobile devices are active participants and can initiate interaction between other devices as well as with people. As is traditional in social interactions, the interaction between different co-located physical nodes is essential regardless of if such nodes are humans or devices. In short, the setting of Social Devices resembles socially interactive pervasive computing between different nodes or an ad-hoc smart space.

The rest of this chapter is structured as follows. Section 2.2 gives the motivation, provides a running example, and lists the key characteristics of Social Devices. Section 2.3 describes the implementation of the SDP. Section 2.4 provides a comparison to related work and Section 2.5 gives a more general discussion. Conclusions are drawn in Section 2.6.

2.2 Social Devices

In this section, we summarize certain key characteristics related to the concept of Social Devices [27]. These characteristics help to distinguish Social Devices from other related concepts and give an understanding of the inherent properties of Social Devices. Furthermore, these characteristics help in forming the basic requirements for the actual implementation of the concept. We define constructs, give the motivation, provide a running example, and finally present the requirements and characteristics that are essential to Social Devices.

2.2.1 Motivation and Background

The primary objective of Social Devices is to enrich various kinds of co-located interactions. When people meet face-to-face, Social Devices can augment the social interaction that takes place. Also, device-to-device interactions can be enriched: Social Devices can make otherwise invisible device interaction explicit to users. Finally, Social Devices enable more intuitive interaction between users and devices, especially when the devices need to give feedback to the users.

Any interaction as a whole that is enriched or enabled by Social Devices is called an *action*. A key characteristic of actions is to provide human

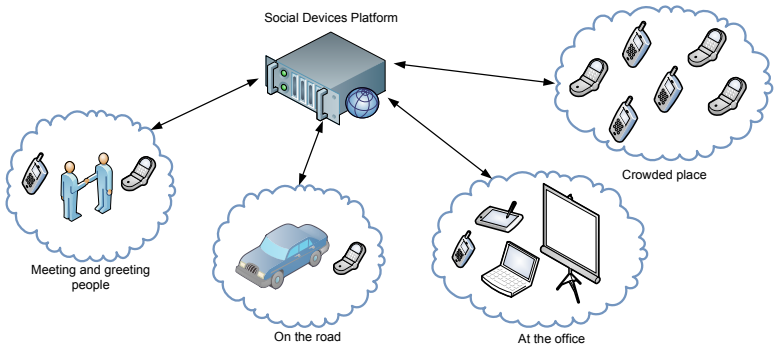


Figure 2.1: Examples of using the Social Devices Platform

users some visible value intelligently. An action is, in practice, a sequence or behavior that is carried out by the Social Devices. The actions can be performed autonomously by the devices, although the actions can require human interaction. The actions differ largely in nature, but typically involve interaction of heterogeneous devices near each other in a situation that is not mission-critical.

Figure 2.1 illustrates various actions that can take place between Social Devices. Meetings and greetings between people offers plenty of situations for Social Devices. For example, the devices of two businessmen can exchange contact cards and at the same time announce each other's information aloud. Social Devices may also help in situations where one cannot remember a person's name by greeting him or her aloud on the street. Social Devices can also make invisible device interactions more explicit; for example, a laptop and a mobile phone can speak about their synchronization progress. As another example, a mobile phone tell to a car navigator where to go in addition to setting the destination automatically based on a calendar entry. The user is then aware of the ongoing navigation and can interrupt if necessary by, for example, specifying a new destination address. Interaction can also involve a large number of devices; for example, mobile devices can make a massive wave of flashing screens at a rock concert. Finally, actions can be instructive by nature, such as in the case when an elevator instructs a visitor to find the correct meeting room based on an upcoming entry in the mobile device's calendar.

A precondition of Social Devices is the availability of computationally capable smart devices that are aware of, and can interact with, other devices and users in their proximity. A prime example of a Social Device is a mobile phone, which most people carry everywhere, although many other

kinds of devices can become social. Some devices are highly personal, such as mobile phones, and some are impersonal, such as meeting room screens. Some devices are very stationary, such as elevators, and others do not have a fixed location but move around other devices freely, such as family laptops or tablets. Finally, the devices have heterogeneous resources with varying levels of quality. These resources include speakers, microphones, and screens.

In Social Devices, actions are executed in a very dynamic environment. Potential social situations arise as people interact and carry out their everyday tasks. Since mobile devices move in and out of the range of other devices, the devices that can participate in an action change constantly. Furthermore, device states change dynamically: they can be turned to silent mode or run out of power.

Several other research areas are related to the concept of Social Devices. Social Devices resemble smart spaces [39] because both utilize the proximity of devices to provide services. In contrast to smart spaces, Social Devices are not tied to a particular location or particular devices, and their actions focus on providing user-visible behavior rather than background services. However, some Social Devices can be stationary, such as a meeting room screen, and then they resemble more a smart space. In contrast to pervasive computing [39, 43] or the Internet of Things [3], Social Devices provide human-visible actions, rather than performing actions and forming compositions in the background. In fact, central to Social Devices is the interruption of social interaction that is, in a sense, just the opposite of pervasive computing that has the vision of indistinguishable form or minimal user distraction [39]. Actually, Social Devices can be used to complement pervasive computing as they can make the background tasks performed by pervasive computing visible. Nevertheless, both pervasive computing and Social Devices rely mainly on smart devices in an environment. In contrast to Web Services or other traditional service-oriented architectures (SOA) [33], the key for Social Devices is to utilize resources and capabilities of various physically co-located devices and the proximity of the devices in a user-observable manner. Finally, a key characteristic in Social Devices compared with many other similar approaches is that instead of devices autonomously forming collaborating networks and agreeing on operations, the users should always have control of the actions in which their devices participate. Furthermore, advanced users could create new actions to make the devices interact.

2.2.2 Running Example

We use PhotoSharing as a running example to explain the motivation and the basic concepts of Social Devices. In our PhotoSharing example, Alice meets

Figure 2.2: Alice and her friends at a cafeteria.

Bob, Carol, and Dan at a cafeteria (Figure 2.2). When Alice joins the table, her phone recognizes that her friends are nearby and suggests that she could share and present the photo album from her latest trip. Although Alice has already previously added her travel photos to Flickr, the cafeteria provides a socially convenient moment for talking about the trip. Instead of squeezing in to see the photos on Alice's screen, or everyone individually browsing the photos, the friends can see the photos on their own devices in a synchronized manner as Alice comments on each photo.

In more detail, what happens in the PhotoSharing example is that when Alice approaches her friends, her phone recognizes and notifies her of the possibility of sharing the new photos. This notification may utilize several forms of interaction, such as talking aloud, vibration, and screen dialogs. After Alice indicates that she wants to share her photos, Bob and Carol join the session with the confirmation dialog that appeared on their devices. Since Dan's phone is running out of power, he decides not to join. Alice browses her photos and selects the ones to be viewed. At the same time, the photos appear in Bob's and Carol's devices, while Alice talks about her trip. Alice has also recorded audio related to some photos, which is played by Bob's device as it has the best loudspeaker.

The PhotoSharing example highlights the key aspects of Social Devices elaborated in more detail in the following section. All functionality is built

around actions: an action provides proactively initiated, typically user-visible, coordinated behavior, such as photo browsing, in a synchronized manner between multiple devices. To participate in an action, each user is assumed to have a device to which the necessary capabilities to join in an action are installed. Actions can start proactively, even without user assistance, but the user can control whether or not the action continues. Therefore, actions can define triggers, upon receipt of which the action is executed, or at least attempted. In addition, the necessary preconditions must be satisfied before any action can take place.

2.2.3 Requirements and Characteristics

The running example of PhotoSharing highlighted several requirements and characteristics of Social Devices. In the following, we discuss these in more detail.

The devices need to have identities, and the SDP needs to keep track of these identities. The identity of a device can be associated with a person, such as Alice and her phone. The need for device identities is highlighted by the fact that devices collaborate in a way that is often personal to the user and closely related to user preferences, and the collaboration takes place in a social situation where devices are identifiable rather than background services. In the PhotoSharing example, all users use their own personal device. The need for identities arises from the fact that users have different preferences for their mobile devices such as not being too verbose; in other words, not to speak aloud about the synchronization's progress. From a technical point of view, the approach for device identity management in the SDP is to use a centralized registry where devices need to register their identity as well as other properties discussed below. Moreover, as users are essential for Social Devices, the platform needs to maintain user models that indicate which device is the most personal for a user and the other kinds of devices that a user has. The information about the most personal devices can be used to indicate the proximity of users. The user model can also contain personality and social network information that can, for example, be used for suggesting actions in different contexts.

Devices need to know the other nearby devices because proximity is a precondition for Social Devices in general: Co-located social interaction requires that users (and devices) are near each other; for example, Alice is near her friends. Thus, the proximity of the devices needs to be discovered with respect to each device that will participate in an action. The SDP maintains proximity information in the form of a weighted graph, called a *proximity graph*, where nodes denote devices and edges denote the mutual distance

```
class PhotoSharing(Action):
    @actionprecondition
    def precondition(self, src, sinks, albumURI):
        return areFacebookFriends(src, sinks) and\
               haveCapabilities(sinks, ['Screen','Notification']) and\
               src.hasCapabilities(['ImageBrowser','Notification']) and ...

    @actionbody
    def body(self, src, sinks, albumURI):
        message = 'Hey ' + src.getOwnerName() + ' let\'s share images'
        if src.notification.notify(message, ['Yes', 'No']) == 'No':
            return
        # notifying the sinks is omitted

        bestAudioDev = self.participants.chooseOne('AudioDevice')
        for image in src.imageBrowser.browse(albumURI):
            relatedAudio = image.getRelatedAudio()
            if bestAudioDev and relatedAudio:
                bestAudioDev.audioDevice.play(relatedAudio)

            startParallelExecution()
            for sink in sinks:
                sink.screen.show(image)
            endParallelExecution()

        # tearing down omitted for brevity

    @classmethod
     def getTriggers(cls):
        return ['PhotoSharingTrigger']

class PhotoSharingTrigger(Trigger):
    def __init__(self, albumURI, sender):
        Trigger.__init__(self, sender)
        self.albumURI = albumURI
```

Figure 2.3: The definition of the PhotoSharing action and PhotoSharingTrigger

of these devices, which discover and measure the signal strength of other nearby devices and report this to a centralized server that calculates the mutual distances of the devices and updates the proximity graph. In the current implementation, the measurement of proximity is simply based on Bluetooth signal strength. Thus, the approach allocates much of the computing to the server side, keeping the devices simple rather than relying on the devices to continuously discover, measure, and keep track of the proximity.

In order to know what kinds of different collaborative actions are available between devices, Social Devices utilize *action descriptions* that describe the actions in more detail. The action descriptions are managed centrally rather than relying on ad-hoc or autonomous behavior. The objective is that developers can easily write new action descriptions. Actions are described in two parts: the *action body*, that is, the description of what the devices

should do during an action, and the *action precondition*, that is, the description of what is expected from the devices participating in the action. The PhotoSharing action description is described in Figure 2.3.

In order for the action to actually do anything meaningful, it needs to be known what the devices can do. This is achieved by the concept of a *device capability*. While actions define behavior, the capabilities of devices (e.g., a capability to view a picture) provide the functionality. When writing actions, a developer can develop new or reuse existing capabilities. For example, an action *PhotoSharing* may need to operate on devices that can show and browse images. These expectations are characterized via *interfaces* that the devices need to fulfill; for example, the ability to show images is characterized with a *Screen* interface. Hence, the interface specifications for services in Social Devices are what WSDL is for Web Services: an interface specifies a contract, while it is the responsibility of a device to implement the interface. A Social Device can provide any number of interfaces: The owner of a social device can install and enable different capabilities to her own liking.

Besides capabilities and proximity, the devices have different kinds of properties and states. For instance, a device can be running out of power, or be in a silent or power-saving mode. Hence, there are not only the static capabilities of devices, but also dynamically changing device properties. Such properties that affect whether a device can participate in an action are monitored via device *states*; for example, the integer-valued state *batteryLevel* can indicate the remaining power. For this purpose, the devices continuously report their states to a central server, which then has all the necessary information to decide whether an action can take place.

The condition when actions are executed needs to be decided automatically at runtime because social devices are autonomous in the sense that direct user interaction is not necessarily needed to start an action. Continuous searching for devices that satisfy the action precondition is computationally hard. Therefore, the SDP has taken the relatively simple approach that a *trigger* marks the need for action execution. In particular, certain actions are attempted to be scheduled when a certain trigger is received: For example, PhotoSharingTrigger is associated with the PhotoSharing action. A trigger can be caused by an external event or a change in the proximity group. In the running example, PhotoSharingTrigger was created when Alice uploaded her photos to Flickr. Triggers and related actions are managed by a centralized server that also receives the triggers and decides whether and which action to initiate. Thus, the intelligence for making decisions about initiated actions is built and refined in one location on the server. Consequently, the devices only need to identify the condition to raise a trigger. Although anything can send a trigger to the server, the server does not necessarily initiate an action.

In addition to identifying when to execute an action, one needs to decide at runtime which concrete devices will participate in the action; this is called *scheduling*. The main task is to relate the potentially available devices to the preconditions stated in the action description and to decide the roles to which the devices should be assigned. An action precondition may require certain capabilities from the devices, such as the implementation of certain interfaces. For example, the PhotoSharing action requires that Bob's, Carol's, and Dan's devices can receive and show images by using the *Screen* interface. When a trigger for action execution is intiated, several devices can potentially fulfill the preconditions. The same set of devices can participate in an action in different roles; for example, with respect to who is sharing the photos. Thus, the configuration problem in Social Devices is as follows: find an action that matches a runtime trigger and a set of devices and device roles that fulfill the action preconditions. However, in practice, the configuration problem is often eased by the action preconditions, which can restrict the number of potential devices that can participate in the action. For instance, the preconditions can require that the device that initiated the trigger should participate in a certain role in the action and that other participating devices are in its proximity.

Finally, the action is executed. For this, the action coordination or orchestration problem requires defining a control flow for a set of independently operating devices. Since each device has been designed to run its own software — apart from mission-specific coordination tasks such as making a phone call on a cell phone — additional features are needed to manage the interactions taking place between the devices. In the SDP, in order to create a generic system where various cooperation possibilities are available, we decided to use a coordination approach by defining a platform into which coordination is built.

To summarize our high-level solutions, we rely on a centralized approach rather than device autonomy and intelligence. This is in contrast to, e.g., agent-based systems or ad-hoc networks. For example, proximity management and decisions about actions and device configurations are not the burden of the devices. As a consequence, quite simple devices can also be Social Devices since complex calculation or special resources are not needed.

2.3 The Social Devices Platform

The concept of Social Devices has been implemented in a supporting infrastructure termed the Social Devices Platform. The SDP implements all the necessary concepts and tasks related to Social Devices. In the following, we

describe the overall architecture, and the different responsibilities in more detail.

2.3.1 Overall Architecture

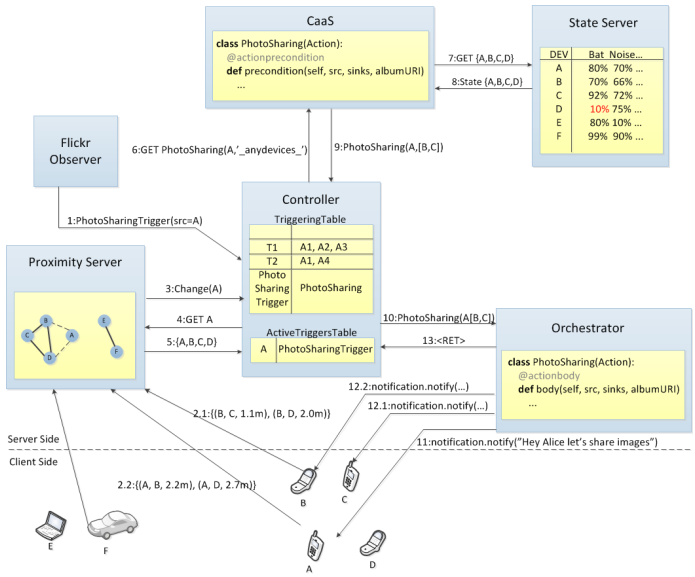


Figure 2.4: The SDP architecture and the interactions in the photo sharing example.

The architecture of the SDP (Figure 2.4) consists of clients and a number of cloud services running on the Amazon cloud. The client side currently consists of Android devices and Python-capable devices, such as Linux laptops and Meego phones. The interactions in Figure 2.4 show the key services and exemplify how the photo sharing example is executed; Alice and her friends are denoted with *A*, *B*, *C*, and *D*.

At the client side, the responsibilities of SocialDevicesClient include reporting the device capabilities and states that are needed for evaluating preconditions, and discovering and reporting other devices in the proximity.

At the server side, ProximityServer is responsible for maintaining information about the proximity and notifying about changes in proximity. StateServer is responsible for managing the device registry and action registry,

whereas Controller is responsible for the triggering table and the active triggers table. In addition, Controller takes care of initiating the scheduling process i.e. other services report triggers to it. CaaS (Configurator-as-a-Service) is responsible for assigning devices to roles by utilizing an inference engine. Finally, Orchestrator is responsible for managing the actual execution of actions between Social Devices.

For example in Figure 2.4, when ProximityServer reports a proximity change in Alice's device, scheduling can be initiated since FlickrObserver has already sent a trigger. Controller retrieves the proximity group, and assigns device A to the src role in the PhotoSharing action. Thereafter, Controller utilizes CaaS. Based on the device states, CaaS knows that the battery of D is dry, so only B and C are assigned to the action. After the scheduling and device assignment, Controller sends the action along with the participants to Orchestrator, whose task is to orchestrate the execution of the action body. The orchestration is based on utilizing the devices' interfaces and their operations.

2.3.2 ProximityServer: Managing Proximity Information

The responsibility of the ProximityServer is to maintain proximity information of the devices in the form of a graph, where nodes represent the devices, and edges have their mutual distance as an attribute. An edge gets stale when a predefined time has passed, after which the edge is removed from the graph. This way the graph is a model of the real world proximity information. The proximity graph can be used for searching for sets of devices in each other's proximity.

In Figure 2.4, the proximity graph is partitioned into (B, C, D) and (E, F) based on earlier measurements. When Alice gets close to her friends (message #2.2), the new edges $(A, B, distance = 2.2)$ and $(A, D, distance = 2.7)$ are added to the graph. When such significant changes occur in the proximity graph, ProximityServer notifies Controller (message #3).

2.3.3 StateServer: Managing Device States and Properties

StateServer is responsible for managing data about device properties, such as screen type, and about device state or context, such as ambient noise and battery level. Thus, the properties are more static characteristics, while states change frequently. The devices report their properties and state to

StateServer. The interaction between the devices and StateServer is omitted from Figure 2.4 for clarity. State data can be used in action preconditions to constrain an action role only to a device with specific state values.

Currently, StateServer does not store historical state data about devices. In other words, StateServer only provides the current state of each device. This is sufficient for simple actions, but more complex actions can require contextual inferences based on the device state history. In this case, each state value must be stored with an associated timestamp.

In the SDP, the actual state values are stored as simple key-value pairs, where the key is the name of the state and the value is the actual state value for the device. The SDP is, however, independent of the storage mechanism and the state values could equally well be stored in a graph database or some other form of database.

2.3.4 Controller: Triggering and Scheduling of Actions

Controller is the main hub of the SDP basically taking care of the interaction within the server side. The starting point of, and the invocation for, Controller is a *trigger* that causes Controller to start interactions that can result in action execution. In order to utilize triggers, there needs to be an explicit definition of the triggers and a mapping between triggers and actions. Figure 2.3 illustrates the definition of PhotoSharingTrigger, which is associated with the PhotoSharing action using the getTriggers method. In general, there can exist a many-to-many relation between triggers and actions. The triggers and the corresponding actions need to be registered in a *triggering table*.

Based on an incoming trigger, an action needs to be selected from the set of actions related to a trigger. This process, called *scheduling*, involves finding an action whose precondition is satisfied by a set of devices. In some cases, it may be possible to assign some devices to action roles in advance, whereby the scheduling process can be eased. For instance, the source of a trigger might be assigned to a specific role in an action. This kind of pre-assignment can be made by Controller.

There may be a need to manage the lifespan of incoming active triggers. For this purpose, all incoming triggers can be stored in an *active triggers table* to wait for rescheduling if no action preconditions are immediately satisfied. Different strategies can be employed for rescheduling active triggers. For instance, scheduling can be reattempted after a certain amount of time or after a predefined change in a state. Also, any failed triggers can simply be discarded.

To be able to recognize various triggers, the architecture can be integrated with several observer services that are responsible for sending the triggers

to Controller. Despite actually not being part of Controller, such observers are in close relation with it. Figure 2.4 illustrates how FlickrObserver sends PhotoSharingTrigger on behalf of Alice's device when Alice uploads her photos to Flickr. At this point, Controller attempts to schedule the PhotoSharing action, but since Alice is at home with no friends nearby, the scheduling fails. Therefore, Controller stores Alice's device A and the corresponding active trigger in the ActiveTriggers table and starts waiting for changes in Alice's proximity. This is an example of a state-based rescheduling strategy described earlier in this section.

2.3.5 CaaS: Configuration of Actions

As mentioned in Section 2.2.3, the SDP needs to determine which devices can participate in an action. In the architecture, this is the responsibility of CaaS (Configuration-as-a-Service). CaaS is based on existing research in software product configuration [19, 28], where the task is to find a valid software configuration given a configuration model and configuration selections. A configuration model encapsulates the architectural variability in software, while the configuration selections provide requirements for the configuration.

In Social Devices, many different factors affect which action can be executed and whether a device can participate in an action. Variability arises from multiple sources:

- New devices are registered in the SDP.
- Different devices have different capabilities.
- Different devices have different states.
- The devices that are in proximity of each other change.
- Several alternative actions can be executed.
- Devices can be in different roles within an action.
- Devices may or may not satisfy the constraints of an action role.

Since, the variability changes dynamically, the configuration model must be generated dynamically. For generating a configuration model dynamically, CaaS requires a list of actions as input. More specifically, the action preconditions encode the constraints for the action roles and can be used to determine which devices can be assigned to what roles.

In addition to the configuration model, CaaS also needs the configuration selections. The selections are generated dynamically based on a list of devices and the capabilities and state values of the devices. Both the list of actions and devices are sent to CaaS by Controller.

When the configuration model and selections have been generated, they are given as input to an inference engine that computes a valid configuration, i.e., an action with a set of devices assigned to the action roles.

Currently, CaaS only returns an arbitrary, valid configuration. In practice, however, not all configurations are equal in terms of quality of service (QoS). In fact, the perceived QoS of a configuration is affected by user preferences, device features, and context. Accordingly, CaaS can be augmented with recommendation techniques that take user preferences, device capabilities, and device context into account when making device selections for particular action roles [31].

Configuration problems can be computationally expensive due to the combinatorial explosion of the number of configurations. In the worst case, the time required to find a valid configuration grows exponentially with the number of variability points. In Social Devices, however, variability is limited by the number of devices in proximity and the amount of roles in an action. In fact, our initial test runs indicate that finding a valid configuration in the context of the SDP seems feasible using a standard PC.

For instance, finding a valid configuration for 1000 devices and two actions with two or three roles, takes approximately two seconds. In general, we can assume that the size of a proximity group is typically less than 100. Additionally, since the triggering of actions in the SDP is not time critical and does not necessarily involve user interaction, the response times only need to be within a few seconds at most. As such, the response times for finding a valid configuration are feasible.

2.3.6 Orchestrator: Execution of Actions

Orchestrator is a centralized server that is responsible for executing the actions or the action body parts to be more specific. The action executions are based on the configuration of devices that has been computed by CaaS and returned to Controller. Controller starts action execution by sending the configuration to Orchestrator using a REST API. When the action execution begins, the participating devices are reserved for the duration of execution. This design decision was made to offer a more transactional way for the devices to behave as they can rollback the changes in case of an error. It is also clearer to the users if their devices are participating to one action at a time. Finally, Orchestrator monitors the clients, and in a centralized man-

ner, handles exceptions raised by the devices when the devices fail or stop responding.

Invoking the operations of devices is done in a synchronized way, allowing the operation calls to act like any operation calls in Python. For the developers, this allows an illustrative way to coordinate the devices and keep the action body clear. After finishing the body execution, Orchestrator notifies Controller.

The Orchestrator uses a synchronized way for calling the actor devices' interface methods. This design decision allows the platform to relay return values directly from the actor devices to an action execution process, and hence the operation calls act like regular method calls. This also makes it illustrative for users to create their own actions and develop new services.

The interfaces have their counterparts on both the server side and the device side. On the server side, the interfaces are implemented in Python and are therefore easily callable from the action, as we are using Python as the coordination language (see Figure 2.3). On the device side, the implementations of these interfaces and their methods are coupled to the platform and its programming language (see Figure 2.5). Moreover, because the way of implementing features varies among different platforms, the service implementations and the quality may vary as well. In other words, each actor device is responsible for its local operations.

2.3.7 SocialDevicesClient: The Client for Social Devices

To participate in actions, the devices need to have SocialDevicesClient installed and running. The architecture of the client (Figure 2.5) is modular and based on plugins. Three plugins are essential: OrchestratorPlugin participates in action execution by maintaining a connection to the Orchestrator service and invoking operations according to server coordination; ProximityPlugin measures periodically the Bluetooth signal strength of nearby devices and reports this information along with the Bluetooth MAC identities to ProximityServer, which then translates the information to distance; and StatePlugin reports contextual data and state values, such as battery level or GPS location, to StateServer. In contrast to the essential plugins, the capabilities are implemented as interface plugins; for example, the ImageBrowser capability is implemented as one plugin. In general, there can be any number of interface plugins.

The whole SocialDevicesClient is implemented as a native application for each environment. While this ensures, e.g., efficiency in terms of battery

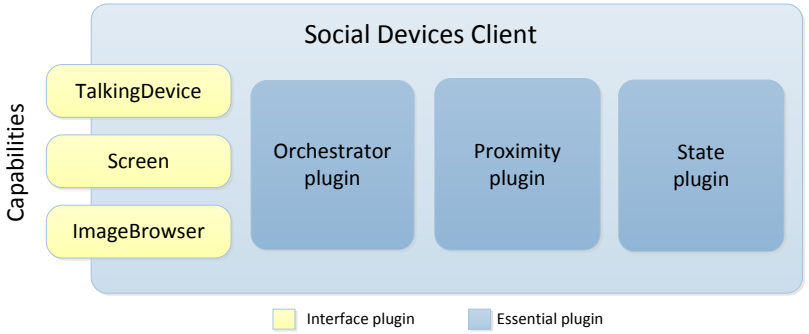


Figure 2.5: The SocialDevicesClient plugin architecture.

consumption, separate implementations of SocialDevicesClient and plugins are needed for different environments. However, an interface plugin can be general and used in different actions. The modularity of the client and the thin implementation of the essential plugins have allowed SocialDevicesClient to be light from the device point of view. Furthermore, the modularity allows users to have more control over their devices, as they can decide the capabilities their devices should provide.

2.4 Related Work

Social Devices, and the SDP in particular, are based on principles adopted from existing technologies. In fact, one design driver has been to utilize existing work in the design of the SDP. In general, the SDP follows some approaches that are similar to a service-oriented architecture approach to software development [32, 33]. However, the interactions between components in the SDP adhere to the REST principles [16]. In particular, the two central servers, the Configurator and the Orchestrator, have been influenced by and are related to earlier research.

The Configurator is based on the concepts of knowledge-based product configuration that aims at satisfying different customer requirements through mass customization [38]. Instead of explicitly enumerating all products, a product is configured from a standardized set of well-defined components that interact in a predefined way [13, 38]. To achieve this, configuration knowledge, typically represented in a configuration model, needs to capture

the rules on how these sets of parts can be combined. Example application areas of product configuration include the computer hardware industry, the telecommunications industry, the automotive industry [14], and traditional services [41]. The concepts have also been applied to software [19, 28] within the broader research topic of software variability [40]. Within the domain of software architecture, the need for dynamic adaptation has been identified. For example, [25, 18] the use of explicit models of components and connectors as runtime artifacts allows architecture-based adaptation. Furthermore, dynamicity in software variability has received increasing attention, such as in a form of a dynamic software product line [17]. Our work builds directly on the results from these domains and provides a significant difference due to the application in an autonomous platform. The dynamic generation of a configuration model is an open area of research to which we provide an initial solution.

The approaches for coordinating multiple devices have mainly focused on information presentations (e.g., [12, 23, 29]), or for multimedia resource synchronization (e.g., [36, 37, 44]). However, our work is different since we are not aiming to offer only automated services or new kinds of interfaces. For example, in [12], we find similarities in the approach for coordinating the devices, but the aim is different. As [12] and [29] focus on generating user interfaces and coordinating them on the devices, the system philosophy is more user-centric than ours. We, on the contrary, aim to make devices interact and socialize independently, and make the operations visible for the users. When the majority of approaches focus on coordinating the devices in predefined locations, such as smart spaces or homes, our focus is in coordinating the devices wherever they are in proximity to each other in any location. Several approaches have been proposed for the modeling and specification of collective actions (e.g. [4, 24]) and for coordinating computational resources (e.g., [2, 6, 8, 9]). We are revitalizing the idea by applying it to mobile clouds, where actors correspond to individual devices forming the cloud and where a central server is responsible for coordinating the execution of the mobile devices. In previous research, the closest relative to our approach is constituted by coordination languages for mobile agents (e.g., [35]). However, our work is different from these since we are treating complete mobile devices as agents. Consequently, the granularity of the coordination is fundamentally different from agent-based approaches.

In the Social Devices, the operations and the device coordination are based on actions. The notion of an action is rooted in the DisCo method [20], which is a formal specification method for reactive and distributed systems based on the *joint action* theory [4]. Unlike DisCo actions that can be mapped to terms of logic, the actions in the action-oriented programming

model do not have a formal meaning. In DisCo, the participants are typically processes, whereas in the action-oriented programming model the action participants are usually devices. A DisCo action has a *guard* that is similar to the precondition. The action body in DisCo is an atomic parallel assignment clause, whereas the body in the action-oriented programming model is a normal function.

The execution model of an action behaves like a single guarded loop in Dijkstra's guarded command language (GCL) [10]. In the GCL, the basic building blocks are guarded commands is a statement list prefixed with boolean expressions. The statement list is eligible for execution only if the boolean expression is *true*. Similarly, an action can be executed if the precondition is *true* and the statement list corresponds to the body of an action. However, since our contribution is a real programming platform, evaluating the preconditions is alleviated by introducing the notion of triggers, whereas the GCL requires continuous evaluation of guards.

Work has been done on advanced discovery of proximity [22] focusing on energy-efficient discovery. Furthermore, existing web services such as Facebook places, Foursquare, and Google Latitude take location and proximity into account typically by a means of GPS or network identification. However, these services facilitate actions or interactions between devices and users in proximity at most in a manual and rudimentary manner. In contrast, the focus of our work is on actions suitable for proximity-based systems, whereas the SDP relies on relatively simple Bluetooth discovery of proximity.

Social Devices can be used to create actions that facilitate the interaction between humans, devices, and humans and devices. Thus, the actions can be considered a form of groupware [11], which can be broadly defined as software systems that facilitate the activities and the interaction of a group of people for achieving a common task or goal. Actions differ, however, from traditional real-time groupware applications such as chats or document editors in that they do not necessarily have to be tied to a common task or goal but can be more ad hoc in nature, such as greeting a person.

Social Devices also have some similarities to service-oriented computing (SOC) [34]. For instance, actions, action bodies, capabilities, and interface definitions correspond roughly to service compositions, workflow processes, basic services, and service descriptions in SOC. Also, action-oriented programming is concerned with concepts such as the publication of capabilities, action coordination, action composition, and QoS. Social Devices can thus be considered as a model of SOC for a pervasive context where service compositions are initiated proactively.

2.5 Discussion and Results

The first prototype implementation, the SDP, has been implemented, but there are several further research challenges as discussed below.

The decomposition of the SDP architecture (Figure 2.4) aims at a clear separation of responsibilities with the use of RESTful interfaces; this has allowed us to develop each component separately. For example, the CaaS as a service has allowed us to experiment with different intelligent tools without largely affecting other parts of the SDP components. Furthermore, we have focused mostly on the server side, whereas the SDP Client only provides basic functionality. However, the decomposition may not be optimal from, e.g., a performance point of view. Many of the architectural pieces are quite efficient but not yet optimal, especially due to the nature of extension discussed below.

The current approach for configuration returns one arbitrary, valid configuration, and there is practically no means to affect what kind of configuration will be returned. However, the capabilities and the QoS of participating devices affect the overall quality of an action. As an example, audio quality depends on the quality of the device speakers. Furthermore, contextual data may also affect the QoS of a service composition: For instance, the user-perceived audio quality depends on the distance from a display or loudspeaker. Finally, device users have different kinds of preferences. Thus, given a set of devices with varying capabilities and levels of QoS, the problem becomes finding the best or good enough devices for the action roles. The configuration of an action is typically an under-constrained problem to which there are several valid configurations. One advanced approach is to use recommender systems as a means for finding devices so that user preferences and device capabilities are taken into account. Recommender systems are information systems that have been introduced in various domains for proposing items to users based on user preferences using techniques from the field of artificial intelligence [15]. Our initial feasibility study indicates that technically recommender systems could be applied instead of configurators by representing the configuration problem as a recommendation problem [31]. Furthermore, despite the computational complexity of the recommendation problem, the response times seem to be quite feasible. However, the recommendation adds challenges, such as whose preferences should be taken into account in social situations and how user preferences should be captured. To better understand the benefits and challenges of using the recommendation, we will augment CaaS with recommendation concepts in the near future.

The implementation of the SDP currently focuses more on devices than users. However, in the concept of Social Devices, both social interations and devices, are tightly coupled with the users. Some devices can be personal

while other devices can be shared. To enhance the personalization and contextualization of social devices, the concept of users, including social relations to other users and devices, is required. For example, the interaction with a friend's device is most likely different compared to one with a colleague at work.

To enhance personalization, we need to introduce the notion of a user model [5]. A user model provides an internal representation of a user and allows the adaption of actions to users' behavior, characteristics, and needs. A user model can include personal information, interests, skills, knowledge, goals, and preferences. The user model can be updated explicitly or implicitly. For instance, the user model could be updated explicitly by asking the user to rate an action after the action has finished. Alternatively, the SDP could observe the user's behavior and implicitly update the user model. For example, the interruption of an action could indicate that the user does not like the specific action and the SDP learns not to execute that action for that user anymore. In addition to deciding what actions to execute, a user model would enable adaptive deployment of an action on specific devices based on user preferences as mentioned earlier about the recommendation. Also, the action behavior and content could be adapted based on the user model. For instance, an action could assign team members in a game based on the user models of the players. We will explore the use of user models and personalization in the SDP in the near future.

Privacy and security are issues that require further investigation. Currently, users can enable and disable capabilities from their devices and adjust settings to their liking. However, as the platform becomes more complex, strict privacy policies will be needed.

Currently, the SDP uses low-level contextual data [7], such as proximity and device state obtained through device sensors, to trigger and determine which actions can be executed. This kind of low-level contextual data may not in itself be sufficient to determine more high-level contexts such as specific social interactions. Instead, to be able to sense more complex social contexts, the SDP needs to have a way to infer high-level contexts by, e.g., combining data from several low-level sensors or by using application data such as a calendar. The identification of specific social situations can also help to increase the privacy of users by not allowing certain actions to be executed in certain situations.

Although Social Devices are not critical in the sense that an action needs to recover or terminate properly, the highly dynamic nature can result in actions that do not execute properly, which can become frustrating and inconvenient. Therefore, the reliability of the SDP would benefit from various additional supporting services. In particular, the SDP Client provides sev-

eral opportunities for advanced context-aware behavior and self-management capabilities.

The action and interface definitions follow certain predefined conventions and are in Python due to its simplicity and, in particular, the use of existing parsers. Currently, however, there are no means to support and ensure that the correct conventions are used when defining the actions and interfaces in Python. An integrated development environment (IDE) would offer an approach to easily construct actions. One approach is to use a plug-in for the Eclipse open source IDE (www.eclipse.org) that then connects to the RESTful API of the SDP to upload actions. Furthermore, it seems to be possible to ease action definition by a domain-specific language, with even a graphical notation, that is then transformed into Python. Such an approach could be used for enabling end-user programming of actions. Additionally, such conventions would facilitate application deployment, using different programming languages, to platforms that do not readily support Python. Nevertheless, Python seems to be quite a feasible coordination language for the actual action coordination requirements.

The concept of Social Devices intervenes with established norms in social interactions between people. The ongoing CoSMo[1] research project studies multimodal interaction techniques and conducts studies in real context to gain an empirical understanding of how users experience action-oriented applications. Moreover, the project studies how users feel about actions that start proactively and which kinds of actions are socially acceptable [42]. The concept itself has received quite a polarized response as some have questioned the concept completely while others have been quite enthusiastic. For example, it was considered tempting that Social Devices would violate certain social norms. Only a few have stayed neutral. Nevertheless, the contribution of this chapter has been to introduce the concept of Social Devices while more studies in the real social context are needed.

From the developer's point of view, we have evaluated the action-oriented programming model by creating a number of demo applications for the SDP. These applications include Photo Sharing, Car Game (each device is a "camera" to the track, and everybody controls theor own car with voice), Greeting Devices (our phones greet each others), and Unread SMS Reminder (the reminder is given by a friend's device).

With this action-oriented paradigm, development has been fast and intuitive. An undergraduate student was able to create the photo sharing action in a couple of days. Constructing a similar application from scratch would

[1] CoSMo — Co-Located User Interaction through Social Mobile Devices, funded by the Academy of Finland in 2013–2015, `http://www.cs.tut.fi/ihte/projects/CoSMo`

have required concentrating on difficult connectivity and synchronization issues. Now, these are hidden by the concepts of the programming model. Moreover, we tested the system from developer's point of view by hiring an outside team to design and implement a multiplayer game by using the platform. The application was developed in cooperation with Demola[2], an innovation instrument targeted at fostering innovation and experimenting with radical ideas. Overall, the results (reported in [26]) were encouraging as the team was enthusiastic about the concept and after only a short introduction managed to start implementing their application.

Furthermore, the applicability of the action-oriented programming model should be studied beyond the SDP. The programming model seems to be applicable to scenarios that involve coordinated and synchronized behavior between several distributed entities; the behavior is then encapsulated into actions that are initiated proactively. Example application areas include smart homes, which often involve distributed, coordinated behavior between home appliances and devices as a reaction to a certain condition or user needs.

A programming model reflects the system that executes the programs. Successful programming models have a straightforward relation to the system. For example, procedural and object-oriented languages usually have an obvious mapping to the von Neumann architecture on which they are executed. Similarly, the action-oriented programming model as described in this chapter reflects the distributed system executing the actions and vice versa. This means that future needs arising from new application areas on which the programming model is applied may require changes or additions to the programming model.

A future work item for the model includes adding *transactionality*. An action is a natural unit for transactional execution. In the beginning of the action, devices join the transaction, and in the end, if all went well, the changes become persistent.

In the programming model and the SDP, action coordination relies on the server side. Some confluence to the Message Queue or Publish-subscribe technologies can be seen: Orchestrator could be regarded as the publisher that sends messages to an action-specific message bus. However, unlike in the aforementioned technologies, Orchestrator is always aware of the message receiver and expects a response since the operations are invoked synchronously. Moreover, a device is reserved for one action at a time, and hence is subscribed to only one action-specific message bus at a time. However, research on different kinds of coordination approaches is ongoing. For

[2] `http://www.demola.fi`

instance, performance overhead could be reduced by one device coordinating the other devices directly with Bluetooth, or even by distributing the coordination among many devices participating in the action.

2.6 Conclusions

The way people communicate and socialize has changed due to social media services. Whereas these services are useful in many ways for supporting remote social interactions, they offer only limited advantages for face-to-face and co-located situations. In this chapter, we described the concept of Social Devices that offers a new kind of socio-digital system for co-located devices and humans to interact with each other. The interactions are based on actions that are predefined processes where devices are used in certain roles and where each device offers certain services according to its capabilities. In contrast to smart spaces, the actions are not tied to any specific location or device, and the devices interact in a more ad-hoc manner. On the other hand, the processes are made visible for the users, and moreover, users may interact in these actions as well.

The SDP was introduced as a prototype implementation for the concept of Social Devices. The SDP offers components for the main practical problems that have emerged with Social Devices. These problems are: tracking the proximity of devices, finding a suitable configuration for a set of devices within close proximity, and orchestrating the operation executions on the devices. While the SDP offers solutions for the main problems, more research is needed in many areas. For instance, personalizing the contents of the actions for users in different contexts requires further studies. Also, understanding how people thoroughly understand the idea of Social Devices and the actions requires studies in a real context. However, as we designed the client to be lightweight, implementing it in other mobile platforms shuold happen in the near future, allowing us to focus on studies on a larger scale.

References

[1] T. Aaltonen et al. “An Action-Oriented Programming Model for Pervasive Computing in a Device Cloud”. In: *20th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE Computer Society, 2013 (to be published).

[2] T. Aaltonen et al. “Coordinating Aspects and Objects”. In: *Electr. Notes Theor. Comput. Sci.* 68.3 (2003), pp. 248–267.

[3] L. Atzori, A. Iera, and G. Morabito. "The Internet of Things: A survey". In: *Comput. Netw.* 54.15 (2010), pp. 2787–2805.

[4] R.-J. Back and R. Kurki-Suonio. "Distributed Cooperation with Action Systems". In: *ACM Trans. Program. Lang. Syst.* 10.4 (1988), pp. 513–554.

[5] P. Brusilovsky and M. T. Maybury. "From adaptive hypermedia to the adaptive web". In: *Commun. ACM* 45.5 (2002), pp. 30–33.

[6] B. Chapman et al. "Opus: A Coordination Language for Multidisciplinary Applications". In: *Sci. Program.* 6.4 (1997), pp. 345–362.

[7] G. Chen and D. Kotz. *A Survey of Context-Aware Mobile Computing Research*. Tech. rep. TR2000-381. Dartmouth College, Computer Science, 2000.

[8] P. Ciancarini. "Coordination Models and Languages as Software Integrators". In: *ACM Comput. Surv.* 28.2 (1996), pp. 300–302.

[9] J. Darlington et al. "Functional Skeletons for Parallel Coordination". In: *Proceedings of the First International Euro-Par Conference on Parallel Processing*. Springer-Verlag, 1995.

[10] E. W. Dijkstra. "Guarded commands, nondeterminacy and formal derivation of programs". In: *Commun. ACM* 18.8 (1975), pp. 453–457.

[11] C. A. Ellis, S. J. Gibbs, and G. Rein. "Groupware: some issues and experiences". In: *Commun. ACM* 34.1 (1991), pp. 39–58.

[12] C. Elting. "Orchestrating output devices: planning multimedia presentations for home entertainment with ambient intelligence". In: *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*. ACM, 2005.

[13] B. Faltings and E. C. Freuder. "Guest Editors' Introduction: Configuration". In: *IEEE Intelligent Systems* 13.4 (1998), pp. 32–33.

[14] A. Felfernig, G. Friedrich, and D. Jannach. "Conceptual Modeling for Configuration of Mass-Customizable Products". In: *Artificial Intelligence in Engineering* 15.2 (2001), pp. 165–176.

[15] A. Felfernig, G. Friedrich, and L. Schmidt-Thieme. "Guest Editors' Introduction: Recommender Systems". In: *IEEE Intelligent Systems* 22.3 (2007), pp. 18–21.

[16] R. T. Fielding. “Architectural styles and the design of network-based software architectures”. AAI9980887. PhD thesis. University of California, Irvine, 2000. ISBN: 0-599-87118-0.

[17] S. Hallsteinsen et al. “Dynamic Software Product Lines”. In: *Computer* 41.4 (2008), pp. 93 –95.

[18] A. van der Hoek. “Design-time product line architectures for any-time variability”. In: *Sci. Comput. Program.* 53.3 (2004), pp. 285–304.

[19] L. Hotz, K. Wolter, and T. Krebs. *Configuration in Industrial Product Families: The ConIPF Methodology*. IOS Press, Inc., 2006. ISBN: 1586036416.

[20] H.-M. Järvinen et al. “Object-oriented specification of reactive systems”. In: *International Conference on Software Engineering*. IEEE Computer Society Press, 1990.

[21] A. Klein et al. “Access Schemes for Mobile Cloud Computing”. In: *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*. IEEE Computer Society, 2010.

[22] R. H. Kravets. “Enabling Social Interactions off the Grid”. In: *IEEE Pervasive Computing* 11.2 (2012), pp. 8–11.

[23] C. Kray, A. Krüger, and C. Endres. “Some Issues on Presentations in Intelligent Environments”. In: *In First European Symposium on Ambient Intelligence (EUSAI)*. Springer, 2003.

[24] R. Kurki-Suonio and T. Mikkonen. “Abstractions of Distributed Cooperation, their Refinement and Implementation”. In: *Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems*. IEEE Computer Society, 1998.

[25] J. Magee and J. Kramer. “Dynamic structure in software architectures”. In: *SIGSOFT Softw. Eng. Notes* 21.6 (1996), pp. 3–14.

[26] N. Mäkitalo, T. Aaltonen, and T. Mikkonen. “First Hand Developer Experiences of Social Devices”. In: *Workshop on Mobile Cloud and Social Perspectives, MoCSoP’13, (to appear)*. 2013.

[27] N. Mäkitalo et al. “Social devices: collaborative co-located interactions in a mobile cloud”. In: *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*. ACM, 2012.

[28] T. Männistö, T. Soininen, and R. Sulonen. “Product Configuration View to Software Product Families”. In: *Proc. of the SCM-10 workshop at ICSE ’01*. 2001, pp. 14–15.

[29] B. A. Myers et al. “Taking Handheld Devices to the Next Level”. In: *Computer* 37.12 (2004), pp. 36–43.

[30] V. Myllärniemi et al. “Configurator-as-a-service: tool support for deriving software architectures at runtime”. In: *Proceedings of the WICSA/ECSA 2012 Companion Volume*. ACM, 2012.

[31] J. Pääkkö et al. “Applying Recommendation Systems for Composing Dynamic Services for Mobile Devices”. In: *19th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE Computer Society, 2012.

[32] M. P. Papazoglou et al. “Service-Oriented Computing: State of the Art and Research Challenges”. In: *Computer* 40.11 (2007), pp. 38–45.

[33] M. Papazoglou. “Extending the Service-oriented architecture”. In: *Business Integration Journal* 7.1 (2005).

[34] M. Papazoglou and D. Georgakopoulos. “Service-oriented computing”. In: *Commun. ACM* 46.10 (2003), pp. 25–28.

[35] F. Peschanski et al. “Coordinating mobile agents in interaction spaces”. In: *Science of Computer Programming* 66.3 (2007), pp. 246–265.

[36] M. Raatikainen et al. “Mobile Content as a Service: A Blueprint for a Vendor-Neutral Cloud of Mobile Devices”. In: *IEEE Software* 29.4 (2012), pp. 28–32.

[37] J. Rekimoto. “Multiple-computer user interfaces: ”beyond the desktop” direct manipulation environments”. In: *CHI ’00 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2000.

[38] D. Sabin and R. Weigel. “Product Configuration Frameworks—A Survey”. In: *IEEE Intelligent Systems* 13.4 (1998), pp. 42–49.

[39] M. Satyanarayanan. “Pervasive computing: Vision and challenges”. In: *IEEE Personal communications* 8.4 (2001), pp. 10–17.

[40] M. Svahnberg, J. van Gurp, and J. Bosch. “A taxonomy of variability realization techniques”. In: *Softw. Pract. Exper.* 35.8 (2005), pp. 705–754.

[41] J. Tiihonen et al. “Applying the Configuration Paradigm to Mass-customize Contract Based Services”. In: *The World Conference on Mass Customization & Personalization*. 2007.

[42] K. Väänänen-Vainio-Mattila et al. “Social Devices as a New Type of Social System: Enjoyable or Embarrassing Experiences?” In: *Workshop on Experiencing Interactivity in Public Spaces in conjunction with CHI ’13 (to appear)*. 2013.

[43] M. Weiser. “The computer for the 21st century”. In: *Scientific American* 265.3 (1991), pp. 94–104.

[44] B. Xing, K. Seada, and N. Venkatasubramanian. “Proximiter: Enabling mobile proximity-based content sharing on portable devices”. In: *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*. IEEE Computer Society, 2009.

3 Prediction-Based Virtual Machine Provisioning and Admission Control for Multi-tier Web Applications

Adnan Ashraf, Benjamin Byholm, and Ivan Porres
Department of Information Technologies
Åbo Akademi University, Turku, Finland
Email: {aashraf, bbyholm, iporres}@abo.fi

Abstract–This chapter presents a prediction-based, cost-efficient Virtual Machine (VM) provisioning and admission control approach for multi-tier web applications. The proposed approach provides automatic deployment and scaling of multiple simultaneous web applications on a given Infrastructure as a Service (IaaS) cloud in a shared hosting environment. It monitors and uses resource utilization metrics and does not require a performance model of the applications or the infrastructure dynamics. The shared hosting environment allows us to share VM resources among deployed applications, reducing the total number of required VMs. The proposed approach comprises three sub-approaches: a reactive VM provisioning approach called ARVUE, a hybrid reactive-proactive VM provisioning approach called Cost-efficient Resource Allocation for Multiple web applications with Proactive scaling (CRAMP), and a session-based adaptive admission control approach called adaptive Admission Control for Virtualized Application Servers (ACVAS). Performance under varying load conditions is guaranteed by automatic adjustment and tuning of the CRAMP and ACVAS parameters. The proposed approach is demonstrated in discrete-event simulations and is evaluated in a series of experiments involving synthetic as well as realistic load patterns.

Keywords-Cloud computing, virtual machine provisioning, admission control, web application, cost-efficiency, performance.

3.1 Introduction

The resource needs of web applications vary over time, depending on the number of concurrent users and the type of work performed. This stands in contrast to static content, which requires no further processing by the server than sending predefined data to an output stream. As the demand for an application grows, so does its demand for resources, until the demand for a key resource outgrows the supply and the performance of the application deteriorates. Users of an application starved for resources tend to notice this as increased latency and lower throughput for requests, or they might receive no service at all if the problem progresses further.

To handle multiple simultaneous users, web applications are traditionally deployed in a three-tiered architecture, where a computer cluster of fixed size represents the application server tier. This cluster provides dedicated application hosting to a fixed amount of users. There are two problems with this approach: firstly, if the amount of users grows beyond the predetermined limit, the application will become starved for resources. Secondly, while the amount of users is lower than this limit, the unused resources constitute waste.

A recent study showed that the underutilization of servers in enterprises is a matter of concern [37]. This inefficiency is mostly due to application isolation: a consequence of dedicated hosting. Sharing of resources between applications leads to higher total resource utilization and thereby to less waste. Thus, the level of utilization can be improved by implementing what is known as shared hosting [36]. Shared hosting is already commonly used by web hosts to serve static content belonging to different customers from the same set of servers, as no sessions need to be maintained.

Cloud computing already allows us to alleviate the utilization problem by dynamically adding or removing available Virtual Machine (VM) instances at the infrastructure level. However, the problem remains to some extent, as Infrastructure as a Service (IaaS) providers operate at the level of VMs, which does not provide high granularity. This can be solved by operating at the Platform as a Service (PaaS) level instead. However, one problem still remains: resources cannot be immediately allocated or deallocated. In many cases, there exists a significant provisioning delay on the order of minutes.

Shared hosting of dynamic content also presents new challenges: capacity planning is complicated, as different types of requests might require varying amounts of a given resource. For example, consider a web shop: adding items to the shopping basket might require less resources than computing the final price with taxes and rebates included. During a shopping session, a user might add several items to their shopping basket, while the final price is only

computed at checkout. The session also has to be reliably maintained, so that the contents of the shopping basket do not suddenly disappear. Otherwise, the shop might lose customers.

Application-specific knowledge is necessary for a PaaS provider to efficiently host complex applications with highly varying resource needs. When hosting third-party dynamic content in a shared environment that application-specific knowledge might be unavailable. It is also unfeasible for a PaaS provider to learn enough about all of the applications belonging to their customers.

Traditional performance models based on queuing theory try to capture the behavior of purely open or closed systems [25]. However, Rich Internet Applications (RIAs) have workloads with sessions, exhibiting a partially-open behavior, which includes components from both the open and the closed model. Given a better performance model of an application, it might be possible to plan the necessary capacity, but the problem of obtaining said model remains.

If the hosted applications are seldom modified it might be feasible to automatically derive the necessary performance models by benchmarking each application in isolation [36]. This might apply to hosting first- or second-party applications. However, when hosting third-party applications under continuous development, they may well change frequently enough for this to be unfeasible.

Another problem is determining the amount of VMs to have at a given moment. As one cannot provision fractions of a VM, the actual capacity demand will need to be quantized in one way or another. Figure 3.1 shows a demand and a possible quantization thereof. Overallocation implies an opportunity cost — underallocation implies lost revenue.

Finally, there is also the issue of admission control. This is the problem of determining how many users to admit to a server at a given moment in time, so that said server does not become overloaded. Preventive measures are a good way of keeping server overload from occurring at all. This is traditionally achieved by only relying on two possible decisions: rejection or acceptance.

Once more, the elastic nature of the cloud means that we have more resources available at our discretion and can scale up to accommodate the increase in traffic. However, resource allocation still takes a considerable amount of time, due to the provisioning delay, and admitting too much traffic is an unattractive option, even if new resources will arrive in a while.

This chapter presents a prediction-based, cost-efficient VM provisioning and admission control approach for multi-tier web applications. The proposed approach provides automatic deployment and scaling of multiple si-

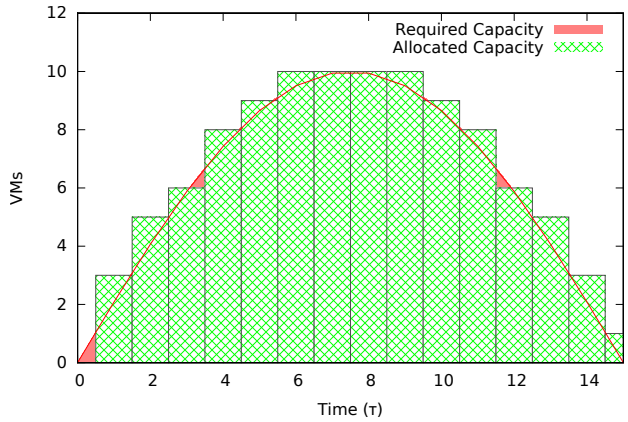


Figure 3.1: The actual capacity demand has to be quantized at a resolution determined by the capacity of the smallest VM available for provisioning. Overallocation means an opportunity cost, underallocation means lost revenue.

multaneous third-party web applications on a given IaaS cloud in a shared hosting environment. It monitors and uses resource utilization metrics and does not require a performance model of the applications or the infrastructure dynamics. The research applies to PaaS providers and large Software as a Service (SaaS) providers with multiple applications. We deal with stateful RIAs over the Hypertext Transfer Protocol (HTTP).

The proposed approach comprises three sub-approaches. It provides a reactive VM provisioning approach called ARVUE [9], a hybrid reactive-proactive VM provisioning approach called Cost-efficient Resource Allocation for Multiple web applications with Proactive scaling (CRAMP) [8], and a session-based adaptive admission control approach called adaptive Admission Control for Virtualized Application Servers (ACVAS) [7]. Both ARVUE and CRAMP provide autonomous shared hosting of third-party Java Servlet applications on an IaaS cloud. However, CRAMP provides better responsiveness and results than the purely reactive scaling of ARVUE. We concluded that admission control might be able to reduce the risk of servers becoming overloaded. Therefore, the proposed approach augments VM provisioning with a session-based adaptive admission control approach called ACVAS. ACVAS implements per-session admission, which reduces the risk of over-admission. Furthermore, instead of relying only on rejection of new sessions, it implements a simple session deferment mechanism that reduces

the number of rejected sessions while increasing session throughput. Thus, the admission controller can decide to admit, defer, or reject an incoming new session. Performance under varying load conditions is guaranteed by automatic adjustment and tuning of the CRAMP and ACVAS parameters. The proposed approach is demonstrated in discrete-event simulations and is evaluated in a series of experiments involving synthetic as well as realistic load patterns. Byholm [11] described the prototype implementation of these concepts.

We proceed as follows. Section 3.2 discusses important related works. Section 3.3 presents the system architecture. The proposed VM provisioning and admission control algorithms are described in Section 3.4. In Section 3.5, we present simulation results before concluding in Section 3.6.

3.2 Related Work

Due to the problems mentioned in Section 3.1, existing works on PaaS solutions tend to use dedicated hosting on a VM-level for RIAs. This gives the level of isolation needed to reliably host different applications without them interfering with each other, as resource management will be handled by the underlying operating system. However, this comes at the cost of prohibiting resource sharing among instances. In order to reliably do shared hosting of third-party applications, there is a need for a way to prevent applications from interfering with each other, without preventing the sharing of resources. Google App Engine is different here in that it instead offers a sandboxed runtime for the applications to run in [17]. Another way is to use shared hosting to run multiple applications in the same Java Virtual Machine (JVM) [1].

There are many metrics available for measuring Quality of Service (QoS). A common metric is Round Trip Time (RTT), which is a measure of the time required for sending a request and receiving a response. This approach has a drawback in that different programs might have various expected processing times for requests of different types. This means that application-specific knowledge is required when using RTT as a QoS metric. This information might not be easy to obtain if an application is under constant development. Furthermore, when a server nears saturation, its response time grows exponentially. This makes it difficult to obtain good measurements in a high-load situation. For this reason, we use server Central Processing Unit (CPU) load average and memory utilization as the primary QoS metrics. An overloaded server will fail to meet RTT requirements.

Reactive scaling works by monitoring user load in the system and reacting to observed variations therein by making decisions for allocation or

deallocation. In our previous work [11, 1, 9], we built a prototype of an autonomous PaaS called ARVUE. It implements reactive scaling. However, in many cases, the reactive approach suffers in practice, due to delays of several minutes inherent in the provisioning of VMs [31]. This shortcoming is avoidable with proactive scaling.

Proactive scaling attempts to overcome the limitations of reactive scaling by forecasting future load trends and acting upon them, instead of directly acting on observed load. Forecasting usually has the drawback of added uncertainty, as it introduces errors into the system. The error can be mitigated by a hybrid approach, where forecast values are supplemented with error estimates, which affect a blend weight for observed and forecast values. We have developed a hybrid reactive-proactive VM provisioning algorithm called CRAMP [8].

Admission control is a strategy for keeping servers from becoming overloaded. This is achieved by limiting the amount of traffic each server receives by means of an intermediate entity known as an *admission controller*. The admission controller may deny entry to fully utilized servers, thereby avoiding server overload. If a server were to become overloaded, all users of that server, whether existing or arriving, would suffer from deteriorated performance and possible Service-Level Agreement (SLA) violations.

Traditional admission control strategies have mostly been request-based, where admission control decisions would be made for each individual request. This approach is not appropriate for stateful web applications from a user experience point of view. If a request were to be denied in the middle of an active session, when everything was working well previously, the user would have a bad experience. Session-Based Admission Control (SBAC) is an alternative strategy, where the admission decision is made once for each new session and then enforced for all requests inside of a session [26]. This approach is better from the perspective of the user, as it should not lead to service being denied in the middle of a session. This approach has usually been implemented using interval-based on-off control, where the admission controller either admits or rejects all sessions arriving within a predefined time interval. This approach has a flaw in that servers may become overloaded if they accept too many requests in an admission interval, as the decisions are made only at interval boundaries. Per-session admission control avoids this problem by making a decision for each new session, regardless of when it arrives. We have developed ACVAS [7], a session-based admission control approach with per-session admission control. ACVAS uses SBAC with a novel deferment mechanism for sessions, which would have been rejected with the traditional binary choice of acceptance or rejection.

3.2.1 VM Provisioning Approaches

Most of the existing works on VM provisioning for web-based systems can be classified into two main categories: plan-based approaches and control theoretic approaches [15, 29, 30]. Plan-based approaches can be further classified into workload prediction approaches [31, 6] and performance dynamics model approaches [39, 21, 14, 23, 19]. One common difference between all existing works discussed here and the proposed approach is that the proposed approach uses shared hosting. Another distinguishing characteristic of the proposed approach is that in addition to VM provisioning for the application server tier, it also provides dynamic scaling of multiple web applications. In ARVUE [9], we used shared hosting with reactive resource allocation. In contrast, our proactive VM provisioning approach CRAMP [8] provides improved QoS with prediction-based VM provisioning.

Ardagna et al. [6] proposed a distributed algorithm for managing SaaS cloud systems that addresses capacity allocation for multiple heterogeneous applications. Raivio et al. [31] used proactive resource allocation for short message services in hybrid clouds. The main drawback of their approach is that it assumes server processing capacity in terms of messages per second, which is not a realistic assumption for HTTP traffic where different types of requests may require different amounts of processing time. Nevertheless, the main challenge in the prediction-based approaches is in making good prediction models that could ensure high prediction accuracy with low computational cost. In our proposed approach, CRAMP is a hybrid reactive-proactive approach. It uses a two-step prediction method with Exponential Moving Average (EMA), which provides high prediction accuracy under real-time constraints. Moreover, it gives more or less weight to the predicted utilizations based on the Normalized Root Mean Square Error (NRMSE).

TwoSpot [39] supports hosting of multiple web applications, which are automatically scaled up and down in a dedicated hosting environment. The scaling down is decentralized, which may lead to severe random drops in performance. Hu et al. [21] presented an algorithm for determining the minimum number of required servers, based on the expected arrival rate, service rate, and SLA. In contrast, the proposed approach does not require knowledge about the infrastructure or performance dynamics. Chieu et al. [14] presented an approach that scales servers for a particular web application based on the number of active user sessions. However, the main challenge is in determining suitable threshold values on the number of user sessions. Iqbal et al. [23] proposed an approach for multi-tier web applications, which uses response time and CPU utilization metrics to determine the bottleneck tier and then scales it by provisioning a new VM. Han et al. [19] proposed

a reactive resource allocation approach to integrate VM-level scaling with a more fine-grained resource-level scaling. In contrast, CRAMP supports hybrid reactive-proactive resource allocation with proportional and derivative factors to determine the number of VMs to provision.

Dutreilh et al. [15] and Pan et al. [29] used control theoretic models to design resource allocation solutions for cloud computing. Dutreilh et al. presented a comparison of static threshold-based and reinforcement learning techniques. Pan et al. used Proportional-Integral (PI)-controllers to provide QoS guarantees. Patikirikorala et al. [30] proposed a multi-model framework for implementing self-managing control systems for QoS management. The work is based on a control theoretic approach called the Multi-Model Switching and Tuning (MMST) adaptive control. In comparison to the control theoretic approaches, our proposed approach also uses proportional and derivative factors, but it does not require knowledge about the performance models or infrastructure dynamics.

3.2.2 Admission Control Approaches

The existing works on admission control for web-based systems can be classified according to the scheme presented in Almeida et al. [3]. For instance, Robertsson et al. [32] and Voigt and Gunningberg [38] are control theoretic approaches, while Huang et al. [22] and Muppala and Zhou [26] use machine learning techniques. Similarly, Cherkasova and Phaal [13], Almeida et al. [3], Chen et al. [12], and Shaaban and Hillston [33] are utility-based approaches.

Almeida et al. [3] proposed a joint resource allocation and admission control approach for a virtualized platform hosting a number of web applications, where each VM runs a dedicated web service application. The admission control mechanism uses request-based admission control. The optimization objective is to maximize the provider's revenue, while satisfying the customers' QoS requirements and minimizing the cost of resource utilization. The approach dynamically adjusts the fraction of capacity assigned to each VM and limits the incoming workload by serving only the subset of requests that maximize profits. It combines a performance model and an optimization model. The performance model determines future SLA violations for each web service class based on a prediction of future workloads. The optimization model uses these estimates to make the resource allocation and admission control decisions.

Cherkasova and Phaal [13] proposed an SBAC approach that uses the traditional *on-off* control. It supports four admission control strategies: responsive, stable, hybrid, and predictive. The hybrid strategy tunes itself to be more stable or more responsive based on the observed QoS. The proposed

approach measures server utilizations during predefined time intervals. Using these measured utilizations, it computes predicted utilizations for the next interval. If the predicted utilizations exceed specified thresholds, the admission controller rejects all new sessions in the next time interval and only serves the requests from already admitted sessions. Once the predicted utilizations drop below the given thresholds, the server changes its policy for the next time interval and begins to admit new sessions again.

Chen et al. [12] proposed Admission Control based on Estimation of Service times (ACES). That is, to differentiate and admit requests based on the amount of processing time required by a request. In ACES, admission of a request is decided by comparing the available computation capacity to the predetermined delay bound of the request. The service time estimation is based on an empirical expression, which is derived from an experimental study on a real web server. Shaaban and Hillston [33] proposed Cost-Based Admission Control (CBAC), which uses a congestion control technique. Rather than rejecting user requests at high load, CBAC uses a discount-charge model to encourage users to postpone their requests to less loaded time periods. However, if a user chooses to go ahead with the request in a high load period, then an extra charge is imposed on the user request. The model is effective for e-commerce web sites when more users place orders that involve monetary transactions. A disadvantage of CBAC is that it requires CBAC-specific web pages to be included in the web application.

Muppala and Zhou [26] proposed the Coordinated Session-based Admission Control (CoSAC) approach, which provides SBAC for multi-tier web applications with per-session admission control. CoSAC also provides coordination among the states of tiers with a machine learning technique using a Bayesian network. The admission control mechanism differentiates and admits user sessions based on their type. For example, browsing mix session, ordering mix session, and shopping mix session. However, it remains unclear how it determines the type of a particular session in the first place. Huang et al. [22] proposed admission control schemes for proportional differentiated services. It applies to services with different priority classes. The paper proposes two admission control schemes to enable Proportional Delay Differentiated Service (PDDS) at the application level. Each scheme is augmented with a prediction mechanism, which predicts the total maximum arrival rate and the maximum waiting time for each priority class based on the arrival rate in the current and last three measurement intervals. When a user request belonging to a specific priority class arrives, the admission control algorithm uses the time series predictor to forecast the average arrival rate of the class for the next interval, computes the average waiting time for the class for the next interval, and determines if the incoming user request

is admitted to the server. If admitted, the client is placed at the end of the class queue.

Voigt and Gunningberg [38] proposed admission control based on the expected resource consumption of the requests, including a mechanism for service differentiation that guarantees low response time and high throughput for premium clients. The approach avoids overutilization of individual server resources, which are protected by dynamically setting the acceptance rate of resource-intensive requests. The adaptation of the acceptance rates (average number of requests per second) is done by using Proportional-Derivative (PD) feedback control loops. Robertsson et al. [32] proposed an admission control mechanism for a web server system with control theoretic methods. It uses a control theoretic model of a $G/G/1$ system with an admission control mechanism for nonlinear analysis and design of controller parameters for a discrete-time PI-controller. The controller calculates the desired admittance rate based on the reference value of average server utilization and the estimated or measured load situation (in terms of average server utilization). It then rejects those requests that could not be admitted.

3.3 Architecture

The system architecture of the proposed VM provisioning and admission control approach is depicted in Figure 3.2. It consists of the following components: a *load balancer* with an accompanying *configuration file*, the *global controller*, the *admission controller*, the *cloud provisioner*, the *application servers* containing *local controllers*, the *load predictors*, an *entertainment server*, and an *application repository*.

The purpose of the *load balancer* is to distribute the workload evenly throughout the system, while the *admission controller* is responsible for admitting users, when deemed possible. The *cloud provisioner* is an external component, which represents the control service of the underlying IaaS provider. *Application servers* are dynamically provisioned VMs belonging to the underlying IaaS cloud, capable of running multiple concurrent applications contained in an *application repository*.

3.3.1 Load Balancer

The purpose of the load balancer is to distribute the workload among the available application servers. The prototype implementations of ARVUE [1, 9, 11] and CRAMP [8] use the free, lightweight load balancer HAProxy [34], which can act as a reverse proxy in either of two modes: Transmission Control

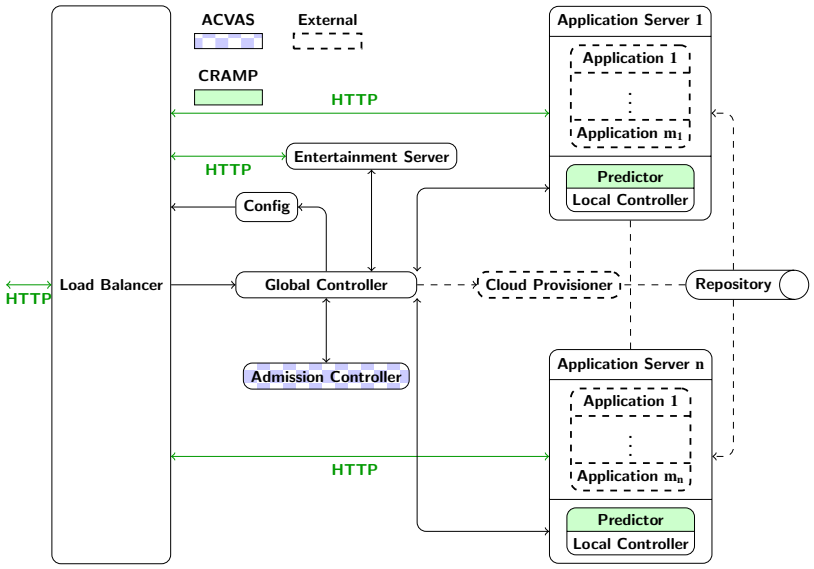


Figure 3.2: System architecture of the proposed VM provisioning and admission control approach.

Protocol (TCP) or HTTP, which correspond to layers 4 and 7 in the Open Systems Interconnection (OSI) model. We use the HTTP mode, as ARVUE and CRAMP are designed for stateful web applications over HTTP.

HAProxy includes powerful logging capabilities using the Syslog standard. It also supports session affinity, the ability to direct requests belonging to a single session to the same server, and Access Control Lists (ACLs), even in combination with Secure Socket Layer (SSL) since version 1.5.

Session affinity is supported by cookie rewriting or insertion. As the prototype implementations of ARVUE and CRAMP are designed for Vaadin applications [18], which use the Java Servlet technology, applications already use the `JSESSIONID` cookie, which uniquely identifies the session the request belongs to. Thus, HAProxy only has to intercept the `JSESSIONID` cookie sent from the application to the client and prefix it with the identifier of the backend in question. Incoming `JSESSIONID` cookies are similarly intercepted and the inserted prefix is removed before they are sent to the applications.

HAProxy also comes with a built-in server health monitoring system, based on making requests to servers and measuring their response times. However, this system is currently not in use, as the proposed approach does its own health monitoring by observing different metrics.

When an application request arrives at the load balancer, it gets redirected to a suitable server according to the current configuration. A request for an application not deployed at the moment is briefly sent to a server tasked with entertaining the user and showing that the request is being processed until the application has been successfully deployed, after which it is delivered to the correct server. This initial deployment of an application will take a much longer time than subsequent requests, currently on the order of several seconds.

The load balancer is dynamically reconfigured by the global controller as the properties of the cluster change. When an application is deployed, the load balancer is reconfigured with a mapping between a Uniform Resource Identifier (URI) that uniquely identifies the application and a set of application servers hosting the application, by means of an ACL, a usage declaration and a backend list. Weights for servers are periodically recomputed according to the health of each server, with higher weights assigned to less loaded servers.

The weights are integers in the range $[0, W_{\mathrm{MAX}}]$, where higher values mean higher priority. In the case of HAProxy, $W_{\mathrm{MAX}} = 255$. The value 0 is special in that it effectively prevents the server from receiving any new requests. This is explained by the weighting algorithm in Algorithm 3.1, which distributes the load among the servers so that each server receives a number of requests proportional to its weight divided by the sum of all the weights. This is a simple mapping of the current load to the weight interval. Here, $S(k)$ is the set of servers at discrete time k, $C_w(s,k)$ is the weighted load average of server s at time k, $C(s,k)$ is the measured load average of server s at time k, and similarly $\hat{C}(s,k)$ is the predicted load average of server s at time k. $w_c \in [0,1]$ is the weighting coefficient for CPU load average, C_{US} is the server load average upper threshold, and $W(s,k)$ is the weight of server s at time k for load balancing. Thus, the algorithm obtains $C(s,k)$ and $\hat{C}(s,k)$ of each server $s \in S(k)$ and uses them along with w_c to compute $C_w(s,k)$ of each server (line 1). Afterwards, it uses $C_w(s,k)$ to compute $W(s,k)$ of each server s (lines 2–10). The notation used in the algorithm is also defined in Table 3.1 in Section 3.4.

3.3.2 Global Controller

The global controller is responsible for managing the cluster by monitoring its constituents and reacting to changes in the observed parameters, as reported by the local controllers. It can be viewed as a control loop that implements the VM provisioning algorithms described in Section 3.4. Inter-VM communication is performed using Java Remote Method Invocation (RMI), which

Algorithm 3.1. Weighting algorithm

$\forall s \in S(k) | C_w(s,k) := w_c \cdot C(s,k) + (1 - w_c) \cdot \hat{C}(s,k)$
for $s \in S(k)$ **do**
  if $C_w(s,k) \geq C_{US}$ **then**
    $W(s,k) := 0$
  else if $C_w(s,k) > 0$ **then**
    $W(s,k) := \left\lceil W_{\text{MAX}} - \frac{C_w(s,k)}{C_{US}} \cdot W_{\text{MAX}} \right\rceil$
  else
    $W(s,k) := W_{\text{MAX}}$
  end if
end for

is a practical implementation of the Proxy pattern, performing distributed object communication: the object-oriented equivalent of Remote Procedure Call (RPC).

An alternative to RMI could be the Remote Open Services Gateway initiative (OSGi) specification [28], implemented in both Apache CXF and Eclipse ECF. This was not attempted, as it would have taken more time to implement. However, this approach might be easier to maintain. It would also be possible to use a Representational State Transfer (REST) interface through HTTP, which could make it easier to interface with the inner workings of the platform.

3.3.3 Admission Controller

The admission controller is responsible for admitting users to application servers. It supplements the load balancer in ensuring that the servers do not become overloaded by deciding whether to admit, defer, or reject traffic. It makes admission control decisions per session, not per request. This allows for a smoother user experience in a stateful environment, as a user of an application would not enjoy suddenly having requests to the application denied, when everything was working fine a moment ago. The admission controller implements per-session admission control. Unlike the traditional on-off approach, which makes admission control decisions on an interval basis, the per-session admission approach is not as vulnerable to sudden traffic fluctuations. The on-off approach can lead to servers becoming overloaded if they are set to admit traffic and a sudden traffic spike occurs [7]. The admission control decisions are based on prediction of future load trends combined with server health monitoring, as explained in Section 3.4.4.

3.3.4 Cloud Provisioner

The cloud provisioner is an external component, which represents the control service of the underlying IaaS provider. The global controller communicates with the cloud provisioner through its custom Application Programming Interface (API) in order to realize the decisions on how to manage the server tier. Proper application of the façade pattern decouples the proposed approach from the underlying IaaS provider. The prototypes [1, 9, 8, 11] currently support Amazon Elastic Compute Cloud (EC2) in homogeneous configurations. For now, we only provision *m1.small* instances, as our workloads are quite small, but the instance type can be changed easily. Provisioning VMs of different capacity could eventually lead to better granularity and lower operating costs. Support for more providers and heterogeneous configurations is planned for the future.

3.3.5 Entertainment Server

The entertainment server acts as a default service, which is used whenever a requested service is unavailable. It amounts to a polling session, notifying the user when the requested service is available and showing a waiting message or other distraction until then. Using server push technology or websockets, the entertainment server could be moved to the client instead.

3.3.6 Application Server

The application servers are dynamically provisioned VMs belonging to the underlying IaaS cloud, capable of concurrently running multiple applications inside an OSGi environment [27]. The prototype implementations of ARVUE [1, 9, 11] and CRAMP [8] use Apache Felix, which is a free implementation of the OSGi R4 Service Platform and other related technologies [35].

The OSGi specifications were originally intended for embedded devices, but have since outgrown their original purpose. They provide a dynamic component model, addressing a major shortcoming of Java. Figure 3.3 illustrates the OSGi architecture.

Each application server has a local controller, responsible for monitoring the state of said server. Metrics such as CPU load and memory usage of both the VM and of the individual deployed applications are collected and fed to the global controller for further processing. The global controller delegates application-tier tasks such as deployment and undeployment of bundles to the local controllers, which are responsible for notifying the OSGi environment of any actions to take.

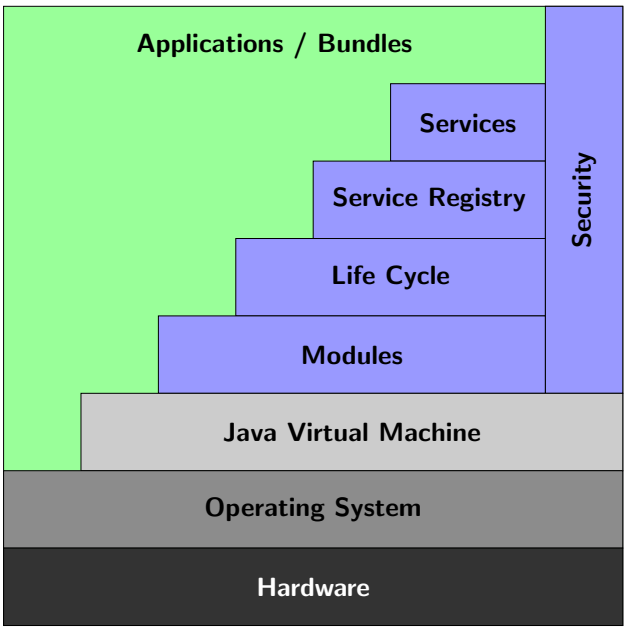


Figure 3.3: The OSGi platform.

The predictor from CRAMP [8] is also connected to each application server, making predictions based on the values obtained through the two-step prediction process. The prototype implementation computes an error estimate based on the NRMSE of predictions in the *past window* and uses that as a weighting parameter when determining how to blend the predicted and observed utilization of the monitored resources, as explained in Section 3.4.1.

3.3.7 Application Repository

Application bundles are contained in an application repository. When an application is deployed to a server, the server fetches the bundle from the repository. This implies that the repository is shared among application servers. A newly provisioned application server is assigned an application repository by the global controller. The applications are self-contained OSGi bundles, which allows for dynamic loading and unloading of bundles at the discretion of the local controller. The service-oriented nature of the OSGi platform suits this approach well.

A bundle is a collection of Java classes and resources together with a

Listing 3.1: Example manifest file with OSGi headers.

```
Bundle-Name: Hello World
Bundle-SymbolicName: org.arvue.helloworld
Bundle-Description: A Hello World bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0.0
Bundle-Activator: org.arvue.helloworld.Activator
Export-Package: org.arvue.helloworld;version="1.0.0"
Import-Package: org.osgi.framework;version="1.3.0"
```

manifest file `MANIFEST.MF` augmented with OSGi headers. Listing 3.1 shows an example manifest file complete with headers.

3.4 Algorithms

The VM provisioning algorithms used by the global controller constitute a hybrid reactive-proactive PD-controller [8]. They implement proportional scaling augmented with derivative control in order to react to changes in the health of the system [9]. The server tier can be scaled independently of the application tier in a shared hosting environment. The VM provisioning algorithms are supplemented by a set of allocation policies. The prototype currently supports the following policies: *lowest memory utilization*, *lowest CPU load*, *least concurrent sessions*, and *newest server first*. In addition to this, we have also developed an admission control algorithm [7]. A summary of the concepts and notations used to describe the VM provisioning algorithms is available in Table 3.1. The additional concepts and notations for the admission control algorithm are provided in Table 3.2.

The input variables are average CPU load and memory usage. Average CPU load is the average Unix-like system load, which is based on the queue length of runnable processes, divided by the number of CPU cores present.

The VM provisioning algorithms have been designed to prevent oscillations in the size of the application server pool. There are several motivating factors behind this choice. Firstly, provisioning VMs takes substantial time. Combined with frequent scaling operations, this may lead to bad performance [39]. Secondly, usage based billing requires the time to be quantized at some resolution. For example, Amazon EC2 bases billing on full used hours. Therefore, it might not make sense to terminate a VM until it is close to a full billing hour, as it is impossible to pay for less than an entire hour. Thus, no scaling actions are taken until previous operations have been completed. This is why an underutilized server is terminated only after being consistently underutilized for at least UC_T consecutive iterations.

Table 3.1: Summary of VM provisioning concepts and their notation

$A(k)$	set of web applications at time k
$A_i(k)$	set of inactive applications at time k
$A_{li}(k)$	set of long-term inactive applications at time k
$A_{over}(k)$	set of overloaded applications at time k
$S(k)$	set of servers at time k
$S_{lu}(k)$	set of long-term underutilized servers at time k
$S_n(k)$	set of new servers at time k
$S_{over}(k)$	set of overloaded servers at time k
$S_{\neg over}(k)$	set of non-overloaded servers at time k
$S_t(k)$	set of servers selected for termination at time k
$S_u(k)$	set of underutilized servers at time k
$C(a,k)$	measured CPU utilization of application a at time k
$C(s,k)$	measured load average of server s at time k
$\hat{C}(s,k)$	predicted load average of server s at time k
$C_w(s,k)$	weighted load average of server s at time k
$dep\_apps(s,k)$	applications deployed on server s at time k
$inactive\_c(a)$	inactivity count of application a
$M(a,k)$	measured memory utilization of application a at time k
$M(s,k)$	measured memory utilization of server s at time k
$\hat{M}(s,k)$	predicted memory utilization of server s at time k
$M_w(s,k)$	weighted memory utilization of server s at time k
$under\_u\_c(s)$	underutilization count of server s
$W(s,k)$	weight of server s at time k for load balancing
A_A	aggressiveness factor for additional capacity
A_P	aggressiveness factor for VM provisioning
A_T	aggressiveness factor for VM termination
$P_P(k)$	proportional factor for VM provisioning
$D_P(k)$	derivative factor for VM provisioning
$P_T(k)$	proportional factor for VM termination
$D_T(k)$	derivative factor for VM termination
w_c	weighting coefficient for CPU load average
w_m	weighting coefficient for memory usage
w_p	weighting coefficient for VM provisioning
w_t	weighting coefficient for VM termination
C_{LA}	application CPU utilization lower threshold
C_{LS}	server load average lower threshold
C_{UA}	application CPU utilization upper threshold
C_{US}	server load average upper threshold
IC_{TA}	inactivity count threshold for an application
IC_{TS}	inactivity count threshold for a server
M_{LA}	application memory utilization lower threshold
M_{LS}	server memory utilization lower threshold
M_{UA}	application memory utilization upper threshold
M_{US}	server memory utilization upper threshold
W_{MAX}	maximum value of a server weight for load balancing
$N_A(k)$	number of additional servers at time k
N_B	number of servers to use as base capacity
$N_P(k)$	number of servers to provision at time k
$N_T(k)$	number of servers to terminate at time k

Table 3.2: Additional concepts and notation for admission control

$se_a(k)$	set of aborted sessions at time k
$se_d(k)$	set of deferred sessions at time k
$se_n(k)$	set of new session requests at time k
$se_r(k)$	set of rejected sessions at time k
$S_{open}(k)$	set of open application servers at time k
$C(ent, k)$	load average of the entertainment server at time k
$M(ent, k)$	memory utilization of the entertainment server at time k
w	weighting coefficient for admission control

The memory usage metric $M(s, k)$ for a server s at discrete time k is given in (3.1). It is based on the amount of free memory mem_{free}, the size of the disk cache mem_{cache}, the buffers mem_{buf}, and the total memory size mem_{total}. The disk cache mem_{cache} is excluded from the amount of used memory, as the underlying operating system is at liberty to use free memory for such purposes as it sees fit. It will automatically be reduced as the demand for memory increases. The goal is to keep $M(s, k)$ below the server memory utilization upper threshold M_{US}. Likewise, the memory usage metric for an application a at discrete time k is defined as $M(a, k)$, which is the amount of the memory used by the application deployment plus the memory used by the user sessions divided by the total memory size mem_{total}.

$$M(s,k) = \frac{mem_{total} - (mem_{free} + mem_{buf} + mem_{cache})}{mem_{total}} \tag{3.1}$$

The proposed approach maintains a fixed minimum number of application servers, known as the *base capacity* N_B. In addition, it also maintains a dynamically adjusted number of additional application servers $N_A(k)$, which is computed as in (3.2), where the aggressiveness factor $A_A \in [0, 1]$ restricts the additional capacity to a fraction of the total capacity, $S(k)$ is the set of servers at time k, and $S_{over}(k)$ is the set of overloaded servers at time k. This extra capacity is needed to account for various delays and errors, such as VM provisioning time and sampling frequency. For example, $A_A = 0.2$ restricts the maximum number of additional application servers to 20% of the total $|S(k)|$.

$$N_A(k) = \begin{cases} \lceil |S(k)| \cdot A_A \rceil, & \text{if } |S(k)| - |S_{over}(k)| = 0 \\ \left\lceil \frac{|S(k)|}{|S(k)| - |S_{over}(k)|} \cdot A_A \right\rceil, & \text{otherwise} \end{cases} \tag{3.2}$$

The number of VMs to provision $N_P(k)$ is determined by (3.3), where $w_p \in [0, 1]$ is a real number called the weighting coefficient for VM provision-

ing. It balances the influence of the proportional factor $P_P(k)$ relative to the derivative factor $D_P(k)$. For example, $w_p = 0.5$ would give equal weight to $P_P(k)$ and $D_P(k)$. A suitable value for this coefficient should be determined experimentally for a given workload. We have used $w_p = 0.5$ in all our experiments so far. The proportional factor $P_P(k)$ given by (3.4) uses a constant aggressiveness factor for VM provisioning $A_P \in [0, 1]$, which determines how many VMs to provision. The derivative factor $D_P(k)$ is defined by (3.5). It observes the change in the total number of overloaded servers between the previous and the current iteration.

$$N_P(k) = \lceil w_p \cdot P_P(k) + (1 - w_p) \cdot D_P(k) \rceil \tag{3.3}$$
$$P_P(k) = |S_{over}(k)| \cdot A_P \tag{3.4}$$
$$D_P(k) = |S_{over}(k)| - |S_{over}(k-1)| \tag{3.5}$$

The number of servers to terminate $N_T(k)$ is computed as in (3.6). It uses a weighting coefficient for VM termination $w_t \in [0, 1]$, similar to w_p in (3.3). The currently required base capacity N_B and additional capacity $N_A(k)$ have to be taken into account. The proportional factor for termination $P_T(k)$ is calculated as in (3.7). Here $A_T \in [0, 1]$, the aggressiveness factor for VM termination, works like A_P in (3.4). Finally, the derivative factor for termination $D_T(k)$ is given by (3.8), which observes the change in the number of long-time underutilized servers between the previous and the current iteration.

$$N_T(k) = \lceil w_t \cdot P_T(k) + (1 - w_t) \cdot D_T(k) \rceil - N_B - N_A(k) \tag{3.6}$$
$$P_T(k) = |S_{lu}(k)| \cdot A_T \tag{3.7}$$
$$D_T(k) = |S_{lu}(k)| - |S_{lu}(k-1)| \tag{3.8}$$

3.4.1 Load Prediction

Prediction is performed with a two-step method [4, 5] based on EMA, which filters the monitored resource trends, producing a smoother curve. EMA is the weighted mean of the n samples in the past window, where the weights decrease exponentially. Figure 3.4 illustrates an EMA over a past window of size $n = 20$, where less weight is given to old samples when computing the mean in each measure.

As we use a hybrid reactive-proactive VM provisioning algorithm, there is a need to blend the measured and predicted values. This is done through linear interpolation [7] with the weights w_c and w_m [8], the former for CPU

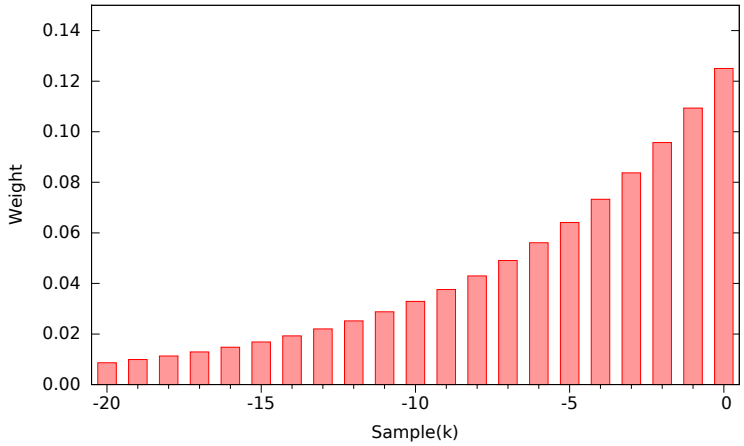


Figure 3.4: Example of EMA over a past window of size $n = 20$, where less weight is given to old samples when computing the mean in each measure.

load average and the latter for memory usage. In the current implementation, each of these weights is set to the NRMSE of the predictions so that lower prediction error will favor predicted values over observed values. The NRMSE calculation is given by (3.9), where y_i is the latest measured utilization, $\hat{y}_i$ is the latest predicted utilization, n is the number of observations, and max is the maximum value of both measured and observed utilizations formed over the current interval, while min is analogous to max. More details of our load prediction approach are provided in [7, 8].

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}}{max - min} \tag{3.9}$$

3.4.2 The Server Tier

The server tier consists of the application servers, which can be dynamically added to or removed from the cluster. The VM provisioning algorithm for the application server tier is presented in Algorithm 3.2. At each sampling interval k, the global controller retrieves the performance metrics from each of the local controllers, evaluates them and decides whether or not to take an action. The set of application servers is partitioned into disjoint subsets according to the current state of each server. The possible server states are: *overloaded*, *non-overloaded*, *underutilized*, and *long-term underutilized*.

Algorithm 3.2. Proactive VM provisioning for the application server tier

1: **while true do**
2: $\forall s \in S(k) | C_w(s,k) := w_c \cdot C(s,k) + (1 - w_c) \cdot \hat{C}(s,k)$
3: $\forall s \in S(k) | M_w(s,k) := w_m \cdot M(s,k) + (1 - w_m) \cdot \hat{M}(s,k)$
4: $S_{over}(k) := \{\forall s \in S(k) | C_w(s,k) \geq C_{US}\} \cup \{\forall s \in S(k) | M_w(s,k) \geq M_{US}\}$
5: $A_{over}(k) := \bigcup_{s \in S_{over}(k)} d\_a(s,k)$
6: $S_{\neg over}(k) := S(k) \setminus S_{over}(k)$
7: **if** $|S_{over}(k)| \geq 1 \wedge |S_{\neg over}(k)| \geq 1$ **then**
8: **for** $a \in A_{over}(k)$ **do**
9: deploy application a as per application-to-server allocation policy
10: **end for**
11: **end if**
12: **if** $|S_{over}(k)| \geq (|S(k)| - N_A(k)) \wedge N_P(k) \geq 1$ **then**
13: provision $N_P(k)$ VMs as a set of new servers $S_n(k)$
14: $S(k) := S(k) \cup S_n(k)$
15: Wait until servers $S_n(k)$ become operational
16: **for** $a \in A_{over}(k)$ **do**
17: deploy application a on servers $S_n(k)$
18: **end for**
19: **end if**
20: $S_u(k) := \{\forall s \in S(k) | C_w(s,k) \leq C_{LS}\} \cap \{\forall s \in S(k) | M_w(s,k) \leq M_{LS}\}$
21: $S_{lu}(k) := \{\forall s \in S_u(k) | under\_u\_c(s) \geq IC_{TS}\}$
22: **if** $(|S_{lu}(k)| - N_B - N_A(k)) \geq 1 \wedge N_T(k) \geq 1$ **then**
23: sort servers $S_{lu}(k)$ with respect to server utilization metrics
24: select $N_T(k)$ servers from $S_{lu}(k)$ as servers selected for termination $S_t(k)$
25: migrate all applications and user sessions from servers $S_t(k)$
26: $S(k) := S(k) \setminus S_t(k)$
27: terminate VMs for servers $S_t(k)$
28: **end if**
29: **end while**

The algorithm starts by partitioning the set of application servers into a set of overloaded servers $S_{over}(k)$ and a set of non-overloaded servers $S_{\neg over}(k)$ according to the supplied threshold levels (C_{US} and M_{US}) of the observed input variables: memory utilization and CPU load (lines 2–4). A server is overloaded if the utilization of any resource exceeds its upper threshold value. All other servers are considered to be non-overloaded (line 6). The applications running on overloaded servers are added to a set of overloaded applications $A_{over}(k)$ to be deployed on any available non-overloaded application servers as per the allocation policy for applications to servers (line 5). If the number of overloaded application servers exceeds the threshold level, a proportional amount of virtualized application servers is provisioned (line 13)

and the overloaded applications are deployed to the new servers as they become available (lines 16–18).

The server tier is scaled down by constructing a set of underutilized servers $S_u(k)$ (line 20) and a set of long-term underutilized servers $S_{lu}(k)$ (line 21), where servers are deemed idle if their utilization levels lie below the given lower thresholds (C_{LS} and M_{LS}). Long-term underutilized servers are servers that have been consistently underutilized for more than a given number of iterations IC_{TS}. When the number of long-term underutilized servers exceeds the base capacity N_B plus the additional capacity $N_A(k)$ (line 22), the remainder are terminated after their active sessions have been migrated to other servers (lines 23–27).

3.4.3 The Application Tier

Applications can be scaled to run on many servers according to their individual demand. Due to memory constraints, the naïve approach of always running all applications on all servers is unfeasible. Algorithm 3.3 shows how individual applications are scaled up and down according to their resource utilization. The set of applications is partitioned into disjoint subsets according to the current state of each application. The possible application states are: *overloaded*, *non-overloaded*, *inactive* and *long-term inactive*.

Algorithm 3.3. Reactive scaling of applications

1: **while true do**
2: $A_{over}(k) := \{s \in S(k), a \in dep\_apps(s,k) \mid C(a,k) > C_{UA}/|dep\_apps(s,k)| \vee M(a,k) > M_{UA}\}$
3: **if** $|A_{over}(k)| \geq 1$ **then**
4: **for all** $a \in A_{over}(k)$ **do**
5: deploy application a as per application-to-server allocation policy
6: **end for**
7: **end if**
8: $A_i(k) := \{s \in S(k), a \in dep\_apps(s,k) \,|C(a,k) < C_{LA} \wedge M(a,k) < M_{LA}\}$
9: $A_{li}(k) := \{a \in A_i(k) \,|\, inactive\_c(a) \geq IC_{TA}\}$
10: **if** $|A_{li}(k)| \geq 1$ **then**
11: migrate all applications and user sessions for applications $A_{li}(k)$
12: $A(k) := A(k) \setminus A_{li}(k)$
13: **for all** $a \in A_{li}(k)$ **do**
14: unload application a
15: **end for**
16: **end if**
17: **end while**

An application is overloaded when it uses more resources than allotted (line 2). Each overloaded application $a \in A_{over}(k)$ is deployed to another server according to the allocation policy for applications to servers (lines 4–6). When an application has been running on a server without exceeding the lower utilization thresholds (C_{LA} and M_{LA}), possible active sessions are migrated to another deployment of the application and then said application is undeployed (lines 8–15). This makes the memory available to other applications that might need it.

3.4.4 Admission Control

The admission control algorithm is given as Algorithm 3.4. It continuously checks for new $se_n(k)$ or deferred sessions $se_d(k)$ (line 1). If any are found (line 2), it updates the weighting coefficient $w \in [0, 1]$, representing the weight given to predicted and observed utilizations (line 3). If $w = 1.0$, no predictions are calculated (lines 5–6). The prediction process uses a two-step approach, providing filtered input data to the predictor [5]. We currently perform automatic adjustment and tuning in a similar fashion to Cherkasova and Phaal [13], where the weighting coefficient w is defined according to (3.10). It is based on the following metrics: number of aborted sessions $|se_a(k)|$, number of deferred sessions $|se_d(k)|$, number of rejected sessions $|se_r(k)|$, and number of overloaded servers $|S_{over}(k)|$.

$$w = \begin{cases} 1, & \text{if } |se_a(k)| > 0 \vee |se_d(k)| > 0 \vee |se_r(k)| > 0 \\ 1, & \text{if } |S_{over}(k)| > 0 \\ max(0.1, w - 0.01), & \text{otherwise} \end{cases} \tag{3.10}$$

For each iteration, a bit more preference is given to the predicted values, up to the limit of 90 %. However, as soon as a problem is detected, full preference is given to the observed values, as the old predictions cannot be trusted. This should help in reducing lag when there are sudden changes in the load trends after long periods of good predictions.

If the algorithm finds servers in good condition (line 12), the session is admitted (lines 13–17), else the session is deferred to the entertainment server (line 20). Only if also the entertainment server is overloaded, will the session be rejected (line 22).

Algorithm 3.4. Admission control

while true do
  if $|se_n(k)| \geq 1 \vee |se_d(k)| \geq 1$ **then**
    update the weighting coefficient w according to (3.10)
    if $w = 1$ **then**
      $\forall s \in S(k) | C_w(s,k) := C(s,k)$
      $\forall s \in S(k) | M_w(s,k) := M(s,k)$
    else
      $\forall s \in S(k) | C_w(s,k) := w \cdot C(s,k) + (1-w) \cdot \hat{C}(s,k)$
      $\forall s \in S(k) | M_w(s,k) := w \cdot M(s,k) + (1-w) \cdot \hat{M}(s,k)$
    end if
    $S_{open}(k) := \{\forall s \in S(k) | C_w(s,k) < LA_{UT} \wedge M_w(s,k) < MU_{UT}\}$
    if $|S_{open}(k)| \geq 1$ **then**
      if $|se_d(k)| \geq 1$ **then**
        pop first session in $se_d(k)$ and admit it on a server in $S_{open}(k)$
      else
        pop first session in $se_n(k)$ and admit it on a server in $S_{open}(k)$
      end if
    else if $|se_n(k)| \geq 1$ **then**
      if $C(ent,k) < LA_{UT} \wedge M(ent,k) < MU_{UT}$ **then**
        pop first session in $se_n(k)$ and defer it
      else
        pop first session in $se_n(k)$ and reject it
      end if
    end if
  end if
end while

3.5 Experimental Evaluation

To validate and evaluate the proposed VM provisioning and admission control approaches, we developed discrete-event simulations for ARVUE, CRAMP, and ACVAS and performed a series of experiments involving synthetic as well as realistic load patterns. The synthetic load pattern consists of two artificial load peaks, while the realistic load pattern is based on real world data. In this section, we present experimental results based on the discrete-event simulations.

3.5.1 VM Provisioning Experiments

This section presents some of the simulations and experiments that have been conducted to validate and evaluate ARVUE and CRAMP VM provisioning algorithms. The goal of these experiments was to test the two approaches and to compare their results.

In order to generate workload, a set of application users was needed. In our discrete-event simulations, we developed a load generator to emulate a given number of user sessions making HTTP requests on the web applications. We also constructed a set of 100 simulated web applications of varying resource needs, designed to require a given amount of work on the hosting server(s). When a new HTTP request arrived at an application, the application would execute a loop for a number of iterations, corresponding to the empirically derived time required to run the loop on an unburdened server. As the objective of the VM provisioning experiments was to compare the results of ARVUE and CRAMP, admission control was not used in these experiments.

Design and Setup

We performed two experiments with the proposed VM provisioning approaches: ARVUE and CRAMP. The first experiment used a synthetic load pattern, which was designed to scale up to 1000 concurrent sessions in two peaks with a period of no activity between them. In the second peak, the arrival rate was twice as high as in the first peak.

The second experiment was designed to simulate a load representing a workload trace from a real web-based system. The traces were derived from Squid proxy server access logs obtained from the IRCache project [24]. As the access logs did not include session information, we defined a session as a series of requests from the same originating Internet Protocol (IP)-address, where the time between individual requests was less than 15 minutes. We then produced a histogram of sessions per second and used linear interpolation and scaling by a factor of 30 to obtain the load traces used in the experiment.

In a real-world application, there would be different kinds of requests available, requiring different amounts of CPU time. Take the simple case of a web shop: there might be one class of requests for adding items to the shopping basket, requiring little CPU time, and another class of requests requiring more CPU time, like computing the sum total of the items in the shopping basket. Users of an application would make a number of varying requests through their interactions with the application. After each request, there would be a delay while the user was processing the newly retrieved

information, like when presented with a new resource. In both experiments, each user was initially assigned a random application and a session duration of 15 minutes. Application 1 to 10 were assigned to 50 % of all users, application 11 to 20 were used by 25 %, application 21 to 30 received 20 % of all users, while the remaining 5 % was shared among the other 70 applications. Each user made requests to its assigned application, none of which was to require more than 10 ms of CPU time on an idle server. In order to emulate the time needed for a human to process the information obtained in response to a request, the simulated users waited up to 20 s between requests. All random variables were uniformly distributed. This means they do not fit the Markovian model.

The sampling period was $k = 10$ s. The upper threshold for server load average C_{US} and the upper threshold for server memory utilization M_{US} were both set to 0.8. These values are considered reasonable for efficient server utilization [25, 2].

The application-server allocation policy used was *lowest load average*. The session-server allocation policy was also set to *lowest load average*, realized through the *weighted round-robin* policy of HAProxy, where the weights were assigned by the global controller according to the load averages of the servers, as described in Section 3.3.1.

Results and Analysis

The results from the VM provisioning experiment with the synthetic load pattern are shown in Figures 3.5a and 3.5b. The depicted observed parameters are: number of servers, average response time, average server CPU load, average memory utilization, and applications per server. The upper half of Table 3.3 contains a summary of the results.

The results from the two approaches are compared based on the following criteria: *number of servers used*, *average CPU load average*, *maximum CPU load average*, *average memory utilization*, *maximum memory utilization*, *average RTT*, and *maximum RTT*. The resource utilizations are ranked according to the utilization error, where over-utilization is considered infinitely bad.

In Figures 3.5a and 3.5b, the number of servers plots show that the number of application servers varied in accordance with the number of simultaneous user sessions. In this experiment, ARVUE used a maximum of 16 servers, whereas CRAMP used no more than 14 servers. The RTT remained quite stable around 20 ms, as expected. The server CPU load average and the memory utilization never exceeded 1.0.

The results from the experiment with the synthetic load pattern indicate

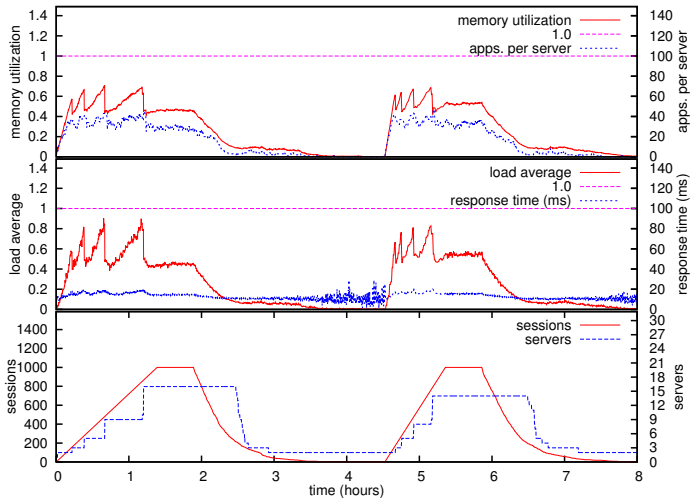
that the system is working as intended. The use of additional capacity seems to alleviate the problem of servers becoming overloaded due to long reaction times. The conservative VM termination policy of the proposed approach explains why the decrease in the number of servers occurs later than the decrease in the number of sessions. As mentioned in Section 3.4, one of the objectives of the proposed VM provisioning algorithms is to prevent oscillations in the number of application servers used. The results indicate that this was achieved.

Figures 3.6a and 3.6b present the results of the VM provisioning experiment with the realistic load pattern. The results are also presented in the lower half of Table 3.3.

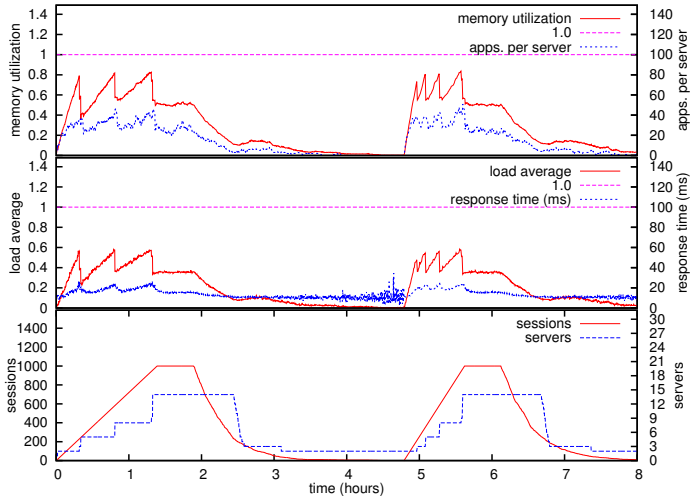
In this experiment, ARVUE used a maximum of 16 servers, whereas CRAMP used no more than 8 servers. In the case of ARVUE, the maximum response time was 21.3 ms and the average response time was 12.63 ms. In contrast, CRAMP had a maximum response time of 27.43 ms and an average response time of 14.7 ms. For both ARVUE and CRAMP, the server CPU load average and the memory utilization never exceeded 1.0.

The results from the experiment with the realistic load pattern show significantly better performance of CRAMP compared to ARVUE in terms of number of servers. CRAMP used half as many servers as ARVUE, but it still provided similar results in terms of average response time, CPU load average, and memory utilization. The ability to make predictions of future trends is a significant advantage, even if the predictions may not be fully accurate. Still, there were significant problems with servers becoming overloaded due to the provisioning delay. Increasing the safety margins further by lowering the upper resource utilization threshold values or increasing the extra capacity buffer further might not be economically viable. We suspect that an appropriate admission control strategy will be able to prevent the servers from becoming overloaded in an economically viable fashion.

Figure 3.7a shows the utilization error in the first experiment that uses the synthetic load pattern. For brevity, we only depict the CPU load in the error analysis. Therefore, error is defined as the absolute difference between the target CPU load average level C_{US} and the measured value of the CPU load average $C(s,k)$ averaged over all servers in the system. Initially, the servers are naturally underloaded due to the lack of work. Thereafter, as soon as the first peak of load arrives, the error shrinks significantly and becomes as low as 0.1 for ARVUE and 0.3 for CRAMP. The higher CPU load error for CRAMP at this point was due to the fact that CRAMP results in this experiment were mostly memory-driven, as can be seen in Figure 3.5b. In other words, CRAMP had higher error with respect to the CPU load, but it had lower error with respect to the memory utilization. The error grows again as the

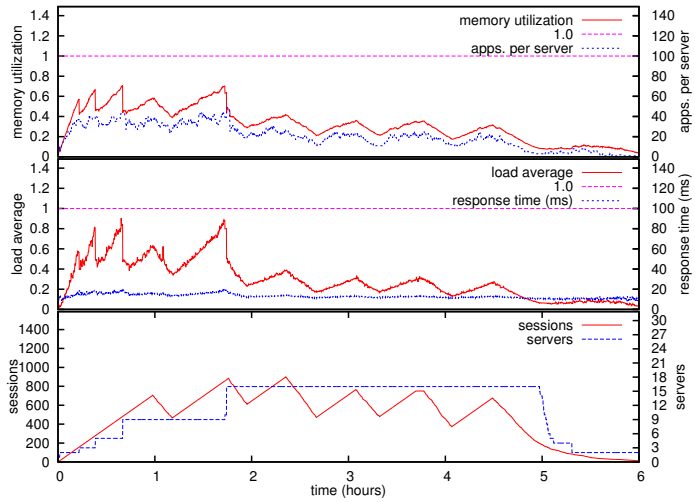


(a) Results of ARVUE with synthetic load.

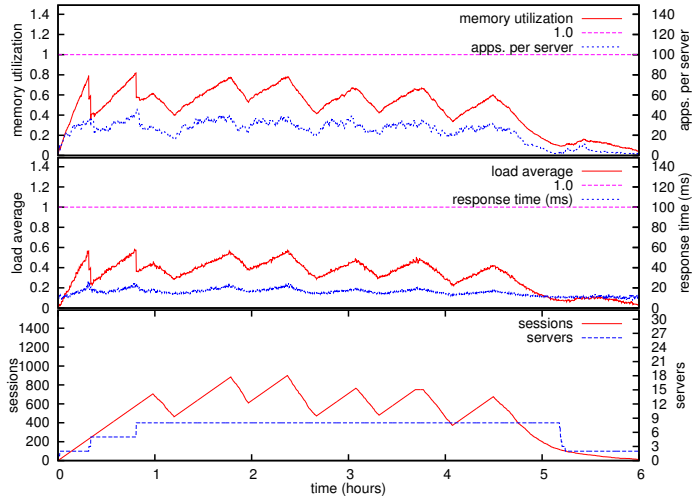


(b) Results of CRAMP with synthetic load.

Figure 3.5: Results of VM provisioning experiment with the synthetic load pattern. In this experiment, both ARVUE and CRAMP had similar results, except that CRAMP used fewer servers.



(a) Results of ARVUE with realistic load.



(b) Results of CRAMP with realistic load.

Figure 3.6: Results of VM provisioning experiment with the realistic load pattern. In this experiment, CRAMP used half as many servers as ARVUE, but it still provided similar performance.

Table 3.3: Results from VM provisioning experiments. The upper half of the table contains results from the first experiment with the synthetic load pattern, while the lower half contains results from the second experiment with the realistic load pattern. Entries in bold are better according to the evaluation criteria.

approach	servers	$load_{avg.}$	$load_{max}$	$mem_{avg.}$	mem_{max}	$RTT_{avg.}$	RTT_{max}
$ARVUE_{synth}$	16	**0.21**	0.9	0.21	**0.71**	**12.23** ms	**32.88** ms
$CRAMP_{synth}$	**14**	0.17	**0.58**	**0.25**	0.84	12.97 ms	34.72 ms
$ARVUE_{real}$	16	0.25	0.9	0.27	**0.71**	**12.63** ms	**21.3** ms
$CRAMP_{real}$	**8**	**0.28**	**0.58**	**0.4**	0.82	14.7 ms	27.43 ms

period of no activity starts after the first peak of load. In the second peak, both ARVUE and CRAMP showed similar results, where the error becomes as low as 0.25. Finally, as the request rate sinks after the second peak of load, the error grows further due to underutilization. This can be attributed to the intentionally cautious policy for scaling down, which is explained in Section 3.4 and ultimately to the lack of work. A more aggressive policy for scaling down might work without introducing oscillating behavior, but when using a third-party IaaS it would still not make sense to terminate a VM until the current billing interval is coming to an end, as that resource constitutes a sunk cost.

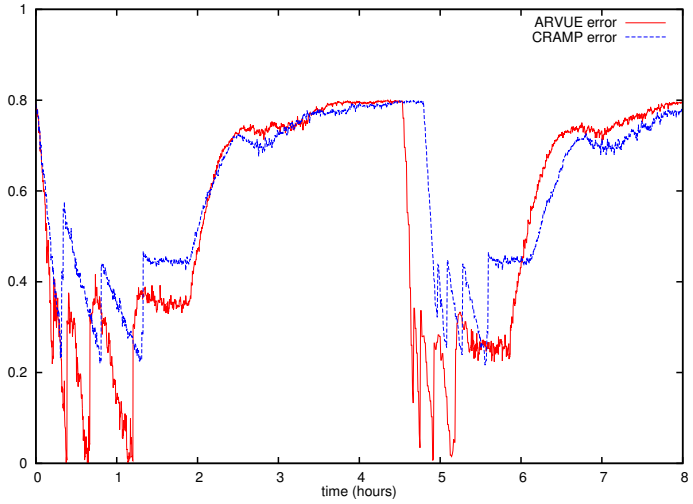
Error analysis of the second experiment that uses the realistic load pattern can be seen in Figure 3.7b. CRAMP appears to have lower error than ARVUE throughout most of the experiment, with the only exceptions being due to underutilization.

3.5.2 Admission Control Experiments

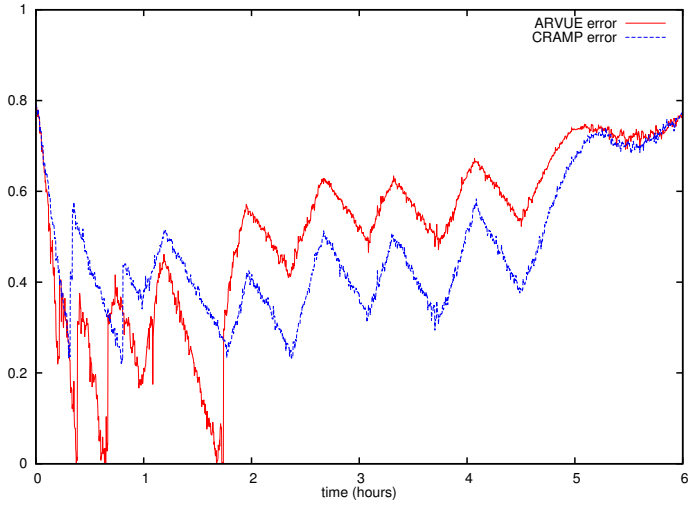
This section presents experiments with admission control. The goal of these experiments was to test our proposed admission control approach ACVAS [7] and to compare it against an existing SBAC implementation [13], here referred to as the *alternative approach*. As in the VM provisioning experiments, the experiments in this section also used 100 simulated web applications of various resource requirements. The experiments were conducted through discrete-event simulations.

Design and Setup

We performed two experiments with ACVAS and the *alternative approach*. The first admission control experiment used the synthetic load pattern, which



(a) Synthetic load pattern experiment: error analysis.



(b) Realistic load pattern experiment: error analysis.

Figure 3.7: CPU load average error analysis in the VM provisioning experiments. In the first experiment, CRAMP appears to have higher error because its results were mostly memory-driven. In the second experiment, CRAMP had lower error than ARVUE, with the only exceptions being due to underutilization.

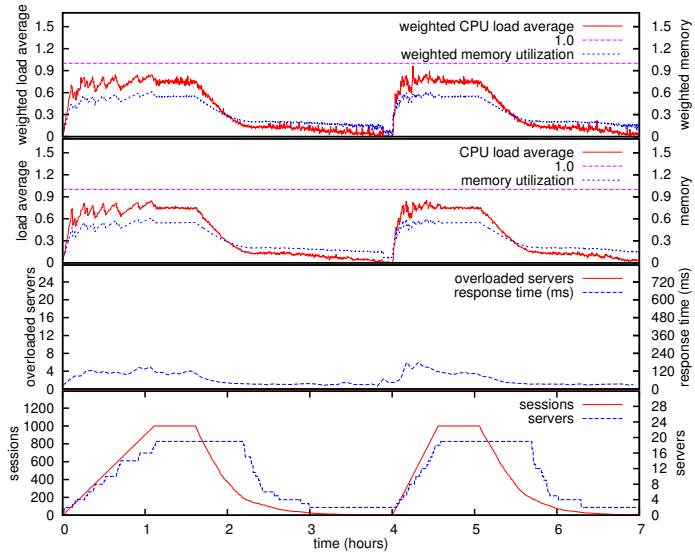
was also used in the first VM provisioning experiment described in Section 3.5.1. This workload was designed to scale up to 1000 concurrent sessions in two peaks with a period of no activity between them. Similarly, the second admission control experiment was designed to use the realistic load pattern, which was also used in the second VM provisioning experiment in Section 3.5.1. The sampling period k, the upper threshold for server load average C_{US}, the upper threshold for server memory utilization M_{US}, the application-server allocation policy, and the session-server allocation policy were all same as in the VM provisioning experiments in Section 3.5.1.

Results and Analysis

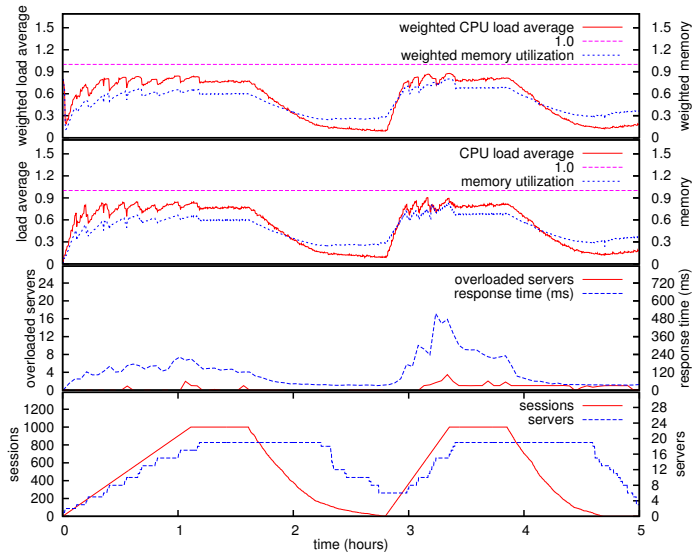
In our previous work [7], we proposed a way of measuring the quality of an admission control mechanism based on the trade-off between the number of servers used and six important QoS metrics: *zero overloaded servers*, *maximum achievable session throughput*, *zero aborted sessions*, *minimum deferred sessions*, *zero rejected sessions* and *minimum average response time* for all admitted sessions. The results from the two approaches will be compared based on these criteria.

Figures 3.8a and 3.8b present the results from the experiment with the synthetic load pattern. A summary of the results is also available in the upper half of Table 3.4. The prediction accuracy was high, the Root Mean Square Error (RMSE) of the predicted CPU and memory utilization was 0.0163 and 0.0128 respectively. ACVAS used a maximum of 19 servers with 0 overloaded servers, 0 aborted sessions, 30 deferred sessions, and 0 rejected sessions. There were a total of 8620 completed sessions with an average RTT of 59 ms. Thus, ACVAS provided a good trade-off between the number of servers and the QoS requirements. The *alternative approach* also used a maximum of 19 servers, but with several occurrences of server overloading. On average, there were 0.56 overloaded servers at all time with 0 aborted sessions and 488 rejected sessions. A total of 9296 sessions were completed with an average RTT of 112 ms. Thus, in the first experiment, the *alternative approach* completed 9296 sessions compared to 8620 sessions by ACVAS, but with 488 rejected sessions and several occurrences of server overloading.

Figures 3.9a and 3.9b show the results of the experiment with the realistic load trace derived from access logs. The lower half of Table 3.4 shows that ACVAS used a maximum of 16 servers with 0 overloaded servers, 0 aborted sessions, 20 deferred sessions, and 0 rejected sessions. There were a total of 8559 completed sessions with an average RTT of 59 ms. In contrast, the *alternative approach* used a maximum of 17 servers with 3 occurrences of server overloading. On average, there were 0.0046 overloaded servers at all



(a) Results of ACVAS with synthetic load.



(b) Results of *alternative approach* with synthetic load.

Figure 3.8: Results of admission control experiment with the synthetic load pattern. ACVAS performed better than the *alternative approach* in all aspects but session deferment and throughput.

Table 3.4: Results from admission control experiments. The upper half of the table contains results from the first experiment with the synthetic load pattern, while the lower half contains results from the second experiment with the realistic load pattern. Entries in bold are better according to the evaluation criteria.

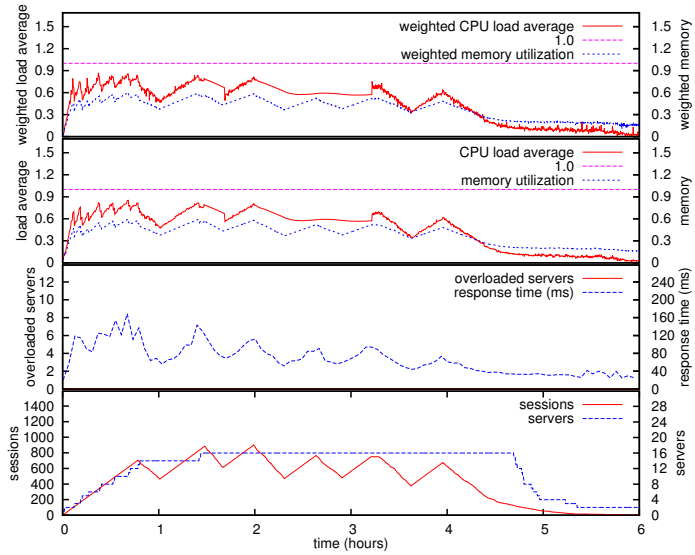
approach	servers	overl.	abort.	def.	rej.	compl.	$RTT_{avg.}$
$ACVAS_{synth}$	19	**0**	0	30	**0**	8620	**59** ms
$alternative_{synth}$	19	0.56	0	**N/A**	488	**9296**	112 ms
$ACVAS_{real}$	**16**	**0**	0	20	**0**	8559	**59** ms
$alternative_{real}$	17	0.0046	0	**N/A**	55	**8577**	72 ms

time with 0 aborted sessions and 55 rejected sessions. There were a total of 8577 completed sessions with an average RTT of 72 ms. Thus, the *alternative approach* used an almost equal number of servers, but it did not prevent them from becoming overloaded. Moreover, it completed 8577 sessions compared to 8559 sessions by ACVAS, but with 55 rejected sessions and 3 occurrences of server overloading.

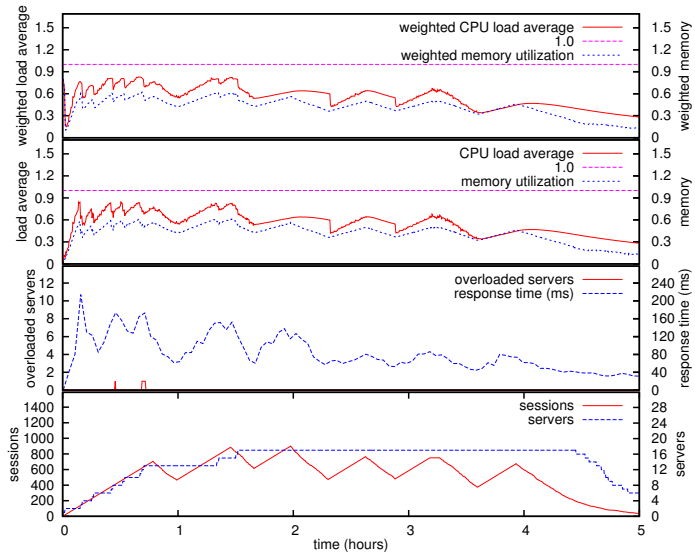
The results from these two experiments indicate that the ACVAS approach provides significantly better results in terms of the previously mentioned QoS metrics. In the first experiment, ACVAS had the best results in three areas: *overloaded servers*, *rejected sessions*, and *average RTT*. The *alternative approach* performed better in two areas: there were no *deferred sessions*, as it did not support session deferment, and it had more *completed sessions*. In the second experiment, ACVAS performed better in four aspects: *number of servers used*, *overloaded servers*, *rejected sessions*, and *average RTT*. The *alternative approach* again showed better performance in the number of *completed sessions* and in the number of *deferred sessions*. We can therefore conclude that ACVAS performed better than the *alternative approach* in both experiments.

The EMA-based predictor appears to be doing a good job on predicting these types of loads. It remains unclear how the system reacts to sudden drops in a previously increasing load trend. Such a scenario could temporarily lead to high preference for predicted results, which are no longer valid.

A plot of the utilization error with the synthetic load pattern can be seen in Figure 3.10a. Likewise, a plot of the utilization error with the realistic load can be seen in Figure 3.10b. Again, we only depict the CPU load, as it played the most significant part. The periods where ACVAS appears to have higher error than the *alternative approach* are due to underutilization amplified by ACVAS being more effective at keeping the average utilization



(a) Results of ACVAS with realistic load.



(b) Results of *alternative approach* with realistic load.

Figure 3.9: Results of admission control experiment with the realistic load pattern. ACVAS performed better than the *alternative approach* in all aspects but session deferment and throughput.

down, as no servers became overloaded during this time. Overall, the results are quite similar, as they should be, the only difference being the admission controller.

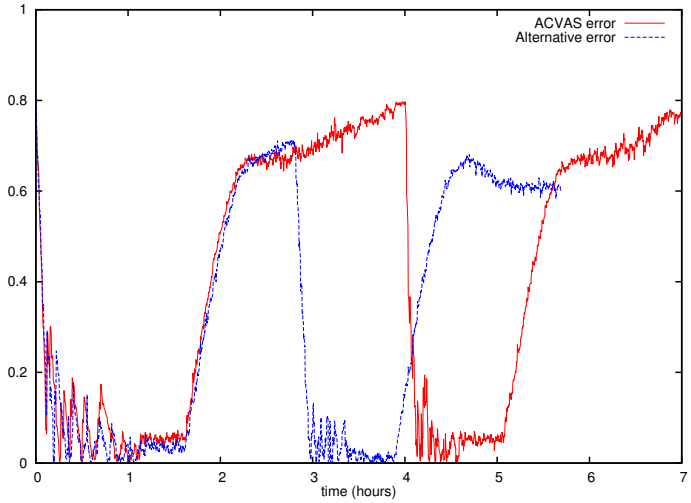
3.6 Conclusions

In this chapter, we presented a prediction-based, cost-efficient VM provisioning and admission control approach for multi-tier web applications. It provides automatic deployment and scaling of multiple simultaneous web applications on a given IaaS cloud in a shared hosting environment. The proposed approach comprises three sub-approaches: a reactive VM provisioning approach called ARVUE, a hybrid reactive-proactive VM provisioning approach called CRAMP, and a session-based adaptive admission control approach called ACVAS. Both ARVUE and CRAMP provide autonomous shared hosting of third-party Java Servlet applications on an IaaS cloud. However, CRAMP provides better responsiveness and results than the purely reactive scaling of ARVUE. ACVAS implements per-session admission, which reduces the risk of over-admission. Moreover, it implements a simple session deferment mechanism that reduces the number of rejected sessions while increasing session throughput. The proposed approach is demonstrated in discrete-event simulations and is evaluated in a series of experiments involving synthetic as well as realistic load patterns.

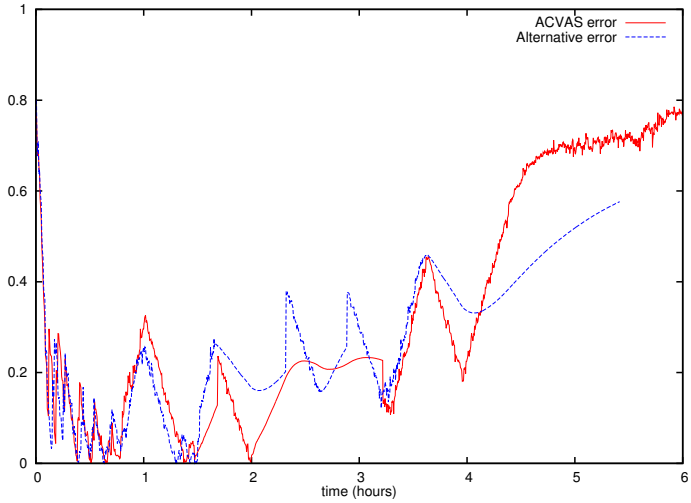
The results of the VM provisioning experiments showed that both ARVUE and CRAMP provide good performance in terms of average response time, CPU load average, and memory utilization. Moreover, CRAMP provides significantly better performance in terms of number of servers. It also had lower utilization error than ARVUE in most of the cases.

The evaluation and analyses concerning our proposed admission control approach compared ACVAS against an existing admission control approach available in the literature. The results indicated that ACVAS provides a good trade-off between the number of servers used and the QoS metrics. In comparison with the alternative admission control approach, ACVAS provided significant improvements in terms of server overload prevention, reduction of rejected sessions, and average response time.

Future work includes implementing and testing the admission controller on the prototype ARVUE PaaS [1, 11]. Furthermore, a case study of the final ARVUE PaaS could yield real data from an actual business case. We have been currently working on server consolidation approaches for web applications. Improved allocation through efficient consolidation should be possible. Moreover, applying metaheuristic approaches [10, 20] to optimize



(a) Synthetic load pattern experiment: error analysis.



(b) Realistic load pattern experiment: error analysis.

Figure 3.10: CPU load average error analysis in the admission control experiments. In the first experiment, both approaches had a similar error plot. However, in the second experiment, ACVAS appears to have lower error than the *alternative approach* throughout most of the experiment, with the only exceptions being due to underutilization.

cost-efficiency is also part of our ongoing research. However, optimal solutions can be seen as a form of the bin-packing problem and is therefore NP-complete [16].

Acknowledgments

This work was supported by an Amazon Web Services research grant. Adnan Ashraf was partially supported by the Foundation of Nokia Corporation and by a doctoral scholarship from the Higher Education Commission (HEC) of Pakistan.

References

[1] T. Aho et al. "Designing IDE as a Service". In: *Communications of Cloud Software* 1 (2011), pp. 1–10.

[2] J. Allspaw. *The Art of Capacity Planning: Scaling Web Resources*. O'Reilly Media, Inc., 2008. ISBN: 0596518579, 9780596518578.

[3] J. Almeida et al. "Joint admission control and resource allocation in virtualized servers". In: *J. Parallel Distrib. Comput.* 70.4 (Apr. 2010), pp. 344–362. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2009.08.009.

[4] M. Andreolini and S. Casolari. "Load prediction models in web-based systems". In: *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*. valuetools '06. New York, NY, USA: ACM, 2006. ISBN: 1-59593-504-5. DOI: 10.1145/1190095.1190129.

[5] M. Andreolini, S. Casolari, and M. Colajanni. "Models and Framework for Supporting Runtime Decisions in Web-Based Systems". In: *ACM Transactions on the Web* 2.3 (2008), pp. 1–43. ISSN: 1559-1131. DOI: 10.1145/1377488.1377491.

[6] D. Ardagna et al. "Service Provisioning on the Cloud: Distributed Algorithms for Joint Capacity Allocation and Admission Control". In: *Towards a Service-Based Internet*. Ed. by E. Di Nitto and R. Yahyapour. Vol. 6481. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 1–12.

[7] A. Ashraf, B. Byholm, and I. Porres. "A Session-Based Adaptive Admission Control Approach for Virtualized Application Servers". In: *The 5th IEEE/ACM International Conference on Utility and Cloud Computing*. Ed. by C. Varela and M. Parashar. IEEE Computer Society, 2012, pp. 65–72.

[8] A. Ashraf, B. Byholm, and I. Porres. "CRAMP: Cost-Efficient Resource Allocation for Multiple Web Applications with Proactive Scaling". In: *4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Ed. by T. W. Włodarczyk, C.-H. Hsu, and W.-C. Feng. IEEE Computer Society, 2012, pp. 581–586.

[9] A. Ashraf et al. "Feedback Control Algorithms to Deploy and Scale Multiple Web Applications per Virtual Machine". In: *38th Euromicro Conference on Software Engineering and Advanced Applications*. Ed. by V. Cortellessa, H. Muccini, and O. Demirors. IEEE Computer Society, 2012, pp. 431–438.

[10] C. Blum et al. "Hybrid metaheuristics in combinatorial optimization: A survey". In: *Applied Soft Computing* 11.6 (2011), pp. 4135 –4151. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2011.02.032.

[11] B. Byholm. "An Autonomous Platform as a Service for Stateful Web Applications". MA thesis. Åbo Akademi University, 2013.

[12] X. Chen, H. Chen, and P. Mohapatra. "ACES: An efficient admission control scheme for QoS-aware web servers". In: *Computer Communications* 26.14 (2003), pp. 1581–1593. ISSN: 0140-3664. DOI: 10.1016/S0140-3664(02)00259-1.

[13] L. Cherkasova and P. Phaal. "Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites". In: *Computers, IEEE Transactions on* 51.6 (2002), pp. 669–685. ISSN: 0018-9340. DOI: 10.1109/TC.2002.1009151.

[14] T. C. Chieu et al. "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment". In: *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*. 2009, pp. 281–286. DOI: 10.1109/ICEBE.2009.45.

[15] X. Dutreilh et al. "From Data Center Resource Allocation to Control Theory and Back". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 2010, pp. 410–417. DOI: 10.1109/CLOUD.2010.55.

[16] M. R. Garey and D. S. Johnson. ""Strong" NP-Completeness Results: Motivation, Examples, and Implications". In: *Journal of the ACM* 25.3 (1978), pp. 499–508. ISSN: 0004-5411. DOI: 10.1145/322077.322090.

[17] Google. *App Engine*. https://developers.google.com/appengine.

[18] M. Grönroos. *Book of Vaadin*. fourth. Vaadin Ltd, 2011.

[19] R. Han et al. "Lightweight Resource Scaling for Cloud Applications". In: *Cluster Computing and the Grid, IEEE International Symposium on* (2012), pp. 644–651.

[20] M. Harman et al. "Cloud engineering is Search Based Software Engineering too". In: *Journal of Systems and Software* 86.9 (2013), pp. 2225 –2241. ISSN: 0164-1212. DOI: http://dx.doi.org/10.1016/j.jss.2012.10.027.

[21] Y. Hu et al. "Resource provisioning for cloud computing". In: *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*. CASCON '09. New York, NY, USA: ACM, 2009, pp. 101–111.

[22] C.-J. Huang et al. "Admission control schemes for proportional differentiated services enabled internet servers using machine learning techniques". In: *Expert Systems with Applications* 31.3 (2006), pp. 458–471. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2005.09.071.

[23] W. Iqbal et al. "Adaptive resource provisioning for read intensive multi-tier applications in the cloud". In: *Future Generation Computer Systems* 27.6 (2011), pp. 871–879. ISSN: 0167-739X.

[24] *IRCache Project Squid Logs*. http://www.ircache.net/.

[25] H. H. Liu. *Software Performance and Scalability: A Quantitative Approach*. Wiley Publishing, 2009. ISBN: 0470462531, 9780470462539.

[26] S. Muppala and X. Zhou. "Coordinated Session-Based Admission Control with Statistical Learning for Multi-Tier Internet Applications". In: *Journal of Network and Computer Applications* 34.1 (2011), pp. 20–29. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2010.10.007.

[27] OSGi Alliance. *OSGi Service Platform Core Specification, Release 4, Version 4.2*. AQute Publishing, 2010.

[28] OSGi Alliance. *OSGi Service Platform Enterprise Specification, Release 4, Version 4.2*. AQute Publishing, 2010.

[29] W. Pan et al. “Feedback Control-Based QoS Guarantees in Web Application Servers”. In: *High Performance Computing and Communications, 2008. HPCC ’08. 10th IEEE International Conference on*. 2008, pp. 328–334. DOI: 10.1109/HPCC.2008.106.

[30] T. Patikirikorala et al. “A multi-model framework to implement self-managing control systems for QoS management”. In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’11. New York, NY, USA: ACM, 2011, pp. 218–227. ISBN: 978-1-4503-0575-4.

[31] Y. Raivio et al. “Hybrid Cloud Architecture for Short Message Services”. In: *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*. Ed. by F. Leymann et al. SciTePress, 2012, pp. 489–500.

[32] A. Robertsson et al. “Admission control for Web server systems - design and experimental evaluation”. In: *Decision and Control, 2004. CDC. 43rd IEEE Conference on*. Vol. 1. 2004, pp. 531–536. DOI: 10.1109/CDC.2004.1428685.

[33] Y. A. Shaaban and J. Hillston. “Cost-based admission control for Internet Commerce QoS enhancement”. In: *Electronic Commerce Research and Applications* 8.3 (2009), pp. 142–159. ISSN: 1567-4223. DOI: 10.1016/j.elerap.2008.11.007.

[34] W. Tarreau. *HAProxy*. http://haproxy.1wt.eu/.

[35] The Apache Software Foundation. *Apache Felix*. http://felix.apache.org/site/.

[36] B. Urgaonkar, P. Shenoy, and T. Roscoe. “Resource Overbooking and Application Profiling in a Shared Internet Hosting Platform”. In: *ACM Trans. Internet Technol.* 9.1 (Feb. 2009), pp. 1–45. ISSN: 1533-5399. DOI: 10.1145/1462159.1462160.

[37] W. Vogels. “Beyond Server Consolidation”. In: *Queue* 6.1 (Jan. 2008), pp. 20–26. ISSN: 1542-7730. DOI: 10.1145/1348583.1348590.

[38] T. Voigt and P. Gunningberg. “Adaptive resource-based Web server admission control”. In: *Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on*. 2002. DOI: 10.1109/ISCC.2002.1021682.

[39] A. Wolke and G. Meixner. “TwoSpot: A Cloud Platform for Scaling out Web Applications Dynamically”. In: *Towards a Service-Based Internet*. Ed. by E. di Nitto and R. Yahyapour. Vol. 6481. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 13–24. ISBN: 978-3-642-17693-7.

4 Proactive Virtual Machine Allocation for Video Transcoding in the Cloud

Fareed Jokhio, Adnan Ashraf, Tewodros Deneke, Sébastien Lafond,
Ivan Porres, and Johan Lilius
Department of Information Technologies
Åbo Akademi University, Turku, Finland
Email: {fjokhio, aashraf, tdeneke, slafond, iporres, jolilius}@abo.fi

Abstract–Video transcoding refers to the process of converting a digital video from one compressed format to another. It is a compute-intensive operation. Therefore, transcoding of a large number of simultaneous video streams requires a large-scale distributed system. Moreover, to handle different load conditions in a cost-efficient manner, the distributed system should be dynamically scalable. Infrastructure as a Service (IaaS) clouds currently offer computing resources, such as Virtual Machines (VMs), under the pay-per-use business model, which can be used to create a dynamically scalable cluster of video transcoding servers. Determining the number of VMs to provision for a dynamic cluster is still an open problem. In this chapter, we present a proactive VM allocation approach to scale video transcoding service on a given IaaS cloud. For better resource utilization, quality of service, and cost-efficiency, we use video segmentation at the Group of Pictures (GOP) level. The proposed approach is demonstrated in discrete-event simulations and an experimental evaluation involving different load patterns. We also present a prototype implementation of a distributed video transcoder based on the message passing programming model and a dynamic load balancing approach.

Keywords-Cloud computing, resource allocation, video transcoding.

4.1 Introduction

The use of multimedia content is common in our life. It may consist of digital images, videos, voice, or animation. The use of video is no longer limited to TV-channels or cinema theaters. There are several user-friendly and inexpensive devices available such as cell phones, digital cameras, which can be used to capture, manipulate and store digital videos. Electronics devices such as digital computers are used to process video data very efficiently. Video production is common nowadays and a large number of digital videos are uploaded on YouTube[1] and other video hosting sites. To store and transmit a digital video in a cost-efficient manner, video compression is used. Video compression is a mature field and several video coding standards are available such as MPEG-4 [36] and H.264 [37]. However, end-user devices do not support all video compression formats. Therefore, an unsupported video format needs to be converted into another format, which is supported by the target device. Converting a compressed video into another compressed video is known as video transcoding [35]. There are different types of video transcoding, such as bit-rate reduction in order to meet network bandwidth availability, resolution reduction for display size adoption, temporal transcoding for frame rate reduction, and error resilience transcoding for insuring high Quality of Service (QoS) [13], [39].

Video transcoding involves decoding and encoding processes. It is a compute-intensive process, usually performed at the server-side. It may be done in real-time or in batch processing. However, for an on-demand video streaming service, if the required video is not available in the desired format, the transcoding needs to be done on-the-fly in real-time. One of the main challenges of a real-time video transcoding operation is that it must avoid over and underflow of the output video buffer, which temporarily stores the transcoded videos at the server-side. The overflow occurs if the video transcoding rate exceeds the video play rate and the capacity of the buffer. Likewise, the buffer underflow may occur when the play rate exceeds the transcoding rate, while the buffer does not contain enough frames either, to avoid the underflow situation.

Video transcoding of a large number of video streams requires a large-scale cluster-based distributed system. Moreover, to handle varying amounts of load in a cost-efficient manner, the cluster should be dynamically scalable. Cloud computing provides theoretically infinite computing resources, which can be provisioned in an on-demand fashion under the pay-per-use business model [5]. Infrastructure as a Service (IaaS) clouds currently offer

[1] http://www.youtube.com/

computing resources, such as Virtual Machines (VMs), storage, and network bandwidth [32], which can be used to create a dynamically scalable cluster of video transcoding servers.

In a cloud environment, a video transcoding operation can be performed in several different ways. For example, it is possible to map an entire video stream on a dedicated VM. However, it requires a large number of VMs to transcode several simultaneous streams. Moreover, transcoding of High Definition (HD) video streams can take more time, which may violate the client-side QoS requirements of desired play rate [10]. Another approach is to split the video streams into smaller segments and then transcode them independently of one another [21]. In this approach, one VM can be used to transcode a large number of video segments belonging to different video streams. Moreover, video segments of one particular stream can be transcoded on multiple VMs.

In this chapter, we present prediction-based dynamic resource allocation and deallocation algorithms to scale video transcoding service on a given IaaS cloud in a horizontal fashion. The proposed algorithms allocate and deallocate VMs to a dynamically scalable cluster of video transcoding servers. We use a two-step load prediction method [2], which predicts a few steps ahead in the future to allow proactive resource allocation. For cost-efficiency, we share VM resources among multiple video streams. The sharing of the VM resources is based on the video segmentation, which splits the streams into smaller segments that can be transcoded independently of one another. The proposed approach is evaluated in two simulation-based experiments involving two different load patterns. The results show that it provides cost-efficient resource allocation for a large number of simultaneous streams while avoiding over and underflow of the output video buffer.

We also present the implementation of a distributed transcoder based on the message passing programming model on top of an Amazon Elastic Compute Cloud (EC2)[2] cluster. Among different methods of distributed computing we have chosen to use Message Passing Interface (MPI)[3] because of its maturity, support, open source nature, scalability, and ease of use. MPI is a message passing interface for Multiple Instruction Multiple Data (MIMD)[4] distributed memory concurrent computers and workstations [27]. In this programming model, a set of tasks that use their own local memory during computation can be performed on the same physical machine as well as across an arbitrary number of machines. Tasks exchange data through

[2]http://aws.amazon.com/ec2/
[3]http://www.mcs.anl.gov/research/projects/mpi/
[4]http://www.springerreference.com/docs/html/chapterdbid/311449.html

communication channels by sending and receiving messages. This means that data transfer usually requires cooperative operations to be performed by each process. MPI provides a library consisting of a set of basic functions that can be used to write parallel and distributed programs. The programmer is thus responsible and free to express all parallelism involved [16], [27]. To deploy the distributed video transcoding framework in a cloud, we selected StarCluster[5], which is used for cluster management. It is an open source cluster computing tool-kit for Amazon EC2. The main purpose of StarCluster is to automate overall process of building a cluster of VMs in Amazon EC2, which can be used for parallel and distributed computing.

We proceed as follows. In Section 4.2, we describe video bit stream structure. Section 4.3 presents the system architecture of the proposed VM allocation approach. Video segmentation is described in Section 4.4. Section 4.5 presents the proposed proactive VM allocation algorithms. Our load prediction approach is detailed in Section 4.6. In Section 4.7, we introduce our prototype implementation of MPI-based distributed video transcoder and present our dynamic load balancing algorithm for video transcoding in cloud computing. Section 4.8 presents simulation results of the proposed VM allocation approach, while Section 4.9 presents results of the prototype implementation of our MPI-based distributed video transcoder and our dynamic load balancing algorithm. Section 4.10 discusses important related works and Section 4.11 presents conclusion.

4.2 Video Bit Stream Structure

A Video stream is made up of different types of compressed frames. The video frames are organized into logical groups known as Group of Picturess (GOPs) and sequences. As shown in Figure 4.1, a video sequence comprises a sequence header and one or more GOPs. A GOP consists of different types of frames such as I (intra), P (predicted), and B (bi-directional predicted) containing all necessary information required to decode them.

Both I and P frames can be used as reference frames. However, an I frame is an independent reference frame that does not require any other reference frame in the decoding process. I-Frames have the highest quality, but have the largest size. A GOP starts with an I frame, which is followed by a number of P and B frames. Both P and B frames always require reference frames in the decoding process [36]. The P-frames use information from the previous I-Frames or P-frames to compress the frame. The B-frames

[5]http://star.mit.edu/cluster/

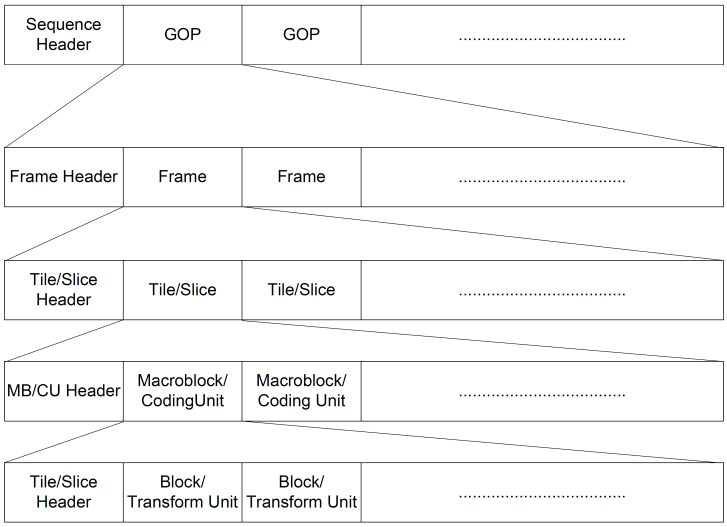


Figure 4.1: Structure of a video down to block level [36]

use information from both the previous and next I-Frame or P-Frames. B-Frames are the smallest in size, but have the least quality.

Frame prediction is used to reduce the size of frames by taking into account the previous and future frames to further improve the efficiency of compressed frames. Furthermore, a frame itself may be divided into smaller blocks or slices with each slice being able to use frame prediction independently of other slices.

A GOP in MPEG-4 can have from 0 to 3 B frames between successive P frames. Usually, it is 2. The distance between successive I frames is N, which includes both P and B frames. In many cases, the value of N is 12, but it can be any value between 1 and a few hundreds.

Different kinds of frames also require a different amount of memory. Typically, I frames require the largest number of bytes to represent images, for example 300 Kilobytes (KB). The P frames require less memory, for example 160 KB. The B frames require even less, for example 40 KB. The resolution of a video stream is measured in pixels and is usually written as horizontal x vertical, such as (1280 x 720). The frame-rate of a video is the number of frames displayed in a second, usually 24 to 30 frames per seconds (fps).

Video coding standards use a YCbCr color space [12] with three different components called *luma* (Y) and *chroma* (Cb, Cr). The *luma* component represents brightness while *chroma* components represents the color information.

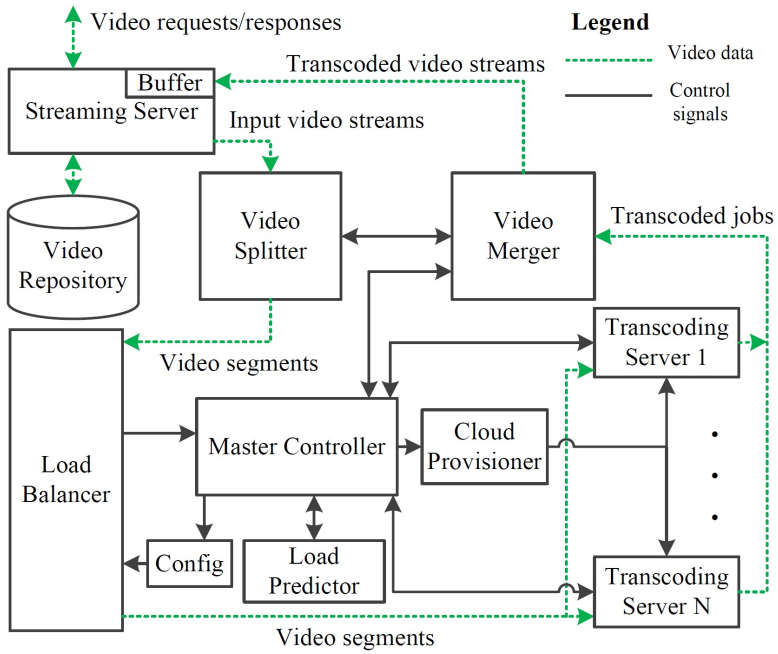


Figure 4.2: System architecture of the proactive VM allocation approach

A picture is usually divided into smaller parts termed as *macroblocks* or *coding units*. The *macroblocks* or *coding units* are the basic building blocks of video standards and the decoding of frames is performed at this level. To compress a frame, different techniques are used at *macroblock* or *coding unit* level such as, frame prediction in either spatial or temporal direction. The smaller partitions of the *macroblocks* are termed as *blocks* or *transform units*. The *transform coding* is performed at block level in various video coding standards.

4.3 System Architecture

The system architecture of the distributed video transcoding in cloud computing environment is shown in Figure 4.2. It consists of a *streaming server*, a *video splitter*, a *video merger*, a *video repository*, a dynamically scalable cluster of *transcoding servers*, a *load balancer*, a *master controller*, a *load predictor*, and a *cloud provisioner*.

In our system, the end-users or clients may send requests for videos. These

requests arrive at the *streaming server*. The *streaming server* provides the video streaming service. Streaming is a general term which means that the data being transfered from one location to another can be used immediately. In case of video streaming, a video is decoded and played as soon as enough data has been transferred[6].

To transfer multimedia contents (video and audio) over the Internet by using streaming services such as Video Desk[7], media streaming technology is used, which can deliver the media contents in real-time. At the user end, parts of a video are downloaded, decoded, and played. The video playing and downloading happen at the same time. A buffer is used to place additional video contents from the streaming server. The overall process is invisible to the viewer. The streaming server works as a media servers which sends streamed video to users connected through different networks. Since the main focus of this research work is on video transcoding, we assume that the *streaming server* is not a bottleneck.

The video streams in certain compressed formats are stored in the *video repository*. The compressed videos can be either source videos or transcoded videos. The source videos are the original videos and transcoded videos are obtained from source videos after transcoding. The transcoded videos are stored as long as it is cost-efficient to store them. The *streaming server* accepts video requests from users and checks if the required video is available in the *video repository*. If it finds the video in the desired format and resolution, it starts streaming the video. However, if it finds that the requested video is stored only in another format or resolution than the one desired by the user, it sends the video for segmentation and subsequent transcoding. Then, as soon as it receives the transcoded video from the *video merger*, it starts streaming the video.

The *video splitter* splits the video streams into smaller segments called jobs, which are placed into the job queue. Due to inter-dependency among different types of frames, video segmentation (splitting) can be performed at certain points only. When splitting the video, the main issue is to perform the segmentation of source video so that parts of video can be distributed among transcoding servers. Section 4.4 discusses video segmentation in more detail.

The *load balancer* employs a task assignment policy, which distributes load on the transcoding servers. In other words, it decides when and to which *transcoding server* a transcoding job should be routed. It maintains a configuration file, which contains information about *transcoding servers*

[6]http://www.wimpyplayer.com/docs/faqs/docs/general_streaming_definition.html
[7]http://www.videodesk.net/

that perform the transcoding operations. As a result of dynamic resource allocation and deallocation operations, the configuration file is often updated with new information. The *load balancer* serves the jobs in First In, First Out (FIFO) order and has only one input queue. The *load balancer* implements the *shortest queue waiting time* policy, which selects a *transcoding server* with the shortest queue waiting time. Our dynamic load balancing algorithm is presented in Section 4.7.

The actual transcoding is performed by the transcoding servers. They get compressed video segments, perform the required transcoding operations, and return the transcoded video segments for merging. A *transcoding server* runs on a dynamically provisioned VM. Each *transcoding server* processes one or more simultaneous jobs. When a transcoding job arrives at a *transcoding server*, it is placed into the server's queue from where it is subsequently processed.

The *master controller* implements prediction-based dynamic resource allocation and deallocation algorithms, as described in section 4.5. For load prediction, the *master controller* uses *load predictor*, which is elaborated in section 4.6. The *cloud provisioner* refers to the cloud provisioner in an IaaS cloud, such as the provisioner in Amazon EC2. It performs the actual lower level tasks of starting and terminating VMs. The *video merger* merges the transcoded jobs into video streams, which form video responses.

The system architecture is similar to our previous work on prediction-based dynamic resource allocation for video transcoding in cloud computing [22]. However, in [22] the load balancer implements the *shortest queue length* policy, while in this chapter it implements the *shortest queue waiting time* policy, which provides improved performance. The *shortest queue length* policy is based on the queue size alone. Therefore, it does not account for the execution time of individual jobs in the queue. Whereas, the *shortest queue waiting time* policy uses estimated execution time of individual jobs in the queue to calculate the waiting time of new arriving jobs at the server queue. In addition to the *shortest queue waiting time* policy, we introduce some important enhancements to our VM allocation algorithms.

4.4 Video Segmentation

Distributed computing allows to speedup the transcoding process while maintaining the same quality of video. Due to inter-dependency among different types of frames, video segmentation can be performed at certain points only. The main problem is to perform the segmentation of source video in such a way that parts of the video can be distributed among transcoding servers.

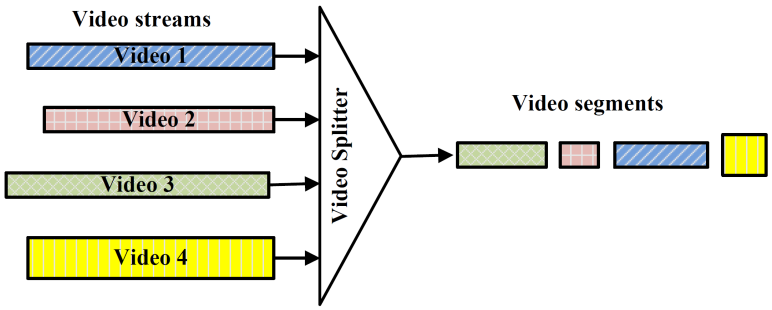


Figure 4.3: Video segmentation

Compressed video files contain different types of frames (I, P, B) which have different compression rates and inter-dependencies among them. Therefore, one can not split a given video at any particular frame or point. Among the frame types, an I-frame is an independent frame that can be decoded without any other reference frame. It is also used as a reference frame for other frames. In a given video, a sequence of frames that constitute an I-frame and a number of other B and P frames is called a GOP. GOPs are atomic units that can be transcoded independently of one another. Therefore, for efficient use of computing resources, we use video segmentation at the GOP level.

Video segmentation of four video streams is shown in Figure 4.3. The output of the video splitter consists of a number of jobs, where each job has at least one GOP. The video splitter tries to manage segmentation in such a way that each user gets a smooth video stream from the *streaming server*. It takes into account the transcoding time and the play time of the video segment. Once a video segment is sent for transcoding, the next segment of the same stream is sent after some delay.

The delay between two jobs during segmentation is based on the play time of a video segment and the number of transcoded video frames of the stream in the output buffer. If the transcoded frames are below certain predefined lower threshhold, the stream segmentation is performed with zero delay. However, if the transcoded frames are above the threshold, the delay for next video segment is set equal to the play time of previous job.

4.5 Proactive VM Allocation Algorithms

Under-utilization of computing servers in cloud computing is a common problem. Due to change in the load patterns at different times, there might be

some over-provisioning of servers, which increases the overall cost [6], [31]. In addition, the over-utilization of resources is also not desirable due to high response times. Therefore, automatic VM allocation is essential for cost and performance efficiency [40]. In this section, the dynamic VM allocation and deallocation algorithms for video transcoding in the cloud are presented. For the sake of clarity, the concepts used in the algorithms and their notation are summarized in Table 4.1.

The algorithms implement proactive control, which uses a two-step prediction in which the load pattern of the system is tracked and then future load pattern is predicted. The *transcoding servers* are added and removed based on the predicted load and current throughput. Moreover, due to the VM provisioning delay, a fixed minimum number of transcoding servers is always maintained to provide an effective service. This is termed as the base capacity N_B.

On discrete-time intervals, the *master controller* obtains the play rate of all video streams, and sums up the play rates of streams, to get the total target play rate $PR(t_i)$. It then obtains the video transcoding rate from each *transcoding server* and calculates the total transcoding rate $TR(t_i)$. Moreover, for proactive VM allocation, it uses *load predictor* to predict the total transcoding rate of all *transcoding servers* $\hat{TR}(t_i)$ a few steps ahead in the future.

The algorithms are designed to be cost-efficient while minimizing potential oscillations in the number of VMs [38]. This is desirable because, in practice, provisioning of a VM takes a few minutes [8]. Therefore, oscillations in the number of VMs may lead to deteriorated performance. Moreover, since some contemporary IaaS providers, such as Amazon EC2, charge on hourly basis, oscillations will result in a higher provisioning cost. Therefore, the algorithms counteract oscillations by delaying new VM allocation operations until previous VM allocation operations have been realized [20]. Furthermore, for cost-efficiency, the deallocation algorithm terminates only those VMs whose renting period approaches its completion.

4.5.1 VM Allocation Algorithm

The VM allocation algorithm is given as Algorithm 4.1. The first two steps deal with the calculation of the target play rate $PR(t_i)$ of all streams and the total transcoding rate $TR(t_i)$ of all *transcoding servers*. The algorithm then obtains the predicted total transcoding rate $\hat{TR}(t_i)$ from the *load predictor*. Moreover, to avoid underflow of the output video buffer that temporarily stores transcoded jobs at the server-side, it considers the size of the output video buffer $B_S(t_i)$. If the target play rate exceeds the predicted transcoding

Table 4.1: Summary of concepts and their notation

$count_{over}(t_i)$	over allocation count at discrete-time t_i
$S(t_i)$	set of *transcoding servers* at t_i
$S_p(t_i)$	set of newly provisioned servers at t_i
$S_c(t_i)$	servers close to completion of renting period at t_i
$S_t(t_i)$	servers selected for termination at t_i
$PR(t_i)$	sum of target play rates of all streams at time t_i
$TR(t_i)$	total transcoding rate of all servers at time t_i
$\hat{TR}(t_i)$	predicted total transcoding rate at time t_i
$RT(s, t_i)$	remaining time of server s at t_i
$V(t_i)$	set of video streams at t_i
$N_P(t_i)$	number of servers to provision at t_i
$N_{P_Q}(t_i)$	number of servers to provision at t_i based on queue length
$N_T(t_i)$	number of servers to terminate at t_i
$getPR()$	get $PR(t_i)$ from video merger
$getTR(s)$	get transcoding rate of server s
$get\hat{TR}()$	get $\hat{TR}(t_i)$ from load predictor
$calN_P()$	calculate the value of $N_P(t_i)$
$calQN_P()$	calculate the value of $N_{P_Q}(t_i)$ based on queue length
$calN_T()$	calculate the value of $N_T(t_i)$
$calRT(s, t_i)$	calculate the value of $RT(s, t_i)$
$delay()$	delay function
$provision(n)$	provision n servers
$select(n)$	select n servers for termination
$sort(S)$	sort servers S on remaining time
$terminate(S)$	terminate servers S
C_T	over allocation count threshold
RT_U	remaining time upper threshold
RT_L	remaining time lower threshold
$MAXQL_{UT}$	Maximum Queue length upper threshold
B_L	buffer size lower threshold in megabytes
$B_S(t_i)$	size of the output video buffer in megabytes
B_U	buffer size upper threshold in megabytes
N_B	number of servers to use as base capacity
$startUp$	server startup delay
$avgQJobs$	average number of jobs in a server Queue
$jobCompletion$	job completion delay

rate while the buffer size $B_S(t_i)$ falls below its lower threshold B_L, the algorithm chooses to allocate resources by provisioning one or more VMs. The number of VMs to provision $N_P(t_i)$ is calculated as follows

$$N_P(t_i) = \left\lceil \frac{PR(t_i) - \hat{TR}(t_i)}{\frac{TR(t_i)}{|S(t_i)|}} \right\rceil \tag{4.1}$$

The algorithm then provisions $N_P(t_i)$ VMs, which are added to the cluster of *transcoding servers*. To minimize potential oscillations due to unnecessary VM allocations, the algorithm adds a delay for the VM startup time. Furthermore, it ensures that the total number of VMs $|S(t_i)|$ does not exceed the total number of video streams $|V(t_i)|$. The algorithm adjusts the number of VMs to provision $N_P(t_i)$ if $|S(t_i)| + N_P(t_i)$ exceeds $|V(t_i)|$. This is desirable because the transcoding rate of a video on a single VM is usually higher than the required play rate.

The VM allocation algorithm also takes into account the current load on servers. It checks the queue lengths of servers and if the average number of jobs in the queues is above a predefined maximum upper threshold, it provisions one or more servers. The number of VMs to provision $N_{P_Q}(t_i)$ is calculated as follows

$$N_{P_Q}(t_i) = \left\lceil \frac{avgQJobs}{MAXQL_{UT}} \right\rceil \tag{4.2}$$

4.5.2 VM Deallocation Algorithm

The VM deallocation algorithm is presented in Algorithm 4.2. The main objective of the algorithm is to minimize the VM provisioning cost, which is a function of the number of VMs and time. Thus, it terminates any redundant VMs as soon as possible. Moreover, to avoid overflow of the output video buffer, it considers the size of the output video buffer $B_S(t_i)$. After obtaining the target play rate $PR(t_i)$ and the predicted total transcoding rate $\hat{TR}(t_i)$, the algorithm makes a comparison. If $\hat{TR}(t_i)$ exceeds $PR(t_i)$ while the buffer size $B_S(t_i)$ exceeds its upper threshold B_U, it may choose to deallocate resources by terminating one or more VMs. However, to minimize unnecessary oscillations, it deallocates resources only when the buffer overflow situation persists for a predetermined minimum amount of time.

In the next step, the algorithm calculates the remaining time of each *transcoding server* $RT(s, t_i)$ with respect to the completion of the renting period. It then checks if there are any *transcoding servers* whose remaining time is less than the predetermined upper threshold of remaining time RT_U and more than the lower threshold of remaining time RT_L. The objective is to terminate only those servers whose renting period is close to completion, while excluding any servers that are extremely close to the completion of their renting period and therefore it is not cost-efficient to terminate them before the start of the next renting period. If the algorithm finds at least one

Algorithm 4.1. VM allocation algorithm

```
1: while true do
2:    N_P(t_i) := 0, N_{P_Q}(t_i) := 0
3:    PR(t_i) := getPR()
4:    TR(t_i) := 0
5:    for sϵS(t_i) do
6:       TR(t_i) := TR(t_i) + getTR(s)
7:    end for
8:    T̂R(t_i) := getT̂R(TR(t_i))
9:    if T̂R(t_i) < PR(t_i) ∧ B_S(t_i) < B_L then
10:      N_P(t_i) := calN_P()
11:   end if
12:   if avgQJobs > MAXQL_UT then
13:      N_{P_Q}(t_i) := calQN_P()
14:   end if
15:   N_P(t_i) := N_P(t_i) + N_{P_Q}(t_i)
16:   if |S(t_i)| + N_P(t_i) > |V(t_i)| then
17:      N_P(t_i) := |V(t_i)| − |S(t_i)|
18:   end if
19:   if N_P(t_i) ≥ 1 then
20:      S_p(t_i) := provision(N_P(t_i))
21:      S(t_i) := S(t_i) ∪ S_p(t_i)
22:      delay(startUp)
23:   end if
24: end while
```

such server $S_c(t_i)$, it calculates the number of servers to terminate $N_T(t_i)$ as

$$N_T(t_i) = \left\lceil \frac{\hat{TR}(t_i) - PR(t_i)}{\frac{TR(t_i)}{|S(t_i)|}} \right\rceil - N_B \tag{4.3}$$

Then, it sorts the *transcoding servers* in $S_c(t_i)$ on the basis of their remaining time, and selects the servers with the lowest remaining time for termination. The rationale of sorting of servers is to ensure cost-efficiency by selecting the servers closer to completion of their renting period. A VM that has been selected for termination might have some pending jobs in its queue. Therefore, it is necessary to ensure that the termination of a VM does not abandon any jobs in its queue. One way to do this is to migrate all pending jobs to other VMs and then terminate the VM [8]. However, since transcoding of video segments takes relatively less time to complete, it is more reasonable

Algorithm 4.2. VM deallocation algorithm

```
1: while true do
2:   PR(t_i) := getPR()
3:   TR(t_i) := 0
4:   for sϵS(t_i) do
5:     TR(t_i) := TR(t_i) + getTR(s)
6:   end for
7:   T̂R(t_i) := getT̂R(TR(t_i))
8:   if T̂R(t_i) > PR(t_i) ∧ B_S(t_i) > B_U ∧ count_over(t_i) > C_T then
9:     for sϵS(t_i) do
10:      RT(s, t_i) := calRT(s, t_i)
11:    end for
12:    S_c(t_i) := {∀sϵS(t_i)|RT(s, t_i) < RT_U ∧ RT(s, t_i) > RT_L}
13:    if |S_c(t_i)| ≥ 1 then
14:      N_T(t_i) := calN_T()
15:      N_T(t_i) := min(N_T(t_i), |S_c(t_i)|)
16:      if N_T(t_i) ≥ 1 then
17:        sort(S_c(t_i))
18:        S_t(t_i) := select(N_T(t_i))
19:        S(t_i) := S(t_i) \ S_t(t_i)
20:        delay(jobCompletion)
21:        terminate(S_t(t_i))
22:      end if
23:    end if
24:  end if
25: end while
```

to let the jobs complete their execution without requiring them to migrate and then terminate a VM when there are no more running and pending jobs on it. Therefore, the deallocation algorithm terminates a VM only when the VM renting period approaches its completion and all jobs on the server complete their execution. Finally, the selected servers are terminated and removed from the cluster.

4.6 Load Prediction

The existing load prediction models for web-based systems, such as [2], [3], [34], can be adapted to predict transcoding rate of the *transcoding servers* a few steps ahead in the future. Andreolini and Casolari [2] proposed a two-step approach to predict future load behavior under real-time con-

straints. The approach involves load trackers that provide a representative view of the load behavior to the load predictors, thus achieving two steps.

A load tracker (LT) filters out noise in the raw data to yield a more regular view of the load behavior [2]. It is a function $LT(\overrightarrow{S_n}(t_i)) : \mathbb{R}^n \rightarrow \mathbb{R}$, which inputs a measure s_i monitored at time t_i, and a set of previously collected n measures, that is $\overrightarrow{S_n}(t_i) = (s_{i-n}, ..., s_i)$, and provides a representation of the load behavior l_i at time t_i [2]. A sequence of LT values yields a regular view of the load behavior. There are different classes of LTs, such as simple moving average (SMA), exponential moving average (EMA), and cubic spline (CS) [3]. More sophisticated (time-series) models often require training periods to compute the parameters and/or off-line analyses [2]. Likewise, the linear (auto) regressive models, such as ARMA and ARIMA, may require frequent updates to their parameters [2], [34]. Therefore, in our approach [7], [22], the *load predictor* implements an LT based on the EMA model, which limits the computation delay without incurring oscillations and computes an LT value for each measure with high prediction accuracy.

The load predictor (LP) is a function $LP_h(\overrightarrow{L_q}(t_i)) : \mathbb{R}^q \rightarrow \mathbb{R}$, which inputs a sequence of LT values $\overrightarrow{L_q}(t_i) = l_{i-q}, ..., l_i$ and outputs a predicted future value at time t_{i+h}, where $h > 0$ [2]. The LP is characterized by the prediction window h and the past time window q. Andreolini and Casolari [2] and Saripalli et al. [34] used linear regression of only two LT values, which are the first l_{i-q} and the last l_i values in the past time window. Ashraf et al. [7] and Jokhio et al. [22] used simple linear regression model [28], which takes into account all LT values $\overrightarrow{L_q}(t_i)$ in the past time window. The LP of the LT in this approach is based on a straight line defined as

$$l = \gamma_0 + \gamma_1 t \tag{4.4}$$

where γ_0 and γ_1 are called regression coefficients, which can be estimated at runtime based on the LT values [7], [28].

4.7 MPI-Based Distributed Video Transcoder

Our distributed video transcoder is based on the open source FFMPEG[8] library. We have modified the original transcoder to execute on multiple machines. The extension is based on MPI, where a set of processes each running a single transcoder, are made to collaborate and transcode a set of streams more efficiently. The MPI programming model allows programmers to explicitly specify the parallel processing in a given system. Unlike other

[8]http://www.ffmpeg.org/

frameworks like Hadoop, which is designed for a specific set of problems(i.e. batch processing), MPI provides programmers with flexibility in expense of some programming overhead.

Figure 4.4 shows the architecture of the MPI-based video transcoder for multiple streams. Each active incoming stream is segmented continuously and stored in a job queue of one of the transcoding servers. Load balancing is performed dynamically depending on an estimated queue waiting time of a segment on each transcoding server. The estimated waiting time of a new segment on a given transcoding server is calculated by dividing the number of frames that are currently in its queue with the average transcoding rate of the server (see Algorithm 4.3). The *manager* sends the next segment to a transcoding server with the smallest estimated queue waiting time. A header containing the stream ID, segment ID, a transcoding parameter, and number of frames in the segment is attached to each segment. This header is used to identify each segment in the system and make load balancing decisions. The total number of transcoding servers is decided by the *manager*. In our distributed transcoder, every server has its own ID and the work is routed according to these IDs. In Figure 4.4, the ID of the *manager* is 0. It implements the proposed dynamic load balancing algorithm. Moreover, it contains the *video splitter* and the *video merger*. The manager invokes *video splitter* before sending transcoding jobs to the transcoding servers. Each transcoding server takes a segment from its queue, parses the segment header to get the transcoding parameters, transcodes the segment, and sends back the transcoded segment to the *manager*. The *manager* then invokes the *video merger* to merge the transcoded segments into output streams. The impact of the dynamic load balancing algorithm is compared with a static round-robin approach in Section 4.9.

4.8 Simulation Results of VM Allocation

Software simulations are often used to test and evaluate new algorithms involving complex environments [11]. We have developed a discrete-event simulation for the proposed VM allocation approach. The simulation is written in the Python programming language and is based on the SimPy simulation framework [26].

4.8.1 Experimental Design and Setup

We considered two different synthetic load patterns in two separate experiments. Load pattern 1 in experiment 1 consists of two load peaks, while

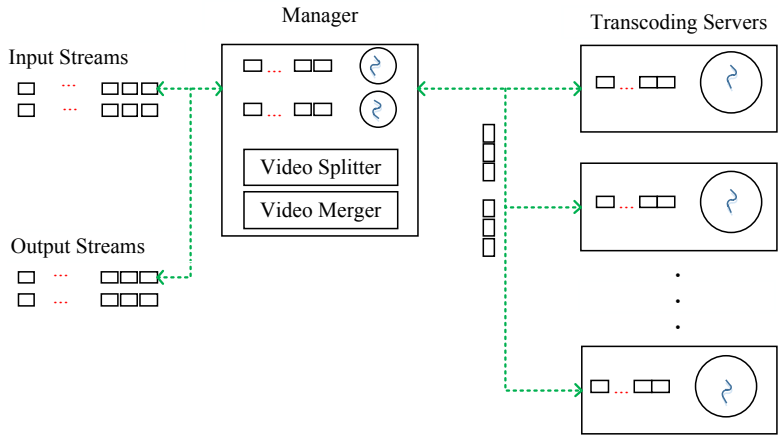


Figure 4.4: MPI-based distributed video transcoder

Algorithm 4.3. Dynamic load balancing

```
1: while true do
2:    selected_server = transcoding_servers.get(0)
3:    smallest_queue_time = TIME_MAX
4:    for transcoding_server in transcoding_servers do
5:       queue_time = (transcoding_server->frames) / (transcoding_server->fps)
6:       if queue_time < smallest_queue_time then
7:          smallest_queue_time = queue_time
8:          selected_server = transcoding_server
9:       end if
10:   end for
11:   segment = queue->take()
12:   send(segment, selected_server)
13: end while
```

load pattern 2 in experiment 2 has six load peaks. For simplicity, the renting period was assumed to be 600 seconds. The remaining time upper threshold RT_U was 60 seconds, while the remaining time lower threshold RT_L was 12 seconds. The Load Tracker (LT) and lp parameters were as follows: $n = 15$, $q = 30$, and $h = 120$.

The experiments used both Standard Definition (SD) and HD video streams. At the time of writing this book chapter, 10% of YouTube's videos are available in HD, while YouTube has more HD content than any other

video hosting site [1]. However, the ratio of HD versus SD is expected to increase in the near future. Therefore, the load generation assumed 70% SD and 30% HD video streams. The video segmentation was performed at the GOP level. The segmentation produced video segments, which were sent to the *transcoding servers* for execution. For HD videos, the average size of a video segment was 75 frames with a standard deviation of 7 frames. Likewise, for SD videos, the average size of a segment was 250 frames with a standard deviation of 20 frames. The total number of frames in a video stream was in the range of 15000 to 18000.

The desired play rate for a video stream is often fixed: 30 fps for SD videos and 24 fps for HD videos. Whereas, the transcoding rate depends on the video contents, such as, frame resolution, type of video format, type of frames, and contents of blocks. Different transcoding mechanisms also require different times.

In our experiments, the maximum transcoding rate for SD videos was assumed to be four times of its play rate. We further assumed that the transcoding rate is always higher than the play rate of all video streams. Similarly, the minimum transcoding rate for SD videos was assumed to be double of its play rate. Since HD videos require more computation, the maximum transcoding rate for an HD video was assumed to be double of the play rate, with the minimum transcoding rate at 1.5 times the play rate.

The objective of experiment 1 was to simulate a relatively normal load. It was designed to generate a load representing a maximum of 200 simultaneous video streams in two different load peaks. In the first peak, the streams were ramped-up from 0 to 200, while adding a new stream every 20 seconds. After the ramp-up phase, the number of streams was maintained constant for 1 hour and then ramped-down to 100 streams.

The second peak ramped-up from 100 streams to 200 streams, while adding a new stream every 30 seconds. The ramp-up phase was followed by a similar constant phase as in the first peak. Then, the ramp-down phase removed all streams from the system.

Experiment 2 was designed to simulate the load pattern of a highly variable video demand. It generated a load representing a maximum of 280 simultaneous video streams consisting of six different load peaks. In the first peak, the streams were ramped-up from 0 to 170, while adding a new stream every 30 seconds. Then, in the second peak from 110 to 250, while adding a new stream every 20 seconds. Likewise, 210 to 280, 215 to 250, 120 to 200, and 100 to 170, respectively, in the third, fourth, fifth, and sixth peak. The stream ramp-up rate was 1 new stream per 30 seconds. Each ramp-up phase was followed by a ramp-down phase. Finally, the last ramp-down phase removed all streams from the system.

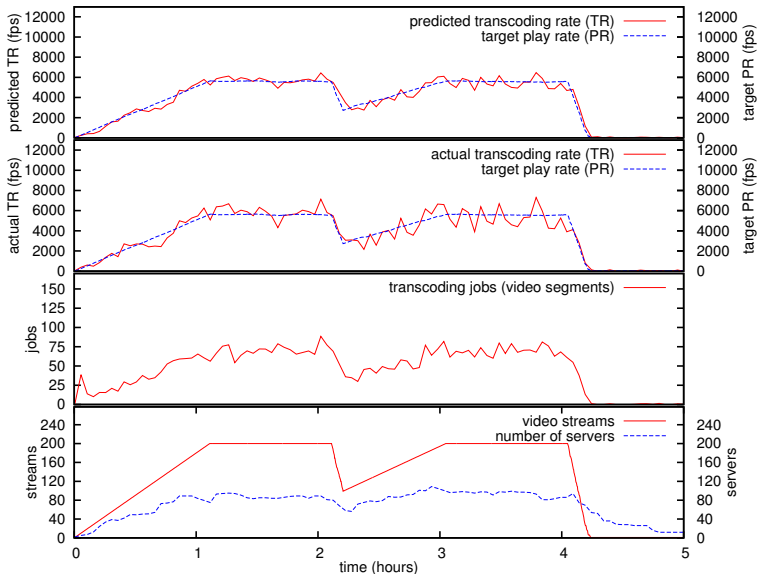


Figure 4.5: Experiment 1 results: relatively normal load

4.8.2 Results and Analysis

In both Figures 4.5 and 4.6, the number of servers plot shows dynamic VM allocation for the cluster of *transcoding servers*. The transcoding jobs plot represents the total number of jobs in the system at a particular time instance. It includes the jobs in execution at the *transcoding servers* and the jobs that were waiting in the queues. The target play rate plot shows the sum of target play rates of all video streams in the system. Likewise, the actual transcoding rate plot represents the total transcoding rate of all servers, while the predicted transcoding rate plot shows results of the load prediction. As described in Section 4.5, the VM allocation decisions were mainly based on the target play rate, the predicted transcoding rate, and the queue length of servers.

Figure 4.5 presents results from experiment 1. The results are also summarized in Table 4.2. Experiment 1 used a maximum of 93 *transcoding servers* for a maximum of 200 simultaneous streams. Moreover, a total of 4596 streams consisting of approximately 5×10^5 transcoding operations and 7×10^7 video frames were completed in 4 hours and 38 minutes. The results indicate that the resource allocation algorithms with the sharing of the VM resources among multiple video streams resulted in a reduced number

Table 4.2: Results from proactive VM allocation experiments. Experiment 1 uses relatively normal load, while experiment 2 uses highly variable load. The results include maximum number of servers used, maximum and average number of transcoding jobs or segments, maximum and average play rate (PR), and maximum and average transcoding rate (TR).

experiment	servers	$\text{jobs}_{\text{avg.}}$	jobs_{max}	$\text{PR}_{\text{avg.}}$	PR_{max}	$\text{TR}_{\text{avg.}}$	TR_{max}
1	109	44.84	95	3597.42 fps	5670 fps	3496.48 fps	7683.34 fps
2	150	40.74	122	3273.61 fps	7818 fps	3446.19 fps	9234.23 fps

of servers as compared with the number of video streams which reduces VM provisioning cost. The resource de-allocation algorithm takes into account the servers remaing renting time. A server is terminated only when it is near its completion of renting period, which avoids unnecessary oscillations in the number of VMs.

The results show that the actual transcoding rate was always close to the target play rate. This was desirable to avoid over and underflow of the output video buffer in the system, as discussed in Section 4.5. Although the actual transcoding rate was sometimes slightly above or below the target play rate, the proactive resource allocation helped to ensure that the cumulative number of transcoded frames was always greater than the cumulative number of played frames.

Figure 4.6 presents results from experiment 2. Table 4.2 also contains a summary of the results. It used a maximum of 120 *transcoding servers* for a maximum of 290 simultaneous streams. Moreover, a total of 7241 streams consisting of approximately 8×10^5 transcoding operations and 1×10^8 video frames were completed in 6 hours and 54 minutes. Although the number of streams was fluctuating rapidly, the algorithms provided a sustainable service with fewer VMs, while minimizing oscillations in the number of servers and avoiding the over and underflow of the output video buffer.

4.9 Evaluation of MPI-Based Distributed Video Transcoder

In this section, we describe the experimental setup, the results obtained, and their analysis from our prototype implementation of the MPI-based distributed video transcoder. The results are focused on the effect of load balancing algorithms on the utilization of transcoding servers. Therefore, provisioning of the transcoding servers is done statically before each experiment.

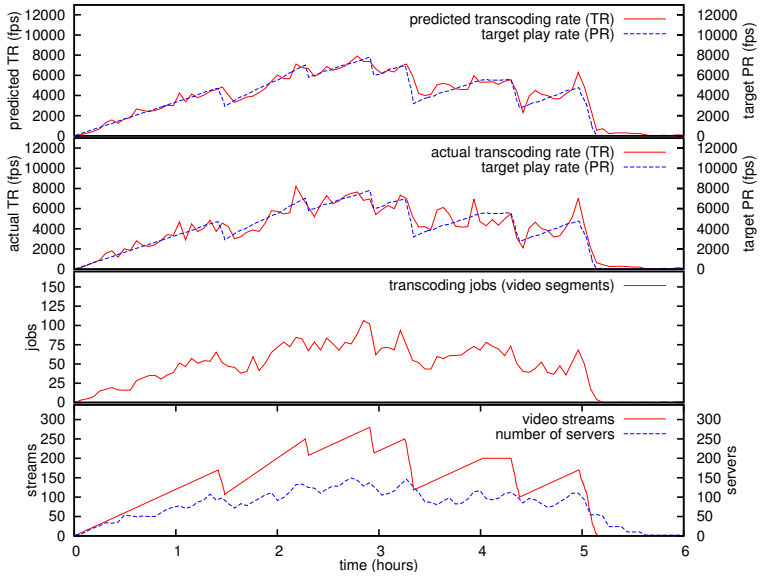


Figure 4.6: Experiment 2 results: highly variable load

4.9.1 Experimental Design and Setup

Our main task was to investigate performance in terms of total processing time of distributed video transcoding in cloud computing with dynamic load balancing. We setup a homogeneous and a heterogeneous test environment in the cloud using StarCluster. The StarCluster is an open source cluster computing tool-kit for Amazon EC2. It has been designed to automate and simplify the process of building, configuring, and managing clusters of VMs suited for distributed and parallel computing applications and systems on Amazon EC2.

The *homogeneous test cluster* consist of a *stream manager* and a maximum of 14 *transcoding servers*. The *stream manager* is assigned the task of splitting, merging, and scheduling of video segments. All the nodes in this cluster are *m1.small* instances from Amazon EC2. The *m1.small* instance is a VM with 1.7 GB memory and one virtual core running 32 bit Ubuntu 11.10 on AMD 2218HE. The *m1.small* instance is the default instance in the Amazon EC2 and is not optimized for anything in particular. Furthermore, the instances are selected from the same availability zone (i.e eu-west-1a). Running virtual cloud instances from the same availability zone ensures a more homogeneous network connection in the cluster.

The Heterogeneous test cluster consists of a maximum of 14 *transcoding servers* and a *stream manager*. Half nodes in the cluster are *m1.small* instances from Amazon EC2 cloud. The other half instances in the cluster are *c1.medium* instances. The *c1.medium* is a VM with $1.7GB$ memory and two virtual core running 32 bit Ubuntu 11.10 on Intel $E5-2650$ at 2 GHz. In contrast to the basic *m1.small* instance, the *c1.medium* instance has two cores and is optimized for speed. Furthermore, the instances are selected from two different availability zones (i.e eu-west-1a and eu-west-1c), which are located apart form each other. Running virtual cloud instances from different availability zone might lead to a more heterogeneous network connection in the cluster.

The aim of of doing the experiment on a heterogeneous clusters is motivated by the concept of job affinity [24], which states that some jobs may run significantly faster on nodes of a particular instance than others. Hence, it is important to know the job/instance type relationship and and design the load balancing algorithm accordingly.

To perform transcoding in the distributed environment, we have modified an existing open source transcoder FFMPEG with Message Passing Programming Model [27]. The first process with rank 0 is assigned the task of splitting, scheduling, and merging input video streams. The other processes with rank 1 to 14 are assigned the task of transcoding. The *manager* first splits incoming video streams at GOP level and uses the static or dynamic load balancing methods explained in Section 4.7. Depending on the scheduler decision, the *manager* sends video segments to a selected transcoding server's queue. The task of each transcoding server is to pick a task from their queue and perform the transcoding job till a termination signal is sent from the stream *manager*.

To perform different experiments in a cluster-based environment, we selected various video sequences having different number of frames. Characteristics of those video sequences such as length, size, total number of frames, and resolution are given in Table 4.3. All video sequences have a frame rate of 24 fps.

In this experiment we have restricted ourselves to using few streams as the focus is to only understand and compare the the effect of job load balancing approaches on the throughput of a distributed transcoding system.

4.9.2 Results and Analysis

We used both SD and HD video sequences in our experiments. Table 4.3 shows characteristics of video sequences, such as, size of video sequence in Mega Bytes, number of frames, and resolution. The total transcoding time

Table 4.3: Characteristics of video sequences

Video	Size	Frames	Resolution
Sintel HD	251 MB	21312	1280x720
Elephants Dream HD	162 MB	15690	1280x720
Big Buck Bunny HD	115 MB	14315	1280x720
Sintel SD	48 MB	21312	854x480
Elephants Dream SD	34 MB	15690	854x480
Big Buck Bunny SD	30 MB	14315	854x480

with different number of transcoding servers in a heterogeneous cloud cluster for Sintel HD, Elephants Dream HD, Big Buck Bunny HD, and for multiple streams with two HD and two SD simultaneous streams is shown in Figures 4.7a-d. Likewise, transcoding time with different number of servers in a homogeneous Cloud cluster for Sintel HD, Elephants Dreams HD, Big Buck Bunny HD, and for multiple streams with two HD and two SD simultaneous streams is shown in Figures 4.7e-h.

Figure 4.7a-d shows the total transcoding time for static round-robin and our proposed dynamic load balancing over a varying number of servers. In all cases, the performance gain accounts to about 45% except the case where there is only one transcoding node and the scheduling overhead matters significantly.

Figure 4.7e-h shows the result of applying the static and dynamic scheduling algorithms specified in Section 4.7 on a cluster of homogeneous transcoding servers. In this case, the performance gain from using the proposed dynamic load balancing algorithm only accounts to about $10 - 12\%$, except the cases when there is only one or two transcoding nodes resulting in a significant scheduling overhead. This result is also expected due to the fact that the experiment is done in a homogeneous platform and the performance gain is only due to the fact that video streams have different computational loads on different segments.

Figure 4.7d and Figure 4.7h show the results for the performance of the proposed dynamic scheduling algorithm against the static one when there are four simultaneous streams in the system. We selected the first two HD and two SD videos from Table 4.3 to test the multiple streams setup. As can be noticed from these figures, the gap between the performance of the dynamic and static schedulers slightly increased due to the non-homogeneity introduced by the Central Processing Unit (CPU) demand difference among HD and SD video streams.

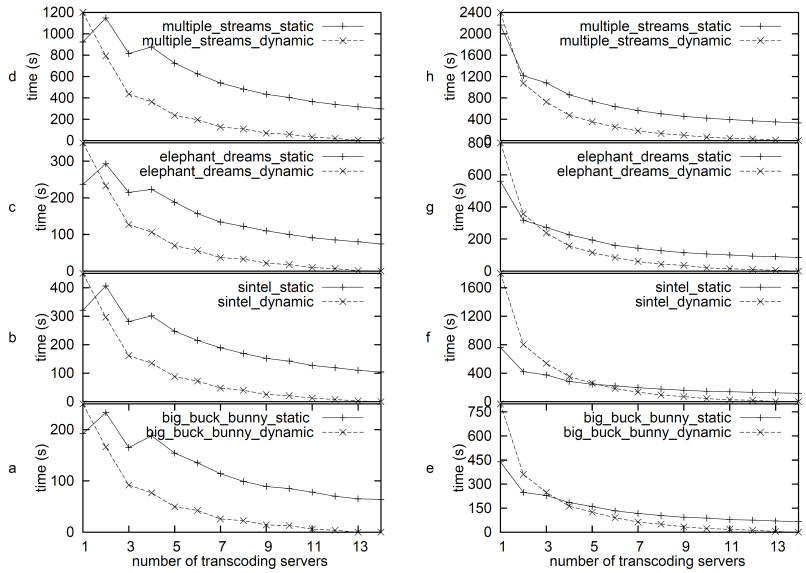


Figure 4.7: MPI-based distributed video transcoder results. Figures a-d show the total transcoding time with different number of transcoding servers in a heterogeneous cloud cluster for Sintel HD , Elephants Dream HD, Big Buck Bunny HD, and for multiple streams. Similarly, Figures e-h show the results in a homogeneous cloud cluster for Sintel HD, Elephants Dream HD, Big Buck Bunny HD, and for multiple streams.

4.10 Related Work

Distributed video transcoding with video segmentation was proposed in [21] and [23]. In these works, video segmentation was performed at the GOPs level. Jokhio et al. [21] presented bit rate reduction video transcoding using multiple processing units. The paper discussed computation, parallelization, and data distribution among computing units. In [23], different video segmentation methods were analyzed to perform spatial resolution reduction video transcoding. The paper compared three possible methods of video segmentation. In both papers, video transcoding was not performed in the cloud and the VM allocation problem was not addressed. In contrast, the main focus of this work is on VM allocation and deallocation algorithms. Huang et al. [19] presented a cloud-based video proxy to deliver transcoded videos for streaming. The main contribution of their work is a multilevel transcoding

parallelization framework. They used Hallsh-based and Lateness-first mapping to optimize transcoding speed and to reduce transcoding jitters. The performance evaluation was done on a campus cloud testbed and the communication latency between cloud and video proxy was neglected. Li et al. [25] proposed *cloud transcoder*, which uses a compute cloud as an intermediate platform to provide transcoding service. Both papers do not discuss the VM allocation problem for video transcoding in cloud computing.

The existing works on dynamic VM allocation can be classified into two main categories: Plan-based approaches and control theoretic approaches. Plan-based approaches can be further classified into workload prediction approaches and performance dynamics model approaches. One example of the workload prediction approaches is Ardagna et al. [4], while TwoSpot [38], Hu et al. [18], Chieu et al. [14], Iqbal et al. [20] and Han et al. [17] use a performance dynamics model. Similarly, Dutreilh et al. [15], Pan et al. [29], Patikirikorala et al. [30], and Roy et al. [33] are control theoretic approaches. One common difference between all of these works and our proposed approach is that they are not designed specifically for video transcoding in cloud computing. In contrast, our proposed approach is based on the important VM allocation metrics for video transcoding service. Moreover, the proposed approach is cost-efficient as it uses fewer VMs for a large number of video streams and it counteracts possible oscillations in the number of VMs that may result in higher provisioning costs.

Ardagna et al. [4] proposed a distributed algorithm for managing Software as a Service (SaaS) cloud systems that addresses capacity allocation for multiple heterogeneous applications. The resource allocation algorithm takes into consideration a predicted future load for each application class and a predicted future performance of each VM, while determining possible Service-Level Agreement (SLA) violations for each application type. The main challenge in the prediction-based approaches is in making good prediction models that should provide high prediction accuracy under real-time constraints. For this, we use a two-step prediction approach, which limits the computation delay without incurring oscillations, while providing high prediction accuracy.

TwoSpot [38] aims to combine existing open source technologies to support web applications written in different programming languages. It supports hosting of multiple web applications, which are automatically scaled up and down in a horizontal fashion. However, the scaling down is decentralized, which may lead to severe random drops in performance. For example, when all controllers independently choose to scale down at the same time. Hu et al. [18] proposed a heuristic algorithm that determines the server allocation strategy and job scheduling discipline which results in the minimum

number of servers. They also presented an algorithm for determining the minimum number of required servers, based on the expected arrival rate, service rate, and SLA. Chieu et al. [14] presented an approach that scales servers for a particular web application based on the number of active user sessions. The main problem with this approach is in determining suitable threshold values on the number of user sessions. Iqbal et al. [20] proposed an approach for adaptive resource provisioning for read intensive multi-tier web applications. Based on response time and CPU utilization metrics, the approach determines the bottleneck tier and then scales it up by provisioning a new VM. Scaling down is supported by checking for any over-provisioned resources from time to time. Han et al. [17] proposed a reactive resource allocation approach to integrate VM-level scaling with a more fine-grained resource-level scaling. In contrast, the proposed approach provides proactive VM allocation, where the VM allocation decisions are based on the important video transcoding metrics, such as video play rate and server transcoding rate.

Dutreilh et al. [15] and Pan et al. [29] used control theoretic models for designing resource allocation solutions for cloud computing. Dutreilh et al. presented a comparison of static threshold-based and reinforcement learning techniques. Pan et al. used Proportional-Integral (PI) controllers to provide QoS guarantees. Patikirikorala et al. [30] proposed a multi-model framework for implementing self-managing control systems for QoS management. Roy et al. [33] presented a look-ahead resource allocation algorithm based on the model predictive control. A common characteristic of the control theretic approaches is that they depend upon performance and dynamics of the underlying system. In contrast, the proposed approach does not require any knowledge about the performance and dynamics of the *transcoding servers*.

4.11 Conclusion

In this chapter, we presented proactive VM allocation algorithms to scale video transcoding service in a cloud environment. The proposed algorithms provide a mechanism for creating a dynamically scalable cluster of video transcoding servers by provisioning VMs from an IaaS cloud. The prediction of the future user load is based on a two-step load prediction method, which allows proactive VM allocation with high prediction accuracy under real-time constraints. For cost-efficiency, we used segmentation of video streams, which splits a stream into smaller segments that can be transcoded independently of one another. This helped us to perform video transcoding of multiple streams on a single server. The proposed VM allocation approach is demonstrated

in a discrete-event simulation. The evaluation and analysis considered two different synthetic load patterns in two separate experiments. Experiment 1 used a relatively normal load, while experiment 2 used a highly variable load. The results show that the proposed approach provides cost-efficient VM allocation for transcoding a large number of video streams, while minimizing oscillations in the number of servers and avoiding over and underflow of the output video buffer.

We also presented a prototype implementation of a MPI-based distributed video transcoder and a dynamic load balancing algorithm for video transcoding in cloud computing. Experimental results from the MPI implementation show that the distributed transcoding approach along with the dynamic load balancing scheme decreases the total transcoding time up to 45% for a heterogeneous set of servers and up to 12% for homogeneous environments.

Future work includes implementing an admission controller to prevent transcoding servers from becoming overloaded. We have been currently working on a stream-based admission control approach for video transcoding in cloud computing [9]. Furthermore, a computation and storage trade-off strategy for video transcoding in cloud computing and using realistic load patterns for experimental evaluation are also part of our ongoing work.

Acknowledgments

This work was supported by an Amazon Web Services research grant. Fareed Jokhio and Adnan Ashraf were partially supported by the Foundation of Nokia Corporation and by doctoral scholarships from the Higher Education Commission (HEC) of Pakistan.

References

[1] *35 Mind Numbing YouTube Facts, Figures and Statistics Infographic.* 2012/05/23. URL: http://www.jeffbullas.com/2012/05/23/35-mind-numbing-youtube-facts-figures-and-statistics-infographic/.

[2] M. Andreolini and S. Casolari. "Load prediction models in web-based systems". In: *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools.* valuetools '06. New York, NY, USA: ACM, 2006. ISBN: 1-59593-504-5. DOI: 10.1145/1190095.1190129.

[3] M. Andreolini, S. Casolari, and M. Colajanni. "Models and Framework for Supporting Runtime Decisions in Web-Based Systems". In: *ACM Transactions on the Web* 2.3 (2008), pp. 1–43. ISSN: 1559-1131. DOI: 10.1145/1377488.1377491.

[4] D. Ardagna et al. "Service Provisioning on the Cloud: Distributed Algorithms for Joint Capacity Allocation and Admission Control". In: *Towards a Service-Based Internet*. Ed. by E. Di Nitto and R. Yahyapour. Vol. 6481. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 1–12.

[5] M. Armbrust et al. "A view of cloud computing". In: *Commun. ACM* 53.4 (Apr. 2010), pp. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672.

[6] M. Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. UCB/EECS-2009-28. EECS Department, University of California, Berkeley, 2009. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[7] A. Ashraf, B. Byholm, and I. Porres. "A Session-Based Adaptive Admission Control Approach for Virtualized Application Servers". In: *The 5th IEEE/ACM International Conference on Utility and Cloud Computing*. Ed. by C. Varela and M. Parashar. IEEE Computer Society, 2012, pp. 65–72.

[8] A. Ashraf et al. "Feedback Control Algorithms to Deploy and Scale Multiple Web Applications per Virtual Machine". In: *38th Euromicro Conference on Software Engineering and Advanced Applications*. Ed. by V. Cortellessa, H. Muccini, and O. Demirors. IEEE Computer Society, 2012, pp. 431–438.

[9] A. Ashraf et al. "Stream-Based Admission Control and Scheduling for Video Transcoding in Cloud Computing". In: *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 2013.

[10] N. Bjork and C. Christopoulos. "Transcoder architectures for video coding". In: *Consumer Electronics, IEEE Transactions on* 44.1 (1998), pp. 88 –98. ISSN: 0098-3063. DOI: 10.1109/30.663734.

[11] R. N. Calheiros et al. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". In: *Software: Practice and Experience* 41.1 (2011), pp. 23–50. ISSN: 1097-024X.

[12] D. Chai and A. Bouzerdoum. "A Bayesian approach to skin color classification in YCbCr color space". In: *TENCON 2000. Proceedings.* Vol. 2. 2000, 421–424 vol.2. DOI: 10.1109/TENCON.2000.888774.

[13] S. F. Chang and A. Vetro. "Video Adaptation: Concepts, Technologies, and Open Issues". In: *Proceedings of IEEE* 93.1 (Jan. 2005), pp. 148–158. URL: http://dx.doi.org/10.1109/JPROC.2004.839600.

[14] T. C. Chieu et al. "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment". In: *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on.* 2009, pp. 281–286. DOI: 10.1109/ICEBE.2009.45.

[15] X. Dutreilh et al. "From Data Center Resource Allocation to Control Theory and Back". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on.* 2010, pp. 410–417. DOI: 10.1109/CLOUD.2010.55.

[16] Gropp, W., Lusk, E., and Skjellum, A. *Using MPI, Portable Parallel Programming with the Message Passing Interface.* MIT Press.

[17] R. Han et al. "Lightweight Resource Scaling for Cloud Applications". In: *Cluster Computing and the Grid, IEEE International Symposium on* (2012), pp. 644–651.

[18] Y. Hu et al. "Resource provisioning for cloud computing". In: *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research.* CASCON '09. New York, NY, USA: ACM, 2009, pp. 101–111.

[19] Z. Huang et al. "CloudStream: Delivering high-quality streaming videos through a cloud-based SVC proxy". In: *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China.* IEEE, 2011, pp. 201–205. DOI: http://dx.doi.org/10.1109/INFCOM.2011.5935009.

[20] W. Iqbal et al. "Adaptive resource provisioning for read intensive multi-tier applications in the cloud". In: *Future Generation Computer Systems* 27.6 (2011), pp. 871–879. ISSN: 0167-739X.

[21] F. Jokhio et al. "Bit Rate Reduction Video Transcoding with Distributed Computing". In: *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on.* 2012, pp. 206 –212. DOI: 10.1109/PDP.2012.59.

[22] F. Jokhio et al. "Prediction-Based Dynamic Resource Allocation for Video Transcoding in Cloud Computing". In: *Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International Conference on*. 2013.

[23] F. A. Jokhio et al. "Analysis of video segmentation for spatial resolution reduction video transcoding". In: *Intelligent Signal Processing and Communications Systems (ISPACS), 2011 International Symposiuml*. 2011, 6 pp.

[24] G. Lee, B.-G. Chun, and H. Katz. "Heterogeneity-aware resource allocation and scheduling in the cloud". In: *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*. HotCloud'11. Portland, OR: USENIX Association, 2011, pp. 4–4. URL: http://dl.acm.org/citation.cfm?id=2170444.2170448.

[25] Z. Li et al. "Cloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices". In: *The 22nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2012. ISBN: 978-1-4503-0752-9/12/06.

[26] N. Matloff. *A Discrete-Event Simulation Course Based on the SimPy Language*. University of California at Davis, 2006. DOI: http://heather.cs.ucdavis.edu/~matloff/simcourse.html. URL: http://heather.cs.ucdavis.edu/~matloff/simcourse.html.

[27] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. Knoxville, TN: University of Tennessee, June 1995.

[28] D. Montgomery, E. Peck, and G. Vining. *Introduction to Linear Regression Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2012. ISBN: 9780470542811.

[29] W. Pan et al. "Feedback Control-Based QoS Guarantees in Web Application Servers". In: *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*. 2008, pp. 328–334. DOI: 10.1109/HPCC.2008.106.

[30] T. Patikirikorala et al. "A multi-model framework to implement self-managing control systems for QoS management". In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 218–227. ISBN: 978-1-4503-0575-4.

[31] M. Pawlish, A. S. Varde, and S. A. Robila. "Cloud computing for environment-friendly data centers". In: *Proceedings of the fourth international workshop on Cloud data management*. CloudDB '12. Maui, Hawaii, USA: ACM, 2012, pp. 43–48. ISBN: 978-1-4503-1708-5. DOI: 10.1145/2390021.2390030. URL: http://doi.acm.org/10.1145/2390021.2390030.

[32] J. Rhoton and R. Haukioja. *Cloud Computing Architected: Solution Design Handbook*. Recursive Press, 2011. ISBN: 9780956355614.

[33] N. Roy, A. Dubey, and A. Gokhale. "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting". In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 2011, pp. 500 –507. DOI: 10.1109/CLOUD.2011.42.

[34] P. Saripalli et al. "Load Prediction and Hot Spot Detection Models for Autonomic Cloud Computing". In: *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. 2011, pp. 397 –402. DOI: 10.1109/UCC.2011.66.

[35] A. Vetro, C. Christopoulos, and H. Sun. "Video transcoding architectures and techniques: an overview". In: *Signal Processing Magazine, IEEE* 20.2 (2003), pp. 18 –29. ISSN: 1053-5888. DOI: 10.1109/MSP.2003.1184336.

[36] J. Watkinson. *The MPEG Handbook: MPEG-1, MPEG-2, MPEG-4*. Broadcasting and communications. Elsevier/Focal Press, 2004. ISBN: 9780240805788.

[37] T. Wiegand, G. J. Sullivan, and A. Luthra. "Draft ITU-T recommendation and final draft international standard of joint video specification". In: *Technical Report*. 2003.

[38] A. Wolke and G. Meixner. "TwoSpot: A Cloud Platform for Scaling out Web Applications Dynamically". In: *Towards a Service-Based Internet*. Ed. by E. di Nitto and R. Yahyapour. Vol. 6481. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 13–24. ISBN: 978-3-642-17693-7.

[39] J. Xin, C.-W. Lin, and M.-T. Sun. "Digital Video Transcoding". In: *Proceedings of the IEEE* 93.1 (2005), pp. 84 –97. ISSN: 0018-9219. DOI: 10.1109/JPROC.2004.839620.

[40] Y. Yazir et al. "Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 2010, pp. 91–98. DOI: 10.1109/CLOUD.2010.66.

5 Hadoop in Large Scale Data Analytics for Bioinformatics

Matti Niemenmaa[1], André Schumacher[1], Keijo Heljanko[1], Aleksi Kallio[2], and Eija Korpelainen[2]

[1]Department of Information and Computer Science
Aalto University, Espoo, Finland
Email: matti.niemenmaa+cloudbook@iki.fi, schumach@icsi.berkeley.edu, keijo.heljanko@aalto.fi

[2]CSC — IT Center for Science
Espoo, Finland
Email: aleksi.kallio@csc.fi, eija.korpelainen@csc.fi

Abstract–Due to the rapidly increasing data amounts in the bioinformatics world, new approaches to analytics are needed. Solving so-called Big Data problems, wherein the data sets are infeasible to work with on a lone computer, requires software that can perform distributed computing in warehouse-scale clusters with up to tens of thousands of nodes. Hadoop is a collection of such software, with a core consisting of the Hadoop MapReduce scalable distributed computing platform and the Hadoop Distributed File System, HDFS. We explain the principles underlying Hadoop MapReduce and HDFS as well as certain prominent higher-level interfaces to them: Pig, Hive, and HBase. We overview the current state of Hadoop usage in bioinformatics before briefly introducing our Hadoop-BAM and SeqPig projects: Hadoop-BAM is primarily a library with support for manipulating common bioinformatics data formats in Hadoop MapReduce, and SeqPig is based on the high-level Pig system and can be used directly by non-developers.

Keywords-Hadoop, analytics, bioinformatics, Big Data, warehouse-scale, MapReduce, Pig, Hive, HBase, Hadoop-BAM, SeqPig.

5.1 Introduction

Data volumes nowadays are increasing to the point that many individual data sets are too large to be analysed, or even stored, on a single computer. Such data sets are known as *Big Data*, and can arise in several contexts. Examples include Internet searches, financial analytics, and various fields of science. Notably many Big Data problems can be found in the field of bioinformatics. A number of them are due to recent advances in *sequencing*: the task of determining the base composition of e.g. DNA, possibly going as far as finding the entire genome of an organism.

In the case of DNA, the number of *base pairs* or *bp*, the building blocks of genomic information, that can be sequenced per unit cost has been growing at an exponential rate for over two decades, doubling approximately every 19 months [115]. This alone would have caused Big Data issues sooner or later. However, the growth rate suddenly increased around the year 2005, due to the emergence of techniques known as *high-throughput sequencing* or *HTS* (a.k.a. *next-generation sequencing* or *NGS*). HTS has resulted in the process speeding up to the point that the cost has now been halving approximately every five months [115]. As an example of current speeds, Pireddu, Leo, and Zanetti [98] claim that their "medium-sized" DNA sequencing laboratory can create 4–5 TB of data every week. At the high end, BGI, "one of the largest producers of genomic data in the world", generates 6 TB of data daily [75]. For comparison, the largest currently available hard drives are 4 TB in size.

Exponential growth due to technological advances is not unusual in the computing world. Consider the following three "laws":

- Moore's law: the number of components in integrated circuits with minimum cost per component doubles every year [84]. Later amended to a doubling every two years without the minimum cost aspect [83], and commonly quoted as 18 months [119]. Together with Dennard scaling [46], Moore's law has meant that processing power has doubled at essentially the same rate.
- Butters' law (of photonics): the cost of transmitting one bit over an optical network halves every nine months [102].
- Kryder's law, which was never given as a prediction, merely an observation: areal storage density of hard disk drives had been increasing at a greater rate than the rate of processor improvement according to Moore's law [127].

Note, however, that none of the above growth rates, corresponding respectively to increases in processing power, network speed, and storage capacity,

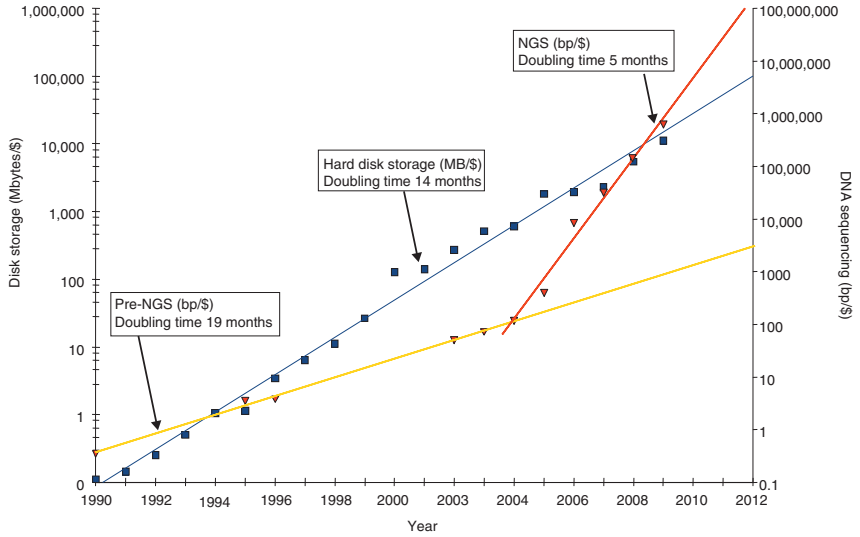


Figure 5.1: Historical trends in storage prices versus DNA sequencing costs. Reproduced from the work of Stein [115].

are even close to as fast as the current speed at which sequencing is improving. See Figure 5.1 for a clarifying plot comparing trends in storage and sequencing costs from 1990 to 2009. (For comparing the actual values instead of only the overall trends, one must know the size of a base pair, which depends on the storage format: for example, a single base is stored in 4 bits in BAM files and 8 bits in SAM files [111], excluding compression.) Note that the source of the plot describes Kryder's law as a doubling every 14 months, significantly more optimistic than more recent studies showing that the period is about 25 months [66]. Nevertheless, storing sequencing data on a hard disk is, or will soon be, actually more expensive than generating the data [115], making its storage an increasingly difficult task. Discarding all but the most informative parts may be the only long-term option.

Storage feasibility is only part of the picture: like any kind of raw data, sequencing data also needs to be analysed in order for it to be of any use. Clearly, if there is too much data for even its storage to be possible, its analysis is equally infeasible. This magnitude of data classifies sequence data analysis as a Big Data problem.

For a problem to qualify as a Big Data problem, attempting to solve it with a single computer should result in one or both of the following:

- The computer has too slow a processor or too little memory to be able to perform the needed computations in a reasonable amount of time. Waiting for better hardware will not help, because the data growth outpaces Moore's law.

- The computer does not have enough disk space to store the data sets on which computations are to be performed. Waiting for larger disk drives will not help, because the data growth outpaces Kryder's law.

Therefore, in order to solve Big Data problems, lone computers are insufficient. *Distributed* computing is required, i.e. having multiple networked computers working together in computer *clusters*. Ideally, the clusters used have been specialized for the task at hand, thus making them effectively *warehouse-scale computers* [18].

Traditionally, distributed software has been created by developing communication protocols specific to the application, using primitives provided by e.g. the Message Passing Interface (MPI) [23], the PVM framework [117] or, in the data communications domain, the Erlang programming language [15]. At this level, implementing the necessary functionality correctly is difficult, especially if the software is to be run not only in small clusters but on warehouse-scale computers, with hundreds to tens of thousands of network nodes. Realizing high performance in such an environment is especially complicated. In addition, fault tolerance becomes a necessity, because the probability of hardware failure is too high to ignore [43].

To ensure that warehouse-scale distributed software can work at high performance and not worry about hardware failure, a framework specifically designed for that use case is necessary. One such framework was developed by Google [52]: *MapReduce* [45] coupled with *GFS* (the Google File System) [50]. Together, they provide fault tolerance both for computations and data: most hardware failures neither interrupt running processes nor cause data loss.

The implementations of the MapReduce system and GFS were not made publicly accessible, leading to the creation of *Hadoop* [5], an open source implementation of the same ideas. Hadoop has since expanded to become a collection of software related to scalable distributed computing.

Unfortunately, there exist problems for which MapReduce's computational model is far from ideal. MapReduce is specifically optimized for throughput over latency, which makes it a poor fit for *interactive* use. Interactive analysis tasks arise when users are not well acquainted with the data sets concerned and must thus *explore* them with repeated queries, either narrowing down areas of interest or requesting more information according to

newly realized needs. MapReduce's typical ten-second job startup time [95, 132] guarantees that most users will shift their focus before a computation completes, slowing down this exploratory process [26]. Interactive tasks are increasingly prevalent in sequencing data analysis [33], making frameworks designed with latency in mind desirable. Low latency is a more difficult goal to reach than high throughput [44, 94], but nevertheless such systems do exist. Two notable ones are Spark [134] and Impala [63], which is based on the design of Google's Dremel [80].

This Chapter proceeds as follows. In Section 5.2 the background and details of MapReduce are explained in detail. Next, Section 5.3 covers the Hadoop project and some notable high-level frameworks based on it. Section 5.4 surveys the current state of Hadoop in bioinformatics and presents two sets of tools we have developed that enable using Hadoop to manipulate and analyse sequencing data. Finally, Section 5.5 provides some closing remarks.

5.2 MapReduce

Applying warehouse-scale computing to Big Data problems is not as simple as setting up the hardware. Programming for a warehouse-scale computer is a far more complex task than programming for a small cluster, which in itself is more challenging than programming for e.g. a typical desktop system. This is especially the case when performance is a concern, since effectively utilizing all available resources involves co-ordinating several hardware and software layers. Examples of things to keep in mind are the complex memory hierarchy, heterogeneous hardware, failure-prone components, and network topology [18]: all in addition to the complexity of implementing the core of the application itself. As such it is no surprise that programming frameworks that ease the burden on the developer of warehouse-scale applications have been created. MapReduce [45] is one such framework, including automatic handling for data distribution and fault tolerance.

MapReduce is intended especially for working with large data sets, i.e. Big Data. As such it is also intended for warehouse-scale computers, which means that in order to be practical, it must be able to tolerate hardware failure. Even with unrealistically reliable servers with a mean time between failures (MTBF) of 30 years, if there are 10 000 servers in a cluster, it will experience on average one failure every day [18]. This makes fault tolerance in software not only useful, but a practical necessity. In addition, it allows for a better price/performance tradeoff by using relatively unreliable, cheaper hardware [19]. MapReduce has been designed with this in mind: it provides

efficiently fault tolerant computations and is intended to be used together with certain file systems that provide fault tolerant data storage.

At its simplest level, MapReduce is a programming model for transforming data: the programmer need only specify two functions—the *Map* and *Reduce* functions—and the input data, and based on this information the corresponding output can be computed in a functional manner. Because of this, the model also allows for a simple strategy for fault tolerance: as re-executing a function will result in the same output, e.g. all computations on a failed computer can trivially be re-executed on another computer, as long as the input data is still available. The MapReduce model allows for easy parallelization (demonstrated in Section 5.2.1) and is relatively simple to program for, making it an attractive choice for distributed computing. However, the term "MapReduce" in a distributed computing context is generally understood as meaning more than just the abstract programming model: it includes the associated implementation that handles scheduling the computation efficiently and dealing with machine failures during execution.

The original MapReduce implementation [45] was developed internally at Google and has not been released to the public. The current *de facto* open source implementation of MapReduce is Apache Hadoop [5], which will be discussed in Section 5.3. Hadoop's existence makes MapReduce an attractive choice as a distributed computation model because Hadoop is well established, having seen use in a variety of fields with good results. (See Section 5.3 for detailed information.)

MapReduce is not perfect, though: its programming model can be considered too rigid for various tasks. PACT [2] has been explicitly designed as an extension of MapReduce with the ability to express more complex operations. Spark [134] instead emphasizes data re-use: MapReduce does not intrinsically allow re-using intermediate results. If such re-use is desired, it must be done by manually saving and loading the corresponding data, which can incur needless I/O and serialization overheads. In spite of these limitations, however, the MapReduce model continues to see use across a wide variety of applications.

In the following Sections we will discuss the MapReduce programming and execution model in detail before delving into the file system that MapReduce is typically paired with. The information on MapReduce is based completely on the works of Dean and Ghemawat [45] and White [130].

5.2.1 Execution Model

Conceptually, the execution model of MapReduce consists only of applying the *Reduce* function to the grouped results of the *Map* function. However,

practical distributed MapReduce frameworks complicate the process: they specify more steps and implement them in certain ways to ensure that good performance and fault tolerance are achieved. Below, we first briefly explain the simpler, conceptual model, and then consider the principles that underlie the warehouse-scale implementations.

The type signatures of the two user-specified functions form a concise description of the conceptual MapReduce execution model. See the following, where k is short for "key" and v for "value", the subscripts serve to differentiate the types, and the superscripts $m \geq 0$, $n > 0$, and $p \geq 0$ denote differing list lengths:

$$\begin{array}{rcccc} \mathit{Map} & : & (k_1, v_1) & \rightarrow & (k_2, v_2)^m \\ \mathit{Reduce} & : & (k_2, v_2^n) & \rightarrow & v_3^p \end{array}$$

As can be deduced from the type signatures, a MapReduce computation takes a sequence of key-value pairs as input, on which it performs the following tasks:

1. The *Map* function is applied to each key-value pair in the input, outputting any number of new key-value pairs for each one.
2. Each key in the output from the previous step is paired with all the values that were associated with that key.
3. The *Reduce* function is applied to each pair in the result of the pairing in the previous step. The resulting list of data forms the final output.

To clarify the process, consider the following simple example, where the task consists of taking as input a set of documents and outputting, for each word encountered, the set of documents it was found in. Here the input type could be e.g. $(k_1, v_1) = (\texttt{document-name}, \texttt{contents})$ for each document. The *Map* function would go through the contents, outputting pairs of type $(k_2, v_2) = (\texttt{word}, \texttt{document-name})$. Thus the *Reduce* function receives as input pairs of the form $(\texttt{word}, \texttt{document-name}^n)$, which is precisely the set we are interested in. The final output v_3^p would depend on the exact format in which the output is desired, but could be e.g. a string (just one string, i.e. $p = 1$) of the form `"word","document-1-name","document-2-name",...` for each word.

While the above description is sufficient for implementing a basic MapReduce framework, fully distributed systems for warehouse-scale computers such as Google's MapReduce implementation and Apache Hadoop are more complex and perform the steps in a very specific way. See Figure 5.2 for a graphical overview of MapReduce execution on such a system.

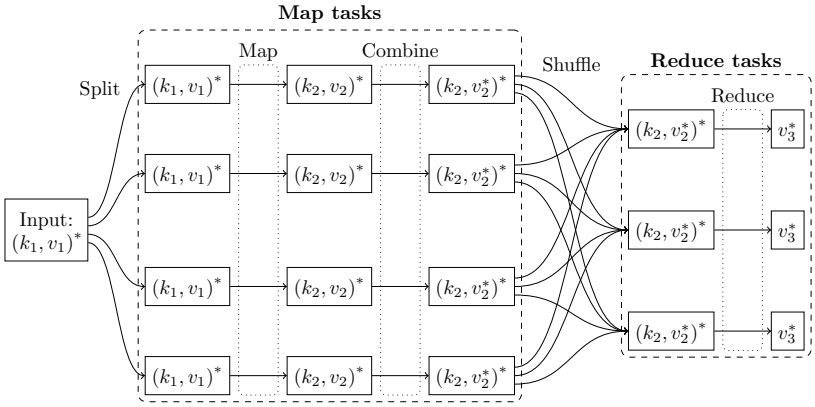


Figure 5.2: Distributed MapReduce execution with four map tasks and three reduce tasks. k_i and v_j denote key type i and value type j respectively. The asterisk superscripts denote unknown list lengths.

Distributed MapReduce is structured as a master-slave system. The master node (known in Hadoop as the *jobtracker* and predefined as a specific node for the whole cluster) allocates workers for different parts of the computation and co-ordinates communication between them. The slaves (known in Hadoop as *tasktrackers*) are the nodes that actually read the input data, run the *Map* and *Reduce* functions, and write the output data. Each slave node provides a number of map and reduce *slots* for running the two different functions. For a computation or *job*, typically the user selects the number of reduce tasks to be performed while the MapReduce system automatically determines the number of map tasks. The full execution process is as follows:

1. **Split** This step is performed solely on the master. The input files are conceptually split into chunks: a set of *splits*, i.e. tuples that identify sequential parts in the input files, is created. These splits are typically tens of megabytes in size, often corresponding to the block size of the file system in use (see Section 5.2.2). Based on this the master creates a map task for each split and assigns as many of them as it can to separate map slots, which are started up and begin running.

2. **Map** Each map task involves reading the corresponding input split, forming key-value pairs of the data therein, and handing them to the *Map* function for processing. These intermediate key-value pairs are written to local disk, sorted by key, and *partitioned*: differentiated based on which reduce task they belong to. The default partitioning is

based on simply assigning each key k to the reducer $h(k)$ mod R where h is a hash function [65] and R is the number of reduce tasks.

3. **Combine** This is an optional step that essentially runs the *Reduce* function on the partitioned output of the *Map* function directly as part of the map task. While a custom *Combine* function can be given, typically *Reduce* is used as-is. This use requires that it be commutative and associative. Note that since combining can reduce the map task's output size, it is performed before writing the partitions to local disk, as long as the task has enough available memory for in-memory sorting and partitioning. This way, fewer I/O operations are performed.

4. **Shuffle** The map tasks communicate the locations of their partitioned outputs to the master node. It then notifies the corresponding reduce tasks (starting them up in reduce slots as required) that new data is available. The reduce tasks read the data from the local disks of the nodes where the data was written—note that this may be the same node on which the reduce task itself is running, in which case no network communication is required. When a reduce task has received all of its input data, it sorts it so that it is grouped by key.

5. **Reduce** Each reduce task iterates over its sorted sequence of key-value pairs, passing each unique key and corresponding sequence of values to the *Reduce* function. The output from it is written directly to the output file of the reduce task, which is one of the final output files generated by the MapReduce computation.

The end result is a set of output files, one from each reduce task. They are not automatically combined to a single file because that is not always necessary: they could be used as-is as inputs for another MapReduce job, for example. It is also possible to run a *map-only job* in which only the *Map* function is used, with the map tasks' output forming the output for the entire computation.

Fault tolerance in this kind of a fully distributed MapReduce system is fairly simple to implement. The master node periodically pings the slaves, assuming them to have failed if it does not receive a response in time. In-progress tasks on failed nodes are rescheduled and eventually restarted. Completed map tasks are also rescheduled, but completed reduce tasks are not: the input and output files are assumed to be on a shared storage system, separate from the local disks that are used for storing the intermediate output from map tasks. Thus, if a node with a completed map task whose output has not yet been sent to a reduce task fails, the map task needs to

be restarted, but if a node with a completed reduce task fails, nothing needs to be done. This way worker failure is fully accounted for, which is important for long-running jobs at warehouse scale. In contrast, master failure is deemed unlikely since it requires a specific node to fail, and is not handled at all, making the master a single point of failure.

Sometimes worker nodes may have unexpectedly poor performance due to e.g. faulty hardware. This results in *stragglers*: members of the last few map or reduce tasks which take a particularly long time to complete, holding up the whole computation. A key optimization in MapReduce systems, that of *speculative* or *backup* execution, was designed to mitigate this problem. After all tasks have been started, if some tasks have been running for a relatively long time and seem to be progressing (performing I/O of key-value pairs) relatively slowly, the master attempts to reschedule those same tasks on different nodes. When a task is successfully completed, any other executing duplicates of that task are stopped. Speculative execution does not significantly affect the resources used by a job but can offer significant speedups.

Since tasks can run multiple times as well as be restarted at any point, the *Map* and *Reduce* (and *Combine*) functions should be free of side effects: pure functions of their input values. Only then is it guaranteed that all the output of a fully distributed MapReduce system is equivalent to a single sequential execution of the program. In the face of nondeterministic user-supplied functions, the output of each reduce task may correspond to a different sequential execution. Whether this inconsistency is a problem in practice depends on the application.

5.2.2 Distributed File System

MapReduce is traditionally paired with a specific distributed file system, designed for large files and streaming access patterns. For Google's MapReduce that file system is GFS (the Google File System) and for Hadoop it is HDFS (the Hadoop Distributed File System). Both share similar design principles and implementation strategies, which will be covered in the remainder of this section. Information on GFS in this section is based on the work of Ghemawat, Gobioff, and Leung [50] and information on HDFS is based on the book by White [130], except where otherwise indicated.

GFS and HDFS are both, like MapReduce, master-slave systems. The master node (known in Hadoop as the *namenode*) keeps track of file metadata and the state of the slaves, and the slave nodes (known in Hadoop as the *datanodes*) are responsible for all data storage and communication. Replication is used to provide fault tolerance: each block is stored on multiple

slaves—three by default. For simplicity reasons [79] the master node is a single point of failure, though HDFS's *secondary namenode* can limit data loss in case of catastrophic master node failure.

When using MapReduce, the slave nodes should be used to run MapReduce workers as well, allowing MapReduce to take advantage of data locality for map tasks. This is done by scheduling map tasks on nodes where the data for that task's split is stored, or, failing that, on nodes that are nearby in terms of the network topology. Replication is advantageous here as well as for fault tolerance, since it improves the odds of being able to schedule a task on a node that has the corresponding split's data available locally. Note that it is possible to run a MapReduce job on GFS and HDFS without any of the input data being sent across the network.

A major design principle of both GFS and HDFS is to support large files efficiently. "Large" in this context means at least 100 megabytes, but typically several gigabytes, and up to terabytes. In contrast, small files are assumed to be rare, and so are not optimized for at all. This is very much the opposite of what file systems are traditionally optimized for [51], which is one of the main reasons that GFS and HDFS are typically paired with MapReduce; they are both intended for large files.

Another important design principle of GFS and HDFS is the emphasis on *write-once, read-many* operation and streaming reads: written files are assumed to be modified rarely if at all, and workloads are expected to include reading entire files or at least significant portions of them. Random reads and writes are not optimized for—in fact, HDFS does not support random writes at all. This lack of arbitrary modifications makes implementing replication much simpler, and the philosophy of large reads makes bandwidth far more important than latency. Once again, this ties in with the way MapReduce works, but it is also a more generally helpful restriction for scalable storage architectures: for example, the lowest layer of the Windows Azure Storage [25] system has the same limitation.

A notable result of these design decisions is that the block size of both GFS and HDFS is unusually large: 64 megabytes. (HDFS does allow changing this, but reducing it to usual file system block sizes would be self-defeating.) This reduces overhead related to metadata management, mainly by drastically reducing the amount of metadata: compared to a more traditional 4 KB block size and assuming a large enough file, 16 384 times less blocks have to be kept track of. Thus metadata can be kept fully in the memory of the master, making metadata operations fast and enabling easy rebalancing (replica distribution) and garbage collection. Keeping metadata in memory has a drawback, however: now the capabilities of the master limit the number of files that can be stored [79]. Another benefit of large block

sizes is that if the time to read a full block is much greater than the physical disk seek time, reading a file consisting of multiple arbitrarily distributed blocks operates close to the disk's sequential read rate.

5.3 Hadoop

Apache Hadoop [5, 130] was originally conceived as a nameless part of the Nutch [10] Web search engine, implementing open source versions of MapReduce [45] and the Google File System (GFS) [50] for its own purposes, in the Java programming language [54]. Yahoo! [133] soon began contributing to the project, at which point these components were separated, forming the Hadoop project, named after Doug Cutting's (the creator) child's toy elephant. At around the same time, Hadoop began to be hosted by the Apache Software Foundation [122], giving it the full name "Apache Hadoop". Since then, Hadoop has grown to become a collection of related projects, two of which are the original MapReduce and file system components: Hadoop MapReduce and HDFS (the Hadoop Distributed File System).

For most of Hadoop's history, the MapReduce component has been the only computational framework supported in Hadoop. Tasks running on other systems, e.g. MPI [23], have not been able to be scheduled on Hadoop clusters. This has meant that the machines in a cluster should be configured to run only one class of tasks, such as Hadoop MapReduce jobs or MPI processes. Otherwise, one node may have several computationally intensive tasks running at once, possibly resulting in resource starvation issues such as running low on memory or disk space, which may in turn cause all tasks on the node to fail. On the other hand, the traditional solution of partitioning the cluster by framework can lead to poor resource utilization, with some machines remaining completely idle while there is work to do, just because they have been configured for a different framework. Apache Mesos [9, 59] is a cluster manager with cross-framework scheduling, solving this problem more effectively. The current beta releases of Hadoop include their own similar system, called *YARN* (Yet Another Resource Negotiator [128]) [6, 57, 85], also known as NextGen MapReduce or MapReduce 2.0. In addition to cross-framework scheduling, YARN also removes the concept of map and reduce slots from MapReduce slave nodes, instead dynamically allocating map and reduce tasks according to what is most needed at the time. YARN takes over some of the cluster management responsibilities currently handled by Hadoop MapReduce, allowing other computational frameworks to effectively co-exist within Hadoop.

Several companies offer their own distributions of Hadoop, for which they

also provide commercial support. The most notable ones are Cloudera [34], Hortonworks [61], and MapR [74]. They are naturally all major contributors to Hadoop, but have their own extensions as well. Hortonworks's distribution is the only one with support for running on Windows Server. Their contributions are also particularly noteworthy for their *Stinger Initiative* [113], which involves improving the performance of the Hive project, which is presented in Section 5.3.2. Cloudera Impala [63] is a distributed query engine meant for interactive use, as opposed to MapReduce's emphasis on throughput. MapR's distribution provides fault tolerance for the master in both MapReduce and HDFS: the jobtracker is restarted on failure and the namenode is fully distributed. MapR is also unique in that it does not use HDFS; a complete rewrite in the C++ programming language [116], whose interface is nevertheless compatible with HDFS, is used instead.

Usage of Hadoop within an organization is unlikely to encompass the entire range of Hadoop-related projects. Some may not even use the MapReduce component, due to the existence of other computational engines and YARN. One thing, however, is common to almost all users of any part of Hadoop: HDFS. The amount of data an organization has stored in HDFS is an indication both of how much the organization uses Hadoop and of what kinds of data volumes Hadoop has been used for. For demonstration purposes, the following is a sample of HDFS usage, co-incidentally all from the year 2010:

- Facebook [47] stored 15 PB of data with 60 TB being added daily, and with compression reducing the space usage to 2.5 PB and 10 TB respectively [123].

- Yahoo! had over 82 PB of data among over 25 000 servers split into clusters of about 4 000 [101] servers each.

- Twitter [126] had "(soon) PBs of data", with 7 TB of new data coming in every day [129].

Presumably these data volumes have only increased since then.

The previous Section already detailed MapReduce and HDFS. In the following Sections we will instead discuss three prominent open source projects related to Hadoop, each offering its own higher-level abstraction on top of Hadoop MapReduce and HDFS. Pig offers a high-level language for expressing MapReduce programs, Hive provides a data management and querying system using an SQL-like language implemented with MapReduce, and HBase allows scalable random access into a key-value store in HDFS.

5.3.1 Pig

Apache Pig [11, 91], originally developed by Yahoo!, is a high-level interface to MapReduce, providing a custom query language for bulk data manipulation called *Pig Latin*. It is compiled into a sequence of MapReduce computations which are executed on Hadoop. Pig drastically lowers the bar of using Hadoop MapReduce, giving users a richer pool of primitives they can use to describe their computations and not requiring them to implement it in Java, a far more low-level programming language than Pig Latin. This can greatly simplify development and maintenance, improving programmer productivity. Similarly, Pig can be used as a high-level way of implementing a so-called *Extract, Transform, and Load* (ETL) pipeline [112].

Pig treats all data as *relations*. Relations are defined as bags (multisets) of tuples. The fields in the tuples can be simple values like integers or strings, but also complex like key-value mappings or even other bags and tuples—arbitrary nesting is allowed. Tuples in a relation are not constrained in any way: they can have different numbers of fields as well as different field types in the same position. It is possible, however, to define a *schema* which specifies a common type for the tuples in a relation. Without a schema, Pig infers a "safe" type for every field (such as double-width floating point for all numbers), which can cause performance to suffer.

The data model is similar to that used by traditional relational database systems [38] but more flexible. The lack of a defined ordering is particularly useful for MapReduce processing, as it does not restrict the partitioning strategy (how map outputs are spread among the reducers) in any way. In addition, allowing arbitrary nesting can simplify operations compared to only having flat tables, especially if they are normalized [42], since all data can be kept in one relation instead of having to perform join operations when needed.

Pig Latin has several *commands* for working with relations, and more are being added as development proceeds. The following list is incomplete but representative:

- `LOAD` and `STORE` interact with external storage, respectively reading and writing relations.
- Standard embarrassingly parallel commands: `FOREACH` transforms every tuple in a relation and `FILTER` selects tuples from a relation based on a condition.
- Commands related to ordering and equality: `ORDER BY` performs sorting, `RANK` adds fields describing sort order but preserves the existing order, and `DISTINCT` removes duplicates.

- Grouping: `GROUP` a.k.a. `COGROUP`, which can be applied to more than one relation at a time.

- Joins: `CROSS` and `JOIN` can be used to respectively form the Cartesian product or any kind of inner or outer join [55, 114] of two or more relations.

Most commands can utilize *functions* to specify their exact effects. For example, `FOREACH` could be used as `FOREACH r GENERATE f(x)` where `r` is a relation, `f` a function, and `x` a field contained in the tuples of `r`. The result of the command is a relation containing 1-tuples whose values are given by the function `f` on the field `x` of each tuple in `r`. There are many built-in functions, including arithmetic operators as well as aggregating functions such as `COUNT`, which computes the number of tuples in the given relation.

Clearly, these operations by themselves are much more expressive than the MapReduce model, but Pig Latin can also be extended by users. While the command set cannot be changed without modifying Pig itself, new functions can easily be added. Furthermore, the flexibility of the data model means that all user-defined functions can be used in any function-using command without restriction, unlike e.g. in Hive where `SELECT` clauses only allow using scalar functions.

Pig has been widely adopted. In June 2009 at Yahoo!, 60% of ad-hoc and 40% of production Hadoop MapReduce jobs came through Pig, and further increases in Pig usage were expected [48]. A cross-industry study performed in 2012 showed three out of seven analysed clusters having significant Pig usage, one of which was observed to have had over 50% of MapReduce jobs submitted via Pig [33]. LinkedIn [72] uses Pig both for user-facing data set generation and for analytics [16]. The reported runtime increase when using Pig instead of hand-written MapReduce has ranged from a factor of 1.3 [112] to 1.5, but it has improved significantly over time and is likely to continue to do so [48]. This level of performance loss seems to be acceptable in practice: consider that Twitter was using "almost exclusively" Pig for its analytics in 2011 [71].

5.3.2 Hive

Apache Hive [8, 124, 125] is a data warehouse system built on top of Hadoop: essentially, it is a high-level interface to both MapReduce and the backend storage system, which is typically HDFS, but can also be HBase (see Section 5.3.3). Hive enforces a structural view, very similar to traditional relational database systems [38], of the data sets it handles. They are queried

and manipulated using a language similar to SQL [28, 55, 114] called *HiveQL*, which is translated to MapReduce computations. Hive was originally developed by Facebook; later, Google created a very similar warehousing solution called Tenzing [32]—a rare example of outside ideas being incorporated so directly at Google, instead of the other way around.

Hive's data model is based on *tables*, akin to those used in relational databases. Records of data are stored in *rows*, which are split among a set of typed *columns*, which are in turn defined in a *schema*. A row may have a null value in any column, but each row in a table always has the same amount of columns. Possible column types include primitive types such as integers and strings as well as complex types: arrays, key-value mappings, and product and sum types called *structs* and *unions*.

All metadata about the tables managed by Hive is catalogued in the *metastore*. The existence of the metastore, i.e. keeping track of persistent metadata about data sets, is what makes Hive a data warehouse system as opposed to purely computational systems such as Pig. The metastore remembers all tables and all information about them; primarily their schemata. Because it is randomly accessed, the metastore is not stored in HDFS. Instead, a traditional relational database is used.

Various settings for performance tuning may be applied to tables in Hive. Tables can be *partitioned* on certain columns, so that rows with the same combination of the partitioned columns' values are stored together. Partitions may furthermore be *bucketed*, which is another layer of partitioning based on the hash of a single column. Table rows can also be stored in sorted order. When using HDFS storage, tables map directly to directories, partitions to subdirectories of the table's directory, and buckets to files in the partitions' directories.

Notably, even though Hive manages storage of tables, it does not rely on any particular file format. As long as the contents of each file can be serialized for storage and deserialized (using a Java class called a *SerDe*) for manipulation in HiveQL according to the table's schema, the files comprising the data of one table can even be in completely different storage formats.

Hive supports *indexing* on table columns, a classical strategy for speeding up query operations in databases. The trade-off is that the index takes up some additional storage space and modifications become slower as the index needs to be updated. Considering that Hive's main use case, data warehousing, consists of managing very large and mostly immutable data sets, the slowdown is irrelevant and the amount of space taken by the index is likely to be negligible, whereas the query speedup is likely to be very welcome. Hive currently provides two kinds of indices: one that identifies HDFS blocks for the rows corresponding to a given key, and a bitmap index [29] that also

identifies which rows in the blocks are populated with that key.

HiveQL currently has two kinds of data manipulation statements: `LOAD`, which simply copies data files into the appropriate HDFS directory of the table, and `INSERT`, which writes the results of a `SELECT` clause into a table while performing appropriate format conversions. `LOAD` is an optimization, relying on the user to make sure that the file is usable in the table as-is, lest the table end up in an unusable state. `INSERT` is more flexible, as it can insert into more than one table at once and compute the partitioning dynamically. There are no other manipulation statements: HiveQL currently has no way of updating or deleting rows. This makes sense given that rows are typically stored as-is in files in HDFS.

Querying in HiveQL is done with the `SELECT` statement, like in SQL. Various clauses to modify the statement's behaviour are supported, as in any modern SQL system. The following is a sample of what is available:

- `WHERE` selects only rows for which a given condition is true.
- `DISTINCT` removes duplicates from the result.
- `GROUP BY` groups data by the given columns' values.
- Sorting clauses: `ORDER BY` and `SORT BY`, the latter of which only guarantees sorting the output of each reduce task, thereby forming a partially ordered result. `ORDER BY` performs a global sort, but its current implementation is poor: all data is sent to a single reduce task for sorting [60]. This issue is to be fixed for the next version of Hive, which has not yet been released at the time of writing.
- Combining results of multiple selections in one query with `UNION ALL`.
- Joins: the various forms of `JOIN` can compute any form of inner or outer join [55, 114] of two or more tables, as well as the Cartesian product.

All in all the functionality available is very similar to that offered by Pig Latin, though HiveQL is not quite as flexible due to Hive's stricter data model. Nevertheless, just like Pig Latin, HiveQL can also be extended by users via user-defined functions. Hive users can define three kinds of functions: ordinary ones, which simply transform one row into another and are therefore always run within map tasks; *table-generating functions*, which can transform one row into multiple rows; and *aggregation functions*, which can combine multiple rows together and thus are run in reduce tasks.

Hive also has support for creating *views* based on `SELECT` queries. Views are essentially named queries that are saved in the metastore, which can

themselves be queried just like tables can. Conceptually, when a view is queried, the result of the view's defining query is computed, and then the original query is evaluated on that result. In practice, the two queries may first be combined into a single one which is executed directly on the tables used.

Hive has seen wide adoption. As the originator of Hive, Facebook is naturally a heavy user, with over 20 000 tables and several petabytes of data in a Hive cluster in 2010 [123]. LinkedIn uses primarily Hive and Pig for its internal analytics [16]. A cross-industry study performed in 2012 showed four out of seven analysed Hadoop clusters having significant Hive usage, three of which had 50% of their MapReduce jobs, sampled over time periods ranging from days to months, submitted via Hive [33].

As Hive is used especially for analytics, the fact that it makes use of the purely throughput-optimized MapReduce as a computational backend has been considered problematic. In an interactive setting the startup costs of a Hadoop MapReduce job are not necessarily insignificant, as they can even dominate the execution time of short computations [95]. Google has also recognized this limitation of MapReduce, creating its own interactive SQL-like query system called Dremel [80]. Two notable freely available Hive-compatible systems that do not use MapReduce and are tailored for exploratory analysis have been created: Shark [132], which is based on Spark [134], and Impala [63]. Apache Drill [4] and Apache Tez [12, 120] are other interactivity-oriented efforts, but are still in early stages of development.

5.3.3 HBase

Apache HBase [7] is an open source version of Google's Bigtable [30]: essentially a distributed data storage system, enabling random read-write access to individual records in Big Data sets. This is a key advantage over MapReduce, which only provides streaming access. In addition, as bulk operations on HBase tables can be performed using Hadoop MapReduce, no functionality is lost by relying on HBase instead of HDFS for data storage—though performance is lower than using HDFS directly. HBase was originally conceived by Powerset as a foundation for their natural language search engine [49]; though the engine never materialized, HBase continues to be developed under the Apache Software Foundation.

HBase provides sorted three-dimensional lookup tables in a manner similar to traditional relational database engines, but with a much simpler data

model, namely:

$$(row : \text{string}, column : \text{string}, version : \text{int64}) \rightarrow \text{string}$$

In other words, each data value, or *cell*, in a *table* is uniquely identified by a *row*, *column*, and *version*, of which the rows, columns, and values are simply arbitrary byte strings while versions are 64-bit integers—typically timestamps. Data is sorted first by row, then by column, and finally by version, with later versions coming first in the sort order. This simple model allows scaling by just adding more nodes, without having to worry about maintaining the complex invariants required by relational databases [121, 130].

HBase has a very simple interface to tables, consisting of only four operations (excluding metadata-related functionality):

1. *Get*: reads a row, possibly with further limitations to specific columns and/or versions.
2. *Put*: writes a row, either creating a new one or overwriting an existing one.
3. *Delete*: removes a row.
4. *Scan*: iterates over a sequential range of rows, returning one at a time to the user.

This limited set of functionality makes HBase's essential nature as a key-value store evident: HBase itself does not provide the more complicated operations that are typically found in database systems, such as joins. As previously mentioned, however, Hive can use HBase as a storage backend, allowing that kind of functionality to be used on data stored in HBase.

As MapReduce handles scheduling computations on a distributed system, so does HBase take care of distributing the data it stores among the available nodes. Tables in HBase are automatically partitioned into sequences of rows called *regions*, which can be distributed among the HBase servers, aptly called *regionservers*. This spreads out computational load on the table as well as the data itself, enabling large tables to utilize the entire cluster's storage space.

HBase naturally also includes fault tolerance, which is mostly reliant on a reliable storage system, typically provided by HDFS. As with MapReduce and HDFS, it is based on a master-slave architecture where the master only co-ordinates the slaves and monitors their health. The aforementioned regionservers are the slaves in an HBase cluster. Unlike MapReduce and HDFS,

HBase provides fault tolerance for the master node: this is facilitated by using ZooKeeper [13], a co-ordination service based on the Zab algorithm [64] (similar but not identical to the classic Paxos [67]). ZooKeeper is used to make sure that only one master is active at any given time, and also to store various metadata about the cluster.

Fault tolerance on the regionservers requires some work due to the method used to implement write operations. For performance, writes (including modifications and deletions) are performed on in-memory caches called *MemStores* (in Bigtable, *memtables*) and only flushed periodically, to HDFS files called *StoreFiles* or *HFiles* (corresponding to the Bigtable *SSTables*, short for Sorted String Tables [79]). Data loss is prevented by also logging writes to HDFS: when a regionserver fails, its log is replayed by all replacement regionservers (i.e. all servers that are assigned any region that was previously assigned to the failed server), bringing them up to date.

Recall that HDFS does not allow modifying files. Thus, whenever a regionserver decides to flush a MemStore to HDFS, it creates a new StoreFile for the cache's contents. Read operations must, in the worst case, consult the MemStore as well as all StoreFiles. As data is kept in sorted order, e.g. reads requesting only the latest version of a record might need to consult only the MemStore, but in the worst case, a read operation involves traversing the whole MemStore as well as all StoreFiles before the appropriate values to return are found. To prevent having to consult too many StoreFiles, they are periodically merged into a single StoreFile in a process called *major compaction*. At this point, all deletions are also fully handled: when a cell that is not currently in the MemStore is deleted, the delete operation is merely noted in a marker called a *tombstone* and eventually flushed, but the supposedly deleted cell still persists in the older StoreFiles. The cell is actually removed from storage only during a major compaction: it and the corresponding tombstone are not written into the final, merged StoreFile. *Minor compactions*, in which only a subset of the StoreFiles are merged and deletions are not processed, also occur occasionally.

Figure 5.3 provides a graphical overview of how operations in HBase affect the different kinds of state. In summary:

1. Write operations, including additions, modifications, and deletions, are logged and then applied to the MemStore.

2. The MemStore is eventually flushed, creating a new StoreFile.

3. StoreFiles are eventually merged together into a single StoreFile during a minor or major compaction.

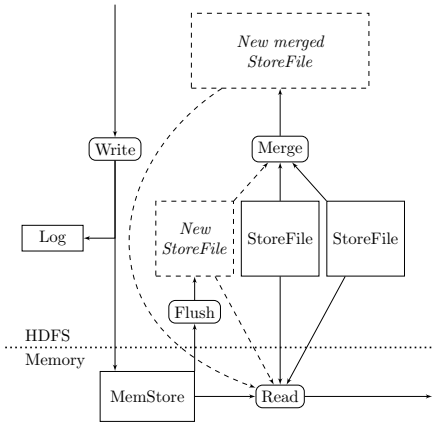


Figure 5.3: HBase state and operations. "Read" includes both single-row reads and scans and "Write" includes single-row additions or modifications as well as deletions. The boundary between HDFS and the MemStore is shown as a dotted line.

4. Read operations access all StoreFiles and the MemStore.

Since StoreFiles are written only when flushing or compacting, the amount of records written at a time is typically quite large. Therefore compression can be utilized more effectively than in systems that simply append or modify existing files: each StoreFile can be compressed as a whole at its creation time, resulting in a better compression ratio than could otherwise be achieved. Additionally, as major compactions are usually run when the HBase cluster is not under heavy load, it is possible to apply a relatively resource-intensive but effective compression algorithm on a large amount of data at once, improving compression ratios even further.

Having to read from several HDFS files for every read operation would be prohibitively slow. Hence, to speed up reads, regionservers cache parts of StoreFiles as well as individual lookup results, and allow using Bloom filters [22] to quickly exclude StoreFiles from being considered for a query. Bigtable tests by Chang et al. [30] show that despite these efforts, random-access reads are approximately an order of magnitude slower than similarly random writes, and sequential reads can be either significantly slower or faster than sequential writes. Results from the Yahoo! Cloud Serving Benchmark [39] have demonstrated similar behaviour in HBase: in 2010, HBase dominated the competition in write-heavy workloads, but was comparatively slow in performing read operations.

Facebook has used HBase heavily with positive results: in 2011, Facebook's HBase clusters consisted of thousands of nodes implementing various applications, including real-time messaging among millions of users [1, 24]. Several other industrial users of HBase exist [58], but none have (or have published information about) notably large cluster sizes or data volumes.

5.4 Hadoop in Bioinformatics

The field of bioinformatics contains a large number of Big Data problems, especially in sequencing data analysis. The tools offered in the Hadoop project have been heavily used in implementing various solutions, although other systems—mainly the Message Passing Interface, MPI [23]—have been the method of choice for some projects [118].

A task that has seen a significant amount of attention is *sequence alignment* or *mapping*: similarity search between two or more sequences in order to estimate either the function or genomic location of the query sequence. Alignment is an important part of almost any analysis process. As such, it is not surprising that much effort has been spent in developing efficient and scalable alignment methods.

CloudBurst [106] and CloudAligner [86] are examples of sequence aligners based on Hadoop MapReduce. CloudAligner is notable in that it uses map-only jobs to achieve greater performance. The publication that presented the Hadoop-based CloudBLAST [77] compared it against a similar MPI implementation, mpiBlast [41], finding that CloudBLAST performed up to approximately 30% better and was simpler to develop and maintain. Many MPI-based aligners [14, 82, 103] have nevertheless been created.

Alignment tools often include other features, either as additional utilities or because they are intended for some specific analysis for which alignment is only a subtask. The following examples all use Hadoop MapReduce for scalability. Seal [98, 99] provides an aligner which includes postprocessing, such as duplicate read removal. Crossbow [69], Myrna [68], and SeqInCloud [81] implement sequence alignment as part of their specific analysis pipelines.

Sequence alignment is, of course, not the only analysis task in bioinformatics for which Hadoop has been utilized. The SeqWare Query Engine [90] uses HBase to implement a database for storing sequence data. MR-Tandem [100] carries out protein identification in sequence data using MapReduce. CloudBrush [31] and Contrail [105] use MapReduce in performing a process called *de novo assembly*: assembly of previously unknown genomes from sequence data. SAMQA [104] detects metadata errors in sequence data files, using MapReduce for parallelization.

Finally, some projects provide support facilities, making it easier for their users to implement the complete analysis pipelines. The Genome Analysis Toolkit (GATK) [78] is one example. It is based on the MapReduce model but does not use Hadoop, instead running on a custom engine and having a separate wrapper for distributed computing called GATK-Queue [17]. The aforementioned Seal project, while focused on alignment, presents its functionality as a set of tools that can have other uses as well. Cloudgene [107] is a platform providing a graphical user interface for executing bioinformatics applications based on Hadoop MapReduce, with support for several of the tools mentioned here. BioPig [89] is a Pig-based framework containing various useful functions, including wrappers for some other commonly used applications.

We have developed two supporting tool sets of our own, offering useful functionality that was not previously available. Hadoop-BAM is a library providing file format support along with some useful command-line tools. SeqPig is a higher-level interface in Pig including special functionality for sequence data analysis. They are presented in the following two Sections.

5.4.1 Hadoop-BAM

Hadoop-BAM [56, 87, 88] is a library written in the Java programming language, providing support for using Hadoop MapReduce to manipulate sequencing data in various common file formats. Currently the formats supported are all of the following:

- Sequence Alignment/Map or *SAM* as well as its binary representation, Binary Alignment/Map or *BAM* [70, 111]. Originally only BAM was supported, giving Hadoop-BAM its name.
- Variant Call Format or VCF and its binary representation, Binary Call Format or BCF [21, 40].
- The format originally created for the FASTA set of tools [96], which is nowadays known as the "FASTA format" or simply FASTA.
- FASTQ [37], a simple extension to the FASTA format.
- QSEQ [27], a file format that is output directly by some sequencing instruments.

Hadoop-BAM has both input and output support for all the above formats apart from FASTA, which can only be input.

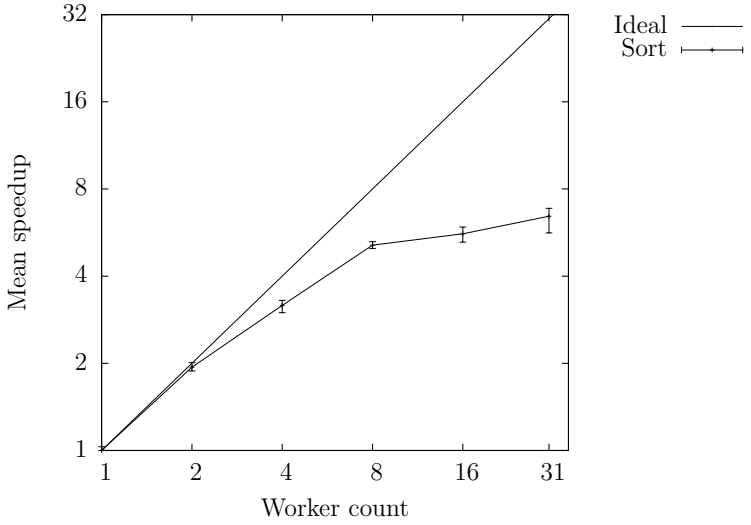


Figure 5.4: Speedup observed when sorting a 50.7 GiB BAM file with Hadoop-BAM.

Command line tools for some tasks commonly performed on SAM and BAM files are also included in Hadoop-BAM, similarly to the SAMtools [70] software package. One such tool can sort and merge SAM and BAM files using MapReduce, which is an important preprocessing step e.g. for visualization [92] and can benefit greatly from parallelization using MapReduce. Testing it on a 50.7 GiB BAM file, we have observed near-linear scaling when using a Hadoop cluster with up to eight slave nodes: see Figure 5.4. The reduced speedup thereafter can be attributed to the relatively small file size leading to quite little data being allocated to each worker node. The machines used in this experiment consisted of the following components each: two six-core Intel Xeon X5650 processors clocked at 2.67 GHz; 48 GB of DDR3-1066 main memory; 4x QDR Infiniband network connections (40 Gbit/s theoretical throughput); and about 830 GB of usable local disk space, striped across two 7200 RPM hard disk drives. Significant comparisons to other software were not performed, as none implement sorting BAM files in HDFS. However, as a simple baseline, the single-threaded `sort` command of SAMtools was tested; operating on local disk on the same hardware, it was over twice as slow as the single-slave Hadoop MapReduce job.

BAM input support is a common desire due to the complexity of the BAM format, as explained in detail in the Section below. Hadoop-BAM is thus often used mainly for its BAM-related functionality. The Seal project donated FASTQ and QSEQ format support to Hadoop-BAM, and later began using Hadoop-BAM for SAM and BAM as well. SeqInCloud's genome analysis pipeline incorporates Hadoop-BAM for BAM input. SAMQA relies on Hadoop-BAM for reading both SAM and BAM. Cloudgene contains Hadoop-BAM's sorting tool among its set of supported applications. ADAM [76] uses Hadoop-BAM to convert FASTA, SAM, and BAM files to the Parquet [93] format, which has been designed for efficient processing in Hadoop.

BAM Splitting Implementation

The primary issue with file format support in Hadoop MapReduce is that files must be *splittable*: disjoint parts of the file must be assignable to different map tasks. Depending on the file format, this can be fairly simple or extremely complicated. In the case of BAM files, the binary encoding alone makes implementing splittability complex, and the layer of compression that is applied on top of the encoding adds some further difficulty. This Section explains the implementation found in Hadoop-BAM.

Hadoop's default file splitting simply divides the input evenly into parts, each part having approximately the same byte length. Due to the nature of the input format, this cannot be relied upon: having a record-oriented file be split along the middle of a record is problematic, since then that record cannot be handled on either side of the split. Typically, it is possible to work around the issue using a simple technique shown in Algorithm 5.1.

Algorithm 5.1. Typical way of reading records from a part of a split file.

```
1: pos ← 0
2: if this is not the first split then
3:     skip input until the beginning of a record
4:     pos ← pos + amount of data skipped
5: end if
6: while pos < end of this split do
7:     r ← record at pos
8:     handle r
9:     pos ← pos + length(r)
10: end while
```

Unfortunately, for BAM files the implementation of line 3 is somewhat complex due to the binary format and the BGZF ("Blocked GNU Zip For-

mat" according to some non-authoritative sources [35, 36]) compression applied on top of it. Two stages of heuristic guesswork are required: one must find, first, the BGZF block containing the position where the split begins; and second, the beginning of the next record, or *alignment*, in BAM.

The first task is easier: BGZF does have, at the start of each block, four bytes with guaranteed values as well as more later on, as can be seen in Table 5.1. Note that the two magic numbers are composed of multiple shorter fields, but they can be considered as units for the purposes of Hadoop-BAM. Recognizing a BGZF block using solely these numbers would unfortunately not work, since nothing prevents a sequence of bytes conforming to these requirements from showing up within the compressed data as well: there is a low probability of treating unrelated data as a BGZF block. Practically speaking, the likelihood of just finding the identifier bits is very low, let alone an otherwise valid-looking block with a correct CRC-32 [97] hash of the uncompressed contents. Even in this ridiculously unlikely situation, the probability of treating the input incorrectly can be further reduced: when the "block" eventually terminates, it is most likely not followed by data that can be again interpreted as a valid BGZF block. Upon noticing this, one can backtrack past the misleading data and search for the next BGZF block.

Table 5.1: The format of one block in the BGZF format. All integers are little-endian.

Description	Type	Value
BGZF block magic number	`uint32`	0x04088b1f
Modification time	`uint32`	
Extra flags	`uint8`	bit 2 is set
Operating system identifier	`uint8`	
Length of extra subfields (XLEN)	`uint16`	at least 6
Extra subfields		
Other extra subfields		
BGZF extra field magic number	`uint16`	0x4342
BGZF extra field length	`uint16`	2
Total block size minus 1 (BSIZE)	`uint16`	
Other extra subfields		
Compressed data	`uint8[`BSIZE − XLEN − 19`]`	
CRC-32 of uncompressed data	`uint32`	
Length of uncompressed data	`uint32`	

The method of determining whether an arbitrary byte sequence appears to be a valid BGZF block, based on the information in Table 5.1, is presented

in Algorithm 5.2. The CRC-32 hash is not checked at this guessing stage, since that would involve unpacking the data and thus is a relatively expensive operation. Instead, the check can be performed later, when the data is actually used.

Algorithm 5.2. Guessing whether a BGZF block starts at the given position.

Require: *bpos*, the position to examine
1: **if** read($bpos, 4$) $\neq$ 0x04088b1f **then** {Incorrect magic number: not a BGZF block.}
2: **return false**
3: **end if**
4: $subpos \leftarrow bpos + 12$ {The offset where the extra subfields begin.}
5: $subend \leftarrow subpos$+read($bpos$+10, 2) {Add the value of the XLEN field.}
6: **while** $subpos < subend$ **do**
7: $magic \leftarrow$ read($subpos, 2$)
8: $slen \leftarrow$ read($subpos + 2, 2$)
9: $subpos \leftarrow subpos + 4 + slen$
10: **if** $magic \neq$ 0x4342$\vee slen \neq 2$ **then** {This is not the BGZF extra field.}
11: **continue**
12: **end if**
13: **while** $subpos < subend$ **do** {Skip over the rest of the extra subfields.}
14: $slen \leftarrow$ read($subpos + 2, 2$)
15: $subpos \leftarrow subpos + slen + 4$
16: **end while**
17: **return** $subpos = subend$ {XLEN must be exact for this to be a valid gzip block.}
18: **end while** {No BGZF extra field found.}
19: **return false**

The second issue, that of finding the next alignment, is somewhat more problematic since BAM records have no clear identifying features. Fortunately, various fields cross-reference each other enough that in practice, some guesswork succeeds.

The following constraints hold on the fields of the BAM record format, displayed in Table 5.2. `n_ref` is not a field in each alignment; it is the number of reference sequences and can be found at the beginning of the BAM file.

1. $\texttt{block\_size} \geq 32 + \texttt{l\_read\_name} + 4 \cdot \texttt{n\_cigar\_op} + (3 \cdot \texttt{l\_seq} + 1)\,/2$
2. The reference IDs are -1 or in the range $[0, \texttt{n\_ref})$:
   $-1 \leq \texttt{refID} < \texttt{n\_ref} \ \wedge \ -1 \leq \texttt{next\_refID} < \texttt{n\_ref}$

Table 5.2: The format of the fields of one alignment in the BAM format. All integers are little-endian. Fields which are not used by the algorithms presented here are marked as *ignored*.

Field name	Description	Type
`block_size`	Record length minus 4	`int32`
`refID`	Reference sequence ID	`int32`
`pos`	0-based co-ordinate	`int32`
`l_read_name`	Length of `read_name`	`uint8`
`mapq`	Mapping quality (*ignored*)	`uint8`
`bin`	Bin number (*ignored*)	`uint16`
`n_cigar_op`	Length of `cigar`	`uint16`
`flag`	Flags bit field (*ignored*)	`uint16`
`l_seq`	Length of decoded `seq`	`int32`
`next_refID`	`refID` of next fragment	`int32`
`next_pos`	`pos` of next fragment	`int32`
`tlen`	Template length (*ignored*)	`int32`
`read_name`	Name, null-terminated	`uint8[l_read_name]`
`cigar`	CIGAR string (*ignored*)	`uint32[n_cigar_op]`
`seq`	Sequence data (*ignored*)	`uint8[(l_seq+1)/2]`
`qual`	Quality (*ignored*)	`uint8[l_seq]`
Auxiliary data until `block_size` is filled (all *ignored*)		
`tag`	Identifier (*ignored*)	`uint8[2]`
`val_type`	Type specifier (*ignored*)	`uint8`
`value`	Value (*ignored*)	depends on `val_type`

3. The positions are -1 or non-negative: $\texttt{pos} \geq -1 \ \wedge \ \texttt{next\_pos} \geq -1$
4. Null-termination of `read_name`: $\texttt{read\_name}[\texttt{l\_read\_name} - 1] = 0$

By using all of these constraints together, one can detect BAM alignments with sufficient accuracy. Pseudocode for this is not given explicitly here, as it is a simple matter of reading integers at constant offsets from each other and performing the comparisons listed above.

Algorithm 5.3 gives a more detailed account of how the splitting can be made to work in all its complexity, with the help of Algorithm 5.2 and an equivalent algorithm for BAM records based on the above constraints.

The bulk of the algorithm is the **while** loop on lines 8–24. Having found a partially validated BAM record, it is fed to a fully featured BAM decoder in order to verify its validity fully (lines 9–11). One can then continue looping through BAM records without any further guessing. The **if** on lines 13–23 handles advancing to the next BGZF block. Note the increment of b: b is the number of BGZF blocks that have been traversed from start to finish.

Algorithm 5.3. Reading BAM records from a part of a split file.

```
 1: pos ← cpos ← 0
 2: if this is not the first split then
 3:   for all pos ∈ apparent BGZF block positions in the split do
 4:     pos_0 ← pos
 5:     for all cpos ∈ apparent BAM record positions in the block at pos do
 6:       cpos_0 ← cpos
 7:       b ← 0
 8:       while pos < end of this split and b < 2 do
 9:         if the data at (pos, cpos) does not form a valid BAM record then
10:           continue at line 5 with pos_0 and next cpos
11:         end if
12:         cpos ← cpos + length(r)
13:         if cpos ≥ block size then
14:           pos ← position of next block after the one at pos
15:           cpos ← 0
16:           if the data at pos does not form a valid BGZF block then
17:             if pos ≥ 2^16 then
18:               input file is invalid or data corruption occurred
19:             end if
20:             continue at line 3 with next pos
21:           end if
22:           b ← b + 1
23:         end if
24:       end while
25:       pos ← pos_0
26:       cpos ← cpos_0
27:       goto 31
28:     end for
29:   end for
30: end if
31: while pos < end of this split do
32:   r ← BAM record at (pos, cpos)
33:   handle r
34:   advance (pos, cpos) by length(r)
35: end while
```

The number two on line 8 is the number of BGZF blocks that should be fully deciphered before accepting that the appropriate location to start reading from has indeed been found. When that occurs, the **while** loop ends and the code proceeds to read records as usual, now that the position to start from is known.

On line 17, the 2^{16} is one past the maximum allowed compressed size of a BGZF block. This limitation can be clearly seen in Table 5.1: it arises due to the fact that the BSIZE field is a 16-bit unsigned integer. Since the input is fully composed only of such blocks, if the algorithm travels past that much space without finding a satisfactory block, something has clearly gone wrong.

BCF Splitting Implementation

The BCF format is highly similar to the BAM format, in that it consists of a binary encoding that is not trivially splittable with a layer of compression on top. One difference is that compression is not mandatory in BCF, but this does not have a significant effect on splitting: it only makes finding the start of the compressed block an optional instead of a mandatory step. Since the compression used in BCF is the same BGZF format as used in BAM files, that step will not be discussed in this Section. Only the features unique to BCF are discussed.

The binary layout of BCF records is shown in Table 5.3. The constraints and redundancies exploited in Hadoop-BAM are listed below. n_c and n_s refer to information found in the BCF header: the length of the chromosome dictionary and the number of samples, respectively.

1. The record length is sensible: $\texttt{l\_shared} + \texttt{l\_indiv} > 32$
2. The chromosome dictionary index is valid: $\texttt{CHROM} \geq 0 \wedge \texttt{CHROM} < n_c$
3. The positions and the two signed counts are nonnegative: $\texttt{POS} \geq 0 \wedge \texttt{n\_info} \geq 0 \wedge \texttt{n\_allele} \geq 0$
4. The sample count matches to the value in the header: $\texttt{n\_sample} = n_s$
5. The ID field should have a sensible type encoding and a reasonable length. Here i_0 and i_1 refer to the first two bytes of ID; l refers to the decoded length of the string, whose encoded size depends on i_0 and i_1; and & is a bitwise AND. The two constraints are as follows:
   (a) i_0 & 0x0f = 0x07

Table 5.3: The format of the fields of one record in BCF. All integers are little-endian and floating point values are in the IEEE 754 [62] format. `int` (when not followed by a bit width), `str`, and `vec` refer to the custom *typed* encodings used in BCF and not detailed here. Fields which are not used by the algorithms presented here are marked as *ignored*. Note that the "BCF2 site information encoding" table in the specification [21] has `QUAL` in an incorrect position.

Field name	Description	Type
`l_shared`	Length from `CHROM` to end of `INFO`	`uint32`
`l_indiv`	Total length of genotype fields	`uint32`
`CHROM`	Chromosome dictionary index	`int32`
`POS`	0-based co-ordinate	`int32`
`rlen`	Projected record length (*ignored*)	`int32`
`QUAL`	Quality (*ignored*)	`float32`
`n_info`	Number of `INFO` pairs	`int16`
`n_allele`	Number of `REF+ALT` records	`int16`
`n_sample`	Number of values in each genotype field	`uint24`
`n_fmt`	Number of genotype fields (*ignored*)	`uint8`
`ID`	Identifier(s)	`str`
`REF+ALT`	Sequence strings (*ignored*)	`str[n_allele]`
`FILTER`	Filter dictionary indices (*ignored*)	`vec`
`INFO`	Additional information (*ignored*)	`vec[n_info]`
	`n_fmt` genotype fields (all *ignored*)	
`fmt_key`	Identifier (*ignored*)	`int`
`fmt_type`	Type specifier (*ignored*)	`uint8`, optional `int`
`fmt_values`	`n_sample` values (*ignored*)	depends on `fmt_type`

(b) If $i_0 \,\&\, 0\text{xf0} = 0\text{xf0}$, then:

$$(i_1 \,\&\, 0\text{x0f}) \in [1, 3] \\ \wedge\, l \geq 15 \wedge l \leq \texttt{l\_shared} - (32 + \texttt{n\_allele} + 2 \cdot \texttt{n\_info})$$

Based on the above it is possible to find candidate locations: positions where a BCF record is likely to begin. A decoding test is then performed starting at each such location; once a certain number of records have been successfully read, it is assumed that the location was valid and should be used for the actual computation. An error at any point during decoding means that the next position should be tried. Thus the algorithm for BCF is essentially equivalent to Algorithm 5.3, with the two differences being that finding BGZF

blocks is optional and, of course, BCF records are detected and validated instead of BAM records.

5.4.2 SeqPig

While Hadoop-BAM gives developers the opportunity to create custom Hadoop MapReduce applications for sequencing data with control over every aspect of processing, SeqPig [108, 109, 110] is a high-level interface based on Pig. With SeqPig, as long as the application can be adequately described in Pig Latin, development is simpler and does not require familiarity with MapReduce or Java.

SeqPig provides almost the same file format functionality as current Hadoop-BAM, lacking only VCF and BCF: all of SAM and BAM, FASTA (read-only), FASTQ, and QSEQ are supported. All data and metadata in these formats can be loaded for manipulation in Pig Latin. In addition, SeqPig includes user-defined functions for several useful operations specific to sequencing data. Thanks to Pig, all processing can take place scalably using Hadoop MapReduce.

The unrelated BioPig project naturally shares the advantages of Pig with SeqPig. The differences between the two lie in their provided bioinformatics-specific functionality. In terms of file formats, BioPig supports only FASTA and FASTQ—although, unlike SeqPig, it has output support for FASTA. Otherwise, the sets of user-defined functions provided by SeqPig and BioPig are intended for very different concerns in sequencing data analysis. For this reason, one may wish to use SeqPig and BioPig together, and due to Pig's simple data model, this is highly straightforward.

As a publication that describes SeqPig more fully is currently in submission [109], we regretfully cannot provide further details of SeqPig here.

5.5 Closing Remarks

Traditional approaches to data analytics are insufficient when dealing with Big Data. Methods that are not specifically designed with Big Data in mind do not scale in any of a number of ways. Most approaches simply do not function on warehouse-scale computers, being designed only for desktop-like systems. Some approaches become prohibitively slow when the data volumes involved grow too large, while others cannot operate effectively at the warehouse scale e.g. due to not being able to work around the highly probable hardware failures. Solving a Big Data problem requires awareness of many such issues.

Part of the problem is the need for a warehouse-scale computer, which are expensive both to construct and to keep running. Existing traditional clusters likely do not consist of nodes that have enough local hard disk drives for Hadoop to work efficiently. For example, Cloudera recommends from 0.5 to 1.5 drives per CPU core [73]. They also tend to have computers containing relatively expensive "server-grade" hardware such as hardware RAID controllers [20]. Therefore the price/performance advantage of Hadoop storage over traditional NAS/SAN storage, made possible by HDFS's software-level handling of fault tolerance, is not realized. Furthermore, bioinformaticians often have mostly idle clusters due to fluctuating usage patterns [115]. Nowadays, it is fortunately possible to provision computational resources on demand via services such as Amazon Elastic Compute Cloud [3], Windows Azure [131], Google App Engine [53]. This form of so-called "cloud computing" may be the best option when a complete warehouse-scale computer is not affordable or when its resources would momentarily not be fully utilized. However, transferring data sets to remote clusters can impose additional costs.

Once one has the hardware, one requires the appropriate software to make it useful and turn it into an actual solution to a Big Data problem. Hadoop provides a framework on which such solutions can be easily constructed. While originally intended for Web search data processing at Nutch and later Yahoo!, it has been shown to be an appropriate "hammer" for many other Big Data "nails". Bioinformatics, thanks to high-throughput sequencing in particular, has many Big Data problems for which Hadoop is a good fit, leading to the development of several applications capable of dealing with them. We contributed with Hadoop-BAM, originally primarily to enable scalable processing of the complicated BAM format, and later to consolidate bioinformatics-related file format support in one library.

Of course, not everyone is a programmer capable of using Hadoop to solve Big Data problems: analysts should not be expected to write a new Hadoop-using program every time they wish to query their data sets in a new way. Thus, more accessible approaches are needed. For Hadoop, they are provided by systems such as Pig and Hive, but the domain-specific part must still be created by software developers. SeqPig is our offering to those who wish to analyse Big Data sets of sequencing data in a relatively high-level language.

Further challenges remain in the Big Data world, however. Major ones are interactivity and real-time analysis, as Hadoop's primarily computational platform, MapReduce, is not well suited to latency-sensitive work such as interactive exploratory querying of data sets. Impala and Spark (and its high-level Hive-compatible companion, Shark) are two solutions, neither of which has yet been applied to bioinformatics. Interactivity is becoming more

and more desirable in sequencing data analysis [33], so the lack of appropriate Big Data tooling is a glaring omission. Plenty of research remains to be done in this field: Big Data is here to stay and data sets are only growing larger.

Acknowledgements

The funding of the Cloud Software Program of Digile funded by the Finnish Funding Agency for Technology and Innovation Tekes, and the Academy of Finland (project 139402) is gratefully acknowledged.

References

[1] A. S. Aiyer et al. "Storage Infrastructure Behind Facebook Messages: Using HBase at Scale". In: *IEEE Data Eng. Bull.* 35.2 (2012), pp. 4–13. URL: http://sites.computer.org/debull/A12june/facebook.pdf.

[2] A. Alexandrov et al. "MapReduce and PACT - Comparing Data Parallel Programming Models". In: *Proceedings of the 14th Conference on Database Systems for Business, Technology, and Web (BTW)*. BTW 2011. Kaiserslautern, Germany: GI, 2011, pp. 25–44. ISBN: 978-3-88579-274-1. URL: https://stratosphere.eu/sites/default/files/papers/ComparingMapReduceAndPACTs_11.pdf.

[3] *Amazon Elastic Compute Cloud*. Amazon Web Services, Inc. URL: https://aws.amazon.com/ec2/.

[4] *Apache Drill*. The Apache Software Foundation. URL: https://incubator.apache.org/drill/.

[5] *Apache Hadoop*. The Apache Software Foundation. URL: https://hadoop.apache.org.

[6] *Apache Hadoop NextGen MapReduce (YARN)*. Version 2.1.0-beta. 2013. URL: https://hadoop.apache.org/docs/r2.1.0-beta/hadoop-yarn/hadoop-yarn-site/YARN.html.

[7] *Apache HBase*. The Apache Software Foundation. URL: https://hbase.apache.org.

[8] *Apache Hive*. The Apache Software Foundation. URL: https://hive.apache.org.

[9] *Apache Mesos*. The Apache Software Foundation. URL: https://mesos.apache.org.

[10] *Apache Nutch*. The Apache Software Foundation. URL: https://nutch.apache.org.

[11] *Apache Pig*. The Apache Software Foundation. URL: https://pig.apache.org.

[12] *Apache Tez*. Hortonworks Inc. URL: https://hortonworks.com/hadoop/tez/.

[13] *Apache ZooKeeper*. The Apache Software Foundation. URL: https://zookeeper.apache.org.

[14] E. de Araujo Macedo et al. "Hybrid MPI/OpenMP Strategy for Biological Multiple Sequence Alignment with DIALIGN-TX in Heterogeneous Multicore Clusters". In: *IPDPS Workshops*. IEEE, 2011, pp. 418–425. ISBN: 978-1-61284-425-1. DOI: 10.1109/IPDPS.2011.169.

[15] J. L. Armstrong. "The Development of Erlang". In: *ICFP*. Ed. by S. L. P. Jones, M. Tofte, and A. M. Berman. ACM, 1997, pp. 196–203. ISBN: 978-0-89791-918-0. DOI: 10.1145/258948.258967.

[16] A. Auradkar et al. "Data Infrastructure at LinkedIn". In: *ICDE*. Ed. by A. Kementsietsidis and M. A. V. Salles. IEEE Computer Society, 2012, pp. 1370–1381. ISBN: 978-0-7685-4747-3. DOI: 10.1109/ICDE.2012.147. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6226952.

[17] G. V. der Auwera. *Overview of Queue*. Broad Institute. URL: https://www.broadinstitute.org/gatk/guide/article?id=1306.

[18] L. A. Barroso, J. Clidaras, and U. Hlzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Second edition. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, July 2013. DOI: 10.2200/S00516ED2V01Y201306CAC024.

[19] L. A. Barroso, J. Dean, and U. Hölzle. "Web Search for a Planet: The Google Cluster Architecture". In: *IEEE Micro* 23.2 (2003), pp. 22–28. DOI: 10.1109/MM.2003.1196112.

[20] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009. DOI: 10.2200/S00193ED1V01Y200905CAC006.

[21] *BCF (Binary VCF) version 2*. Tech. rep. Version 2.1. URL: http://www.1000genomes.org/wiki/analysis/variant-call-format/bcf-binary-vcf-version-2.

[22] B. H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (1970), pp. 422–426. DOI: 10.1145/362686.362692.

[23] "MPI: A Message Passing Interface". In: *SC*. Ed. by B. Borchers and D. Crawford. The MPI Forum. IEEE Computer Society / ACM, 1993, pp. 878–883. ISBN: 978-0-8186-4340-8. DOI: 10.1145/169627.169855.

[24] D. Borthakur et al. "Apache Hadoop Goes Realtime at Facebook". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. SIGMOD '11. Athens, Greece: ACM, 2011, pp. 1071–1080. ISBN: 978-1-4503-0661-4. DOI: 10.1145/1989323.1989438.

[25] B. Calder et al. "Windows Azure Storage: a Highly Available Cloud Storage Service with Strong Consistency". In: *SOSP*. Ed. by T. Wobber and P. Druschel. ACM, 2011, pp. 143–157. ISBN: 978-1-4503-0977-6. DOI: 10.1145/2043556.2043571.

[26] S. K. Card, G. G. Robertson, and J. D. Mackinlay. "The information visualizer, an information workspace". In: *CHI*. Ed. by S. P. Robertson, G. M. Olson, and J. S. Olson. ACM, 1991, pp. 181–186. ISBN: 978-0-89791-383-6. DOI: 10.1145/108844.108874.

[27] *CASAVA v1.8 User Guide*. Illumina, Inc. 2011. URL: http://biowulf.nih.gov/apps/CASAVA_UG_15011196B.pdf.

[28] D. D. Chamberlin and R. F. Boyce. "SEQUEL: A Structured English Query Language". In: *SIGMOD Workshop, Vol. 1*. Ed. by R. Rustin. ACM, 1974, pp. 249–264. DOI: 10.1145/800296.811515.

[29] C. Y. Chan and Y. E. Ioannidis. "Bitmap Index Design and Evaluation". In: *SIGMOD Conference*. Ed. by L. M. Haas and A. Tiwary. ACM Press, 1998, pp. 355–366. ISBN: 978-0-89791-995-1. DOI: 10.1145/276304.276336.

[30] F. Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *OSDI*. USENIX Association, 2006, pp. 205–218.

[31] Y.-J. Chang et al. "*De Novo* Assembly of High-Throughput Sequencing Data with Cloud Computing and New Operations on String Graphs". In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. 2012, pp. 155–161. DOI: 10.1109/CLOUD.2012.123.

[32] B. Chattopadhyay et al. "Tenzing. A SQL Implementation On The MapReduce Framework". In: *PVLDB* 4.12 (2011), pp. 1318–1327. URL: http://www.vldb.org/pvldb/vol4/p1318-chattopadhyay.pdf.

[33] Y. Chen, S. Alspaugh, and R. H. Katz. "Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads". In: *PVLDB* 5.12 (2012), pp. 1802–1813. URL: http://vldb.org/pvldb/vol5/p1802_yanpeichen_vldb2012.pdf.

[34] *Cloudera, Inc.* URL: http://www.cloudera.com.

[35] R. Cnovas and A. Moffat. "Practical Compression for Multi-Alignment Genomic Files". In: *Computer Science 2013 (ACSC 2013)*. Ed. by B. Thomas. Vol. 135. CRPIT. Adelaide, Australia: ACS, 2013, pp. 51–60. URL: https://crpit.com/confpapers/CRPITV135Canovas.pdf.

[36] P. Cock. "BGZF - Blocked, Bigger & Better GZIP!" In: *Blasted Bioinformatics!?* (Nov. 8, 2011). URL: http://blastedbio.blogspot.com/2011/11/bgzf-blocked-bigger-better-gzip.html.

[37] P. J. A. Cock et al. "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants". In: *Nucleic Acids Research* 38.6 (2010), pp. 1767–1771. DOI: 10.1093/nar/gkp1137.

[38] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (1970), pp. 377–387. DOI: 10.1145/362384.362685.

[39] B. F. Cooper et al. "Benchmarking Cloud Serving Systems with YCSB". In: *SoCC*. Ed. by J. M. Hellerstein, S. Chaudhuri, and M. Rosenblum. ACM, 2010, pp. 143–154. ISBN: 978-1-4503-0036-0. DOI: 10.1145/1807128.1807152.

[40] P. Danecek et al. "The variant call format and VCFtools". In: *Bioinformatics* 27.15 (2011), pp. 2156–2158. DOI: 10.1093/bioinformatics/btr330.

[41] A. E. Darling, L. Carey, and W. chun Feng. "The Design, Implementation, and Evaluation of mpiBLAST". In: ClusterWorld. (2003). 2003.

[42] C. J. Date. *Date on Database: Writings 2000–2006*. Apress, 2006. ISBN: 978-1-59059-746-0.

[43] J. Dean. "Designs, Lessons and Advice from Building Large Distributed Systems". 2009. URL: https://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf.

[44] J. Dean and L. A. Barroso. "The Tail at Scale". In: *Commun. ACM* 56.2 (2013), pp. 74–80. DOI: 10.1145/2408776.2408794.

[45] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI*. 2004, pp. 137–150.

[46] R. Dennard et al. "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions". In: *Solid-State Circuits Society Newsletter, IEEE* 12.1 (2007), pp. 38–50. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2007.4785543.

[47] *Facebook*. URL: https://www.facebook.com.

[48] A. Gates et al. "Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience". In: *PVLDB* 2.2 (2009), pp. 1414–1425. URL: http://www.vldb.org/pvldb/2/vldb09-1074.pdf.

[49] L. George. *HBase: The Definitive Guide. Random Access to Your Planet-Size Data*. O'Reilly, 2011, pp. I–XXVII, 1–522. ISBN: 978-1-449-39610-7.

[50] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google file system". In: *SOSP*. Ed. by M. L. Scott and L. L. Peterson. ACM, 2003, pp. 29–43. ISBN: 1-58113-757-5.

[51] D. Giampaolo. *Practical File System Design with the Be File System*. Morgan Kaufmann, 1999. ISBN: 978-1-55860-497-1. URL: http://www.nobius.org/~dbg/practical-file-system-design.pdf.

[52] *Google*. URL: https://www.google.com.

[53] *Google App Engine*. URL: https://developers.google.com/appengine/.

[54] J. Gosling et al. *The Java Language Specification, Java SE 7 Edition*. Addison-Wesley Professional, Feb. 2013. ISBN: 978-0-1332-6022-9.

[55] J. R. Groff, P. N. Weinberg, and A. J. Oppel. *SQL: The Complete Reference*. Third Edition. McGraw-Hill Osborne Media, Aug. 2009. ISBN: 978-0-0715-9255-0.

[56] *Hadoop-BAM*. URL: http://sf.net/projects/hadoop-bam/.

[57] *Hadoop YARN*. Hortonworks Inc. URL: https://hortonworks.com/hadoop/yarn/.

[58] *Hbase/PoweredBy*. URL: https://wiki.apache.org/hadoop/Hbase/PoweredBy.

[59] B. Hindman et al. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center". In: *NSDI*. Boston, MA, USA: USENIX Association, Apr. 2011, pp. 295–308. ISBN: 978-931971-84-3. URL: https://www.usenix.org/legacy/events/nsdi11/tech/full_papers/Hindman_new.pdf.

[60] *[HIVE-1402] Add parallel ORDER BY to Hive*. Apache Software Foundation, 2010. URL: https://issues.apache.org/jira/browse/HIVE-1402.

[61] *Hortonworks Inc.* URL: http://www.hortonworks.com.

[62] "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2008* (2008), pp. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.

[63] *Introducing Impala*. Cloudera, Inc. URL: http://cloudera.com/impala/.

[64] F. Junqueira, B. Reed, and M. Serafini. "Zab: High-performance broadcast for primary-backup systems". In: *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. 2011, pp. 245–256. DOI: 10.1109/DSN.2011.5958223.

[65] D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973. ISBN: 978-0-201-03803-3.

[66] M. Kryder and C. S. Kim. "After Hard Drives What Comes Next?" In: *Magnetics, IEEE Transactions on* 45.10 (2009), pp. 3406–3413. ISSN: 0018-9464. DOI: 10.1109/TMAG.2009.2024163.

[67] L. Lamport. "The Part-Time Parliament". In: *ACM Trans. Comput. Syst.* 16.2 (1998), pp. 133–169.

[68] B. Langmead, K. Hansen, and J. Leek. "Cloud-scale RNA-sequencing differential expression analysis with Myrna". In: *Genome Biology* 11.8 (2010), R83. ISSN: 1465-6906. DOI: 10.1186/gb-2010-11-8-r83. PMID: 20701754.

[69] B. Langmead et al. "Searching for SNPs with cloud computing". In: *Genome Biology* 10.11 (2009), R134. ISSN: 1465-6906. DOI: 10.1186/gb-2009-10-11-r134. PMID: 19930550.

[70] H. Li et al. "The Sequence Alignment/Map format and SAMtools". In: *Bioinformatics* 25.16 (2009), pp. 2078–2079. DOI: 10.1093/bioinformatics/btp352.

[71] J. Lin, D. Ryaboy, and K. Weil. "Full-text Indexing for Optimizing Selection Operations in Large-Scale Data Analytics". In: *Proceedings of the second international workshop on MapReduce and its applications*. MapReduce '11. San Jose, California, USA: ACM, 2011, pp. 59–66. ISBN: 978-1-4503-0700-0. DOI: 10.1145/1996092.1996105.

[72] *LinkedIn*. URL: https://www.linkedin.com.

[73] A. Loddengaard. *Clouderas Support Team Shares Some Basic Hardware Recommendations*. Mar. 2010. URL: http://blog.cloudera.com/blog/2010/03/clouderas-support-team-shares-some-basic-hardware-recommendations/.

[74] *MapR Technologies, Inc.* URL: http://www.mapr.com.

[75] V. Marx. "Biology: The big challenges of big data". In: *Nature* 498.7453 (June 2013). Technology Feature, pp. 255–260. ISSN: 0028-0836. DOI: 10.1038/498255a.

[76] M. Massie. *ADAM: Datastore Alignment Map*. URL: https://github.com/massie/adam/.

[77] A. M. Matsunaga, M. O. Tsugawa, and J. A. B. Fortes. "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications". In: *eScience*. IEEE Computer Society, 2008, pp. 222–229. DOI: 10.1109/eScience.2008.62.

[78] A. McKenna et al. "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data". In: *Genome Research* 20.9 (2010), pp. 1297–1303. DOI: 10.1101/gr.107524.110.

[79] M. K. McKusick and S. Quinlan. "GFS: Evolution on Fast-forward". In: *Queue* 7.7 (Aug. 2009), 10:10–10:20. ISSN: 1542-7730. DOI: 10.1145/1594204.1594206.

[80] S. Melnik et al. "Dremel: Interactive Analysis of Web-Scale Datasets". In: *PVLDB* 3.1 (2010), pp. 330–339. URL: http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R29.pdf.

[81] N. M. Mohamed, H. Lin, and W.-c. Feng. "Accelerating Data-Intensive Genome Analysis in the Cloud". In: *5th International Conference on Bioinformatics and Computational Biology (BICoB)*. Honolulu, Hawaii, USA, Mar. 2013. URL: http://synergy.cs.vt.edu/pubs/papers/nabeel-bicob13-genome-analysis-cloud.pdf.

[82] A. Montañola, C. Roig, and P. Hernández. "Pairwise Sequence Alignment Method for Distributed Shared Memory Systems". In: *PDP*. IEEE Computer Society, 2013, pp. 432–436. ISBN: 978-1-4673-5321-2. DOI: 10.1109/PDP.2013.69.

[83] G. Moore. "Progress in digital integrated electronics". In: *Electron Devices Meeting, 1975 International*. Vol. 21. 1975, pp. 11–13.

[84] G. E. Moore. "Cramming more components onto integrated circuits". In: *Electronics* 38.8 (Apr. 1965).

[85] A. Murthy. "Apache Hadoop YARN Background and an Overview". In: (Aug. 7, 2012). URL: https://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/.

[86] T. Nguyen, W. Shi, and D. Ruden. "CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping". In: *BMC Research Notes* 4.1 (2011), p. 171. ISSN: 1756-0500. DOI: 10.1186/1756-0500-4-171. PMID: 21645377. URL: http://www.biomedcentral.com/1756-0500/4/171.

[87] M. Niemenmaa. *Interactivity for Big Data: Preprocessing genomic data with MapReduce*. Bachelor's Thesis. May 4, 2011. URL: http://users.ics.aalto.fi/mniemenm/online-papers/niemenmaa-thesis-b.pdf.

[88] M. Niemenmaa et al. "Hadoop-BAM: directly manipulating next generation sequencing data in the cloud". In: *Bioinformatics* 28.6 (2012), pp. 876–877. DOI: 10.1093/bioinformatics/bts054.

[89] H. Nordberg et al. "BioPig: A Hadoop-based Analytic Toolkit for Large-Scale Sequence Data". In: *Bioinformatics* (2013). DOI: 10.1093/bioinformatics/btt528.

[90] B. D. O'Connor, B. Merriman, and S. F. Nelson. "SeqWare Query Engine: storing and searching sequence data in the cloud". In: *BMC Bioinformatics* 11.S-12 (2010), S2. DOI: 10.1186/1471-2105-11-S12-S2.

[91] C. Olston et al. "Pig latin: A not-so-foreign language for data processing". In: *SIGMOD Conference*. Ed. by J. T.-L. Wang. ACM, 2008, pp. 1099–1110. ISBN: 978-1-60558-102-6.

[92] S. Pabinger et al. "A survey of tools for variant analysis of next-generation genome sequencing data". In: *Briefings in Bioinformatics* (2013). DOI: 10.1093/bib/bbs086.

[93] *Parquet: Columnar Storage for Hadoop*. URL: http://parquet.io.

[94] D. A. Patterson. "Latency lags bandwith". In: *Commun. ACM* 47.10 (2004), pp. 71–75.

[95] A. Pavlo et al. "A comparison of approaches to large-scale data analysis". In: *SIGMOD Conference*. Ed. by U. Çetintemel et al. ACM, 2009, pp. 165–178. ISBN: 978-1-60558-551-2.

[96] W. R. Pearson and D. J. Lipman. "Improved tools for biological sequence comparison". In: *Proc. Natl. Acad. Sci. USA* 85.8 (Apr. 1988), pp. 2444–2448. PMID: 3162770.

[97] W. W. Peterson and D. T. Brown. "Cyclic Codes for Error Detection". In: *Proceedings of the IRE* 49.1 (1961), pp. 228–235. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1961.287814.

[98] L. Pireddu, S. Leo, and G. Zanetti. "MapReducing a Genomic Sequencing Workflow". In: *Proceedings of the second international workshop on MapReduce and its applications*. MapReduce '11. San Jose, California, USA: ACM, 2011, pp. 67–74. ISBN: 978-1-4503-0700-0. DOI: 10.1145/1996092.1996106.

[99] L. Pireddu, S. Leo, and G. Zanetti. "SEAL: a distributed short read mapping and duplicate removal tool". In: *Bioinformatics* 27.15 (2011), pp. 2159–2160. DOI: 10.1093/bioinformatics/btr325.

[100] B. Pratt et al. "MR-Tandem: parallel X!Tandem using Hadoop MapReduce on Amazon Web Services". In: *Bioinformatics* 28.1 (2012), pp. 136–137. DOI: 10.1093/bioinformatics/btr615.

[101] B. Reed. "Hadoop @ Yahoo! an admin's perspective". 2010. URL: http://www.cs.duke.edu/smdb10/_files/toc_data/SMDB/panel/reed.pdf.

[102] C. Reynolds. "As We May Communicate". In: *SIGCHI Bull.* 30.3 (July 1998), pp. 40–44. ISSN: 0736-6906. DOI: 10.1145/565711.565714.

[103] S. Rezaei, M. Monwar, and J. Bai. "Performance Comparison of MPI-Based Parallel Multiple Sequence Alignment Algorithm Using Single and Multiple Guide Trees". In: *Cognitive Informatics, 2006. ICCI 2006. 5th IEEE International Conference on*. Vol. 1. 2006, pp. 595–600. DOI: 10.1109/COGINF.2006.365552.

[104] T. Robinson et al. "SAMQA: error classification and validation of high-throughput sequenced read data". In: *BMC Genomics* 12.1 (2011), p. 419. ISSN: 1471-2164. DOI: 10.1186/1471-2164-12-419. PMID: 21851633.

[105] M. Schatz et al. *Contrail: Assembly of Large Genomes using Cloud Computing*. URL: http://sourceforge.net/apps/mediawiki/contrail-bio/.

[106] M. C. Schatz. "CloudBurst: highly sensitive read mapping with MapReduce". In: *Bioinformatics* 25.11 (2009), pp. 1363–1369. DOI: 10.1093/bioinformatics/btp236.

[107] S. Schönherr et al. "Cloudgene: A graphical execution platform for MapReduce programs on private and public clouds". In: *BMC Bioinformatics* 13 (2012), p. 200. DOI: 10.1186/1471-2105-13-200.

[108] A. Schumacher et al. "Scripting for large-scale sequencing based on Hadoop". In: *EMBnet.journal* 19.A (2013). URL: http://journal.embnet.org/index.php/embnetjournal/article/view/628.

[109] A. Schumacher et al. "SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop". 2013. Submitted.

[110] *SeqPig Manual*. URL: http://seqpig.sf.net.

[111] *Sequence Alignment/Map Format Specification*. Tech. rep. Version 321f786. The SAM/BAM Format Specification Working Group, May 29, 2013. URL: http://samtools.sourceforge.net/SAMv1.pdf.

[112] W. Shang, B. Adams, and A. E. Hassan. "Using Pig as a data preparation language for large-scale mining software repositories studies: An experience report". In: *Journal of Systems and Software* 85.10 (2012), pp. 2195–2204. DOI: 10.1016/j.jss.2011.07.034.

[113] *SQL-in-Hadoop : The Stinger Initiative*. Hortonworks Inc. URL: https://hortonworks.com/stinger/.

[114] I. O. for Standardization. *ISO/IEC 9075:1992. Information technology -- Database languages -- SQL*. Geneva, Switzerland, 1992. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=16663.

[115] L. Stein. "The case for cloud computing in genome informatics". In: *Genome Biology* 11.5 (2010), p. 207. ISSN: 1465-6906. DOI: 10.1186/gb-2010-11-5-207. PMID: 20441614.

[116] B. Stroustrup. *The C++ Programming Language*. Fourth edition. Addison-Wesley Professional, May 2013. ISBN: 978-0-3215-6384-2.

[117] V. S. Sunderam. "PVM: A Framework for Parallel Distributed Computing". In: *Concurrency - Practice and Experience* 2.4 (1990), pp. 315–339. DOI: 10.1002/cpe.4330020404.

[118] R. C. Taylor. "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics". In: *BMC Bioinformatics* 11.S-12 (2010), S1. DOI: 10.1186/1471-2105-11-S12-S1.

[119] *Excerpts from A Conversation with Gordon Moore: Moores Law*. Intel Corporation, 2005. URL: ftp://download.intel.com/museum/Moores_Law/Video-Transcripts/Excepts_A_Conversation_with_Gordon_Moore.pdf.

[120] *Tez*. The Apache Software Foundation. URL: http://tez.incubator.apache.org/.

[121] *The Apache HBase Reference Guide*. The Apache Software Foundation. URL: https://hbase.apache.org/book.html.

[122] *The Apache Software Foundation*. URL: https://www.apache.org.

[123] A. Thusoo et al. "Data Warehousing and Analytics Infrastructure at Facebook". In: *SIGMOD Conference*. Ed. by A. K. Elmagarmid and D. Agrawal. ACM, 2010, pp. 1013–1020. ISBN: 978-1-4503-0032-2. DOI: 10.1145/1807167.1807278.

[124] A. Thusoo et al. "Hive – A Petabyte Scale Data Warehouse Using Hadoop". In: *ICDE*. Ed. by F. Li et al. IEEE, 2010, pp. 996–1005. ISBN: 978-1-4244-5444-0. DOI: 10.1109/ICDE.2010.5447738.

[125] A. Thusoo et al. "Hive - A Warehousing Solution Over a Map-Reduce Framework". In: *PVLDB* 2.2 (2009), pp. 1626–1629. URL: http://www.vldb.org/pvldb/2/vldb09-938.pdf.

[126] *Twitter*. URL: https://twitter.com.

[127] C. Walter. "Kryder's Law". In: *Scientific American* 293 (2 2005), pp. 32–33. DOI: 10.1038/scientificamerican0805-32.

[128] J. K. Waters. "Apache Hadoop Community Promotes YARN -- But Don't Call it MapReduce 2". In: *WatersWorks* (Aug. 15, 2012). URL: http://adtmag.com/blogs/watersworks/2012/08/apache-yarn-promotion.aspx.

[129] K. Weil. "Hadoop at Twitter". 2010. URL: http://www.slideshare.net/kevinweil/hadoop-at-twitter-hadoop-summit-2010.

[130] T. White. *Hadoop: The Definitive Guide. MapReduce for the Cloud*. O'Reilly, 2009, pp. I–XIX, 1–501. ISBN: 978-0-596-52197-4.

[131] *Windows Azure*. Microsoft Corporation. URL: www.windowsazure.com.

[132] R. Xin et al. “Shark: SQL and Rich Analytics at Scale”. In: *CoRR* abs/1211.6176 (2012). arXiv: 1211.6176 [cs.DB].

[133] *Yahoo!* URL: http://www.yahoo.com.

[134] M. Zaharia et al. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”. In: *NSDI*. San Jose, CA, USA: USENIX Association, Apr. 2012, pp. 15–28. ISBN: 978-931971-92-8. URL: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf.

6 Performance Testing in the Cloud Using MBPeT

Fredrik Abbors, Tanwir Ahmad, Dragos Truscan, and Ivan Porres
Department of Information Technologies
Åbo Akademi University, Turku, Finland
Email: {fredrik.abbors, tanwir.ahmed, dragos.truscan, ivan.porres}@abo.fi

Abstract–We present a model-based performance testing approach using the MBPeT tool. We use of probabilistic timed automata to model the user profiles and to generate synthetic workload. The MBPeT generates the load in a distributed fashion and applies it in real-time to the system under test, while measuring several key performance indicators, such as response time, throughput, error rate, etc. At the end of the test session, a detailed test report is provided. MBPeT has a distributed architecture and supports load generation distributed over multiple machines. New generation nodes are allocated dynamically during load generation. In this book chapter, we will present the MBPeT tool, its architecture, and demonstrate its applicability with a set of experiments on a case study. We also show that using abstract models for describing the user profiles allows us quickly experiment different load mixes and detect worst case scenarios.

Keywords-Performance testing, model-based testing, MBPeT, cloud.

6.1 Introduction

Software testing is the process of identifying incorrect behavior of a system, also known as revealing defects. Uncovering these defects, typically, consists of running a batch of software tests (*test suite*) against the software itself. In some sense, a second software artefact is built to test the primary one. This is normally referred to as *functional testing*. A software test compares the actual output of the system with the expected output for a particular

known input. If the actual output is the same as the expected output the test passes, otherwise a test fails and a defect is found. Software testing is also the means to assess the quality of a software product. The fewer the defects found during testing, the higher the quality is of that software product. However, not all software defects are related to functionality. Some systems may stop functioning or may prevent other users to access the system simple because the system is under a heavy workload with which it cannot cope. Performance testing is the means of detecting such errors.

Performance testing is the process of determining how a software system performs in terms of responsiveness and stability under a particular workload. The purpose of the workload is that it should match the expected workload (the load that normal users put on the system when using it) as closely as possible. This can be achieved by running a series of tests in parallel, but instead of focusing on the right output the focus is shifted towards measuring non-functional aspects, i.e. the time between input and output (response time) or number of requests processed in a second (throughput).

Traditionally, performance testing has been conducted by running a number of predefined scenarios (or scripts) in parallel. One drawback to this approach is that real users do not behave as static scripts. This can also lead to certain paths in the system being left untested or that certain caching mechanisms in the system kick in due the repetitiveness of the test scripts.

Software testing can be extremely time consuming and costly. In 2005, Caper Jones - chief scientist of Software Productivity Research in Massachusetts - estimated that as much as 60 percent of the software work in the United States was related to detecting and fixing defects [1]. Another drawback is that software testing, as well as performance testing, involves tedious manual work when creating test cases. A software system typically undergoes a lot of changes during its lifetime. Whenever a piece of code is changed, a test has to be updated or created to show that the change did not break any existing functionality or introduce any new defects. This adds more time and cost to testing. In the case of performance testing this implies that one has to be able to benchmark quickly and effectively to check if the performance of the system is affected by the change of the code.

Research effort have be put into solving this dilemma. One of the most promising techniques is Model-Based Testing (MBT). In MBT, the central artefact is a system model. The idea is that the model represents the behavior or the use of the system. Tests are then automatically generated form the model. In MBT the focus has shifted from manually creating tests to maintaining a model that represents the behavior of the system. Due to the fact that tests are automatically generated from a model, MBT copes better with changing requirements and code than traditional testing. Research has

shown that MBT could reduce the total testing costs with 15 percent [8]. MBT has mostly been targeted towards functional testing, however, there exist a few tools that utilizes the power of MBT in the domain of performance testing. In our research we make use of the advantages of MBT in our performance testing approach.

MBPeT is a Python-based tool for performance testing. Load is generated from *probabilistic timed automata* (PTA) models describing the behavior of groups of virtual users. The models are then executed in parallel to get a semi-random workload mix. The abstract PTA models are easy to create and update, facilitating quick iteration cycles. During the load generation phase, the tool also monitors different *key performance indicators* (KPIs) such as response times, throughput, memory, CPU, disk, etc. The MBPeT tool has a distributed architecture where one master node controls several slave node or load generator. This facilitates deployment to a cloud environment. Besides monitoring, the tool also produces a performance test report at the end of the test. The report contains information about the monitored KPIs, such as response times, throughput etc, but also graphs showing how CPU, memory, disk, network utilization varied during a performance test session.

The rest of the report is structured as follows: we briefly enumerate several related works in the following section. Then, is Section 6.3, we briefly describe the load generation process. In Section 6.4, we give an overview of the architecture of the tool. In Section 6.5, we describe how the workload models are created and discuss the probabilistic timed automata formalism. In Section 6.6, we discuss the performance testing process in more detail. In Section 6.7, we present a auction web service case study and a series of experiments using our tool. Finally, in Section 6.8 we present our conclusions and discuss future work.

6.2 Related Work

There exist a plethora of commercial performance testing tools. In the following, we briefly enumerate couple of popular performance testing tools. FABAN is an open source framework for developing and running multi-tier server benchmarks [18]. FABAN has a distributed architecture meaning load can be generated from multiple machines. The tool has three main components: *A harness* - for automating the process of a benchmark run and providing a container for the benchmark driver code, a *Driver framework* - provides an API for people to develop load drivers, and an *Analysis tool* - to provide comprehensive analysis of the data gathers for a test. Load is generated by running multiple scripts in parallel. JMeter [19] is an open source

Java tool for load testing and measuring performance, with the focus on web applications. Jmeter can be set up in a distributed fashion and load is generated from manually created scenarios that are run in parallel. Httperf [6] is a tool for measuring the performance of web servers. Its aim is to facilitate the construction of both micro and macro-level benchmarks. Httperf can be set up to run on multiple machines and load is generated from pre-defined scripts. LoadRunner [7] is a performance testing tool from Hewlett-Packard for examining system behavior and performance. The tool can be run in a distributed fashion and load is generated from pre-recorded scenarios.

Recently several authors have focused on using models for performance analysis and estimation, as well as for load generation. Barna et al., [2] present a model-based testing approach to test the performance of a transactional system. The authors make use of an iterative approach to find the workload stress vectors of a system. An adaptive framework will then drive the system along these stress vectors until a performance stress goal is reached. They use a system model, represented as a two-layered queuing network, and they use analytical techniques to find a workload mix that will saturate a specific system resource. Their approach differs from ours in the sense that they use a model of the system instead of testing against a real implementation of a system.

Other related approaches can be found in [16] and [15]. In the former, the authors have focused on generating valid traces or a synthetic workload for inter-dependent requests typically found in sessions when using web applications. They describe an application model that captures the dependencies for such systems by using Extended Finite State Machines (EFSMs). Combined with a workload model that describes session inter-arrival rates and parameter distributions, their tool *SWAT* outputs valid session traces that are executed using a modified version of *httperf* [12]. The main use of the tool is to perform a sensitivity analysis on the system when different parameters in the workload are changed, e.g., session length, distribution, think time, etc. In the latter, the authors suggest a tool that generates representative user behavior traces from a set of Customer Behavior Model Graphs (CBMG). The CBMG are obtained from execution logs of the system and they use a modified version of the *httperf* utility to generate the traffic from their traces. The methods differ from our approach in the sense they both focus on the trace generation and let other tools take care of generating the load/traffic for the system, while we do on-the-fly load generation from our models.

Denaro [4] proposes an approach for early performance testing of distributed software when the software is built using middleware components technologies, such as J2EE or CORBA. Most of the overall performance of

such a system is determined by the use and configuration of the middleware (e.g. databases). They also note that the coupling between the middleware and the application architecture determines the actual performance. Based on architectural designs of an application the authors can derive application-specific performance tests that can be executed on the early available middleware platform that is used to build the application with. This approach differs from ours in that the authors mainly target distributed systems and testing of the performance of middleware components.

6.3 The Performance Testing Process

In this section we are briefly going to describe the steps of the performance testing process. A more detailed description is given in Section 6.6.

6.3.1 Model Creation

Before we start generation load for the system we first have to create a load profile or a load model that describe the behavior of the users. Since we can not have a model for each individual user we have to create one or several models that represent the behavior for a larger group of users. These models describe how a groups of virtual users (VUs) behave and they are simplified models of how a real users would behave. Section 6.5 gives more details of how the models are constructed. Essentially, we use probabilistic timed automata (PTA) to specify user behavior which describe in an abstract way the sequence of actions a VU can execute against the system and their probabilistic distribution.

6.3.2 Model Validation

Once the models have been created they are checked for consistency and correctness. For instance, we check that the models have a start and end point, that there are no syntactical errors in the models, and that the probabilities and actions have been defined correctly. Once the models have been checked by the MBPeT tool we start generating load for the system under test (SUT).

6.3.3 Test Setup

Before we can actually start generating load we need to set up everything correctly so that the MBPeT can connect to the SUT and generate the appropriate amount of load. To do that one have to fill in a settings file. This file contains e.g., the IP-address of the SUT, what load models to use, how

many parallel virtual users to simulate, ramp up period, and the duration of the performance test. The MBPeT tool needs this information in order to be able to generate the right amount of load.

The tester also needs to implement an adapter for the tool. Every SUT will have its own adapter implementation. The purpose of the adapter is to translate the abstract actions found in the model into concrete actions understandable by the SUT. In case of a web page, a *browse* action would need to be translated into a HTTP *GET* request.

6.3.4 Load Generation

Once everything is set up, load generation begins. The MBPeT tool generates load from the models by starting a new process for every simulated user. Inside that process load is generated by executing the PTA model. For more details please see Section 6.6.2. Please see Section 6.5.2 for more information on PTAs.

6.3.5 Monitoring

During the load testing phase the MBPeT tool monitors the traffic sent on the network to the SUT. The tool monitors the throughput and response time for every action sent to the system. If there is a possibility to connect to the SUT remotely, the MBPeT tool can also monitor the utilization of the CPU, memory, network, disk, etc. This information can be very useful when trying to identify potential bottlenecks in the system. Once the test run is complete and all information is gathered, the tool will create a test report.

6.3.6 Test Reporting

The test report contains information about the parameters monitored during the performance test. It gives statistical values of the mean and max response time for individual actions and displays graphs that show how the repones time varied over time when the load increases. If the tool can be connected remotely to the SUT, the test report will also show how the CPU, memory, and disk was utilized over time when the load was applied to the SUT. Both of these sources of information can be helpful when trying to pin the a potential bottleneck in the system.

6.4 MBPeT Tool Architecture

MBPeT has a distributed architecture. It consists of two types of nodes: a master node and slave nodes. A single master node is responsible of initiating and controlling multiple remote slave nodes, as shown in Figure 6.1. Slave nodes are designed to be identical and generic, in a sense that they do not have prior knowledge of the SUT, its interfaces, or the workload models. That is why for each test session, the master gathers and parses all the required information regarding the SUT and the configuration for each test session and sends that information to all the slave nodes. Once all slaves have been initialized, the master begins the load generation process by starting a single slave while rest of the slaves are idling.

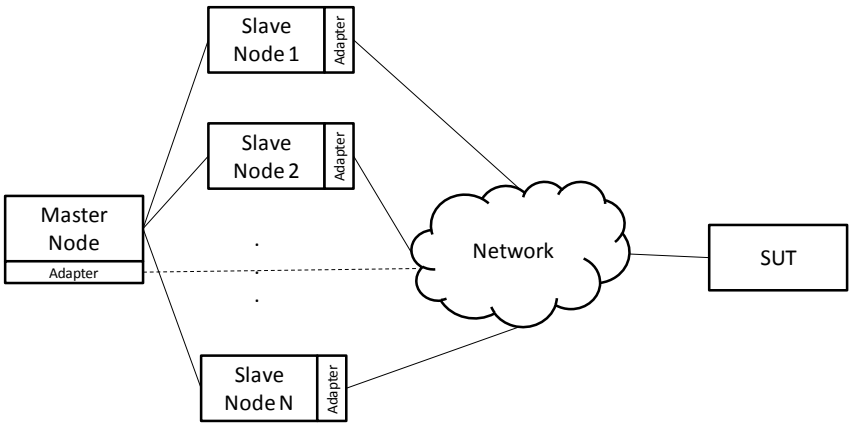


Figure 6.1: Distributed architecture of MBPeT tool

6.4.1 The Master Node

The internal architecture of the master node is shown in Figure 6.2. It contains the following components:

Core Module

The core module of the master node controls the activities of other modules as well as the flow of information among them. It initiates the different modules when their services are required. The core module takes as input the following information and distributes it among all the slave nodes:

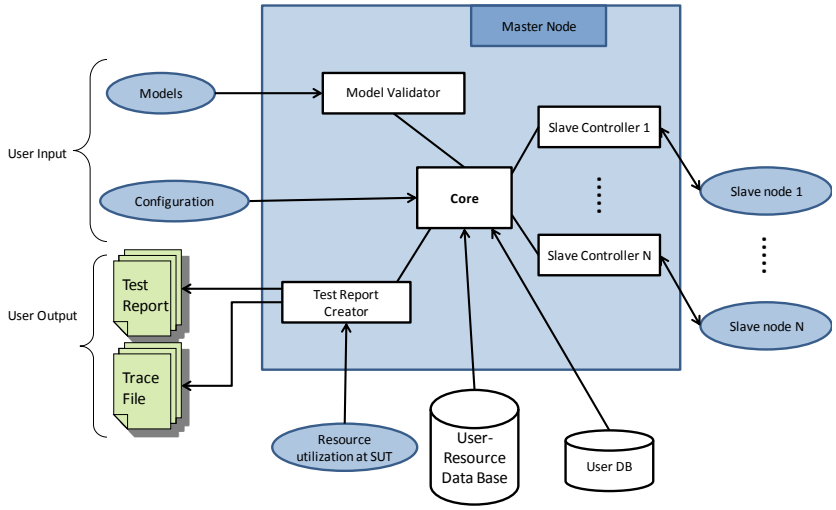


Figure 6.2: Master Node

1. *User Models:* PTA models are employed to mimic the dynamic behavior of the users. Each case-study can have multiple models to represent different types of users. User models are expressed in DOT language [5].

2. *Test Configuration:* It is a collection of different parameters, that are defined in a *Settings* file, which is a case-study specific. A *Settings* file specifies the necessary information about the case-study and this information is later used by the tool to run the experiment. There are some mandatory parameters in the *Settings* file, which have been listed below with the brief description. These parameters can also be provided as command-line arguments to the master node.

   (a) *Test duration*: It defines the duration of a test session in seconds.

   (b) *Number of users*: It specifies the maximum number of concurrent users for a test session.

   (c) *Ramp*: The ramp period is specified for all types of users. It can be defined in two ways. One way is to specify it as a percentage

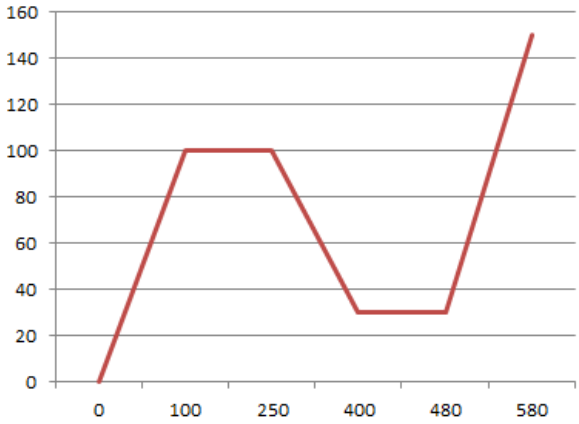


Figure 6.3: Example ramp function

of the total test duration. For example, if the objective of the experiment is to achieve the given number of concurrent users within the 80% of total test duration, then the ramp value would be equal to 0.8. Then, the tool would increase the number of users at a constant rate, in order to achieve the given number of concurrent users within the ramp period.

The ramp period can also be defined as an array of tuples. For instance the ramp function depicted in Figure 6.3, as illustrated in the Listing 6.1. A pair value is referred to as a *milestone*. The first integer in a milestone describes the time duration in seconds since the experiment started and the second integer states the target number of concurrent users at that moment. For example, the fourth milestone in the Listing 6.1, that is (400, 30), indicates that at 400 seconds the number of concurrent users should be 400, and thus starting from the previous milestone (100, 30) the number of concurrent users should drop linearly in the interval 250-400 seconds. Further, a ramp period may consist of several milestones depending upon the experiment design. The benefit of defining the ramp period in this way is that the number of concurrent users could increase and decrease during the test session.

Listing 6.1: Ramp section of Settings file

```
#================ Ramp Period ================
ramp_list = [(0, 0), (100, 100), (250, 100),
(400, 30),(480, 30), (580, 150), ... ]
```

(d) *Monitoring interval*: It specifies how often a slave node should check and report its own local resource utilization level for saturation.

(e) *Resource utilization threshold*: It is a percentage value which defines the upper limit of local resource load at the slave node. A slave node is considered to be saturated if the limit is exceeded.

(f) *Models folder*: A path to a folder which contains all the user models.

(g) *Test report folder*: The tool will save the test report at this given path.

In addition to mandatory parameters, the *Settings* file can contain other parameters, which are related to a particular case-study only. For example, if a SUT is a web server then the IP address of the web server would be an additional parameter in the *Settings* file.

3. *Adapter:* This is a case-study specific module which is used to communicate with SUT. This module translates each action interpreted from the PTA model into a form that is understandable by the SUT, for instance a HTTP request. It also parses the response from the SUT and measures the response time.

4. *Number of Slaves:* This number tells the master node how many slave nodes that are participating in the test session.

Two test databases are used by MBPeT: a user database and a user resource database. The user database contains all the information regarding users such as usernames, passwords or name spaces. In certain cases, the current state of the SUT must be captured, in order to be able to address at load generation time data dependencies between successive requests. As such, the user resource database is used to store references to the resources (e.g. files) available on the SUT for different users. The core module of the master node uses an instance of the test adapter to query the SUT and save that data in the user resource database.

Further, the core module remotely controls the Dstat[1] tool on SUT via SSH protocol. Dstat is a tool that provides detailed information about the system resource utilization in real-time. It logs the system resources utilization information after every specific time interval, one second by default. The delay between each update is specified in the command along with the names of resources to be monitored. This tool creates a log file in which it appends

[1] http://dag.wieers.com/home-made/dstat/

a row of information for each resource column after every update. The log file generated by the Dstat tool is used as basis for generating the test report, including graphs on how SUT's KPIs vary during the test session.

Model Validation Module

The *Model Validator* module validates the load models. It performs different numbers of syntactic checks on all models and generates a report.This report gives error descriptions and the location in model where the error occurred. A model with syntax anomalies could lead to inconclusive results. Therefore it is important to ensure that the all given models are well-formed and no syntax mistakes have been made in implementing the models. Examples of couple of validation rules are:

- Each model should have an initial and a final state
- All transitions have either probabilities or actions
- The sum of probabilities of transitions originating from a location is 1.
- All locations are statically reachable

Slave Controller Module

For each slave node there is an instance of *SlaveController* module in the master node. The purpose of the SlaveController module is to act as a bridge between slave nodes and the core master process and to control the slave nodes until the end of the test. The benefit of this architecture is to keep the master core process light and active, and more scalable. The SlaveController communicates with master core process only in few special cases, so that the core process could perform other tasks instead of communicating with slave nodes. Moreover, it also increases the parallelism in our architecture, all the SlaveControllers and the master's core processes could execute in parallel on different processor cores. Owning to the efficient usage of available resources, the master can perform more tasks in less period of time. A similar approach has been employed at the slave node, where each user is simulated as an independent process for the performance gain.

Test Report Creation Module

This module performs two tasks: Data Aggregation and Report Creation. In the first task, it combines the test results data from all slaves into an internal representation. Further, it retrieves the log file generated by the Dstat tool

from the SUT via Secure File Transfer Protocol (SFTP). The second task of this module is to calculate different statistical indicators and render a test report based on the aggregated data.

6.4.2 The Slave Node

Slave nodes are started with one argument, the IP-address of the master node. The *Core* module opens the socket and connects to the master node at the given IP-address with the default port number. After connecting with the master node successfully, it invokes the Load Initiator module.

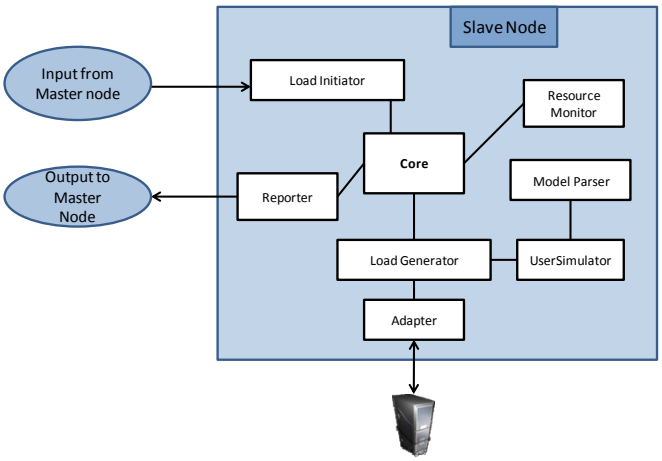


Figure 6.4: Slave Node

Load Initiation Module

The *Load Initiator* module is responsible for initializing the test setup at the slave node as well as storing the case-study and model files in a proper directory structure. It receives all the information from the master node at initialization time.

Model Parser Module

The *Model Parser* module reads the PTA model into an internal structure. It is a helper module that facilitates the UserSimulator module to perform different operations on the PTA model.

Load Generation Module

The purpose of this module is to generate load for the SUT at the desired rate, by creating and maintaining the desired number of concurrent virtual users. It uses the *UserSimulator* module to simulate virtual users where each instance of UserSimulator presents a separate user with unique user ID and session. The UserSimultor utilizes the Model Parser module to get the user's action from the user model and uses the Adapter module to perform the action. Then it waits for a specified period of time (i.e. the user think time) before performing the next action, which is chosen based on the probabilistic distribution.

Resource Monitoring Module

The *Resource Monitor* module runs as a separate thread and wakes up regularly after a specified time period. It performs two tasks every time it wakes up: 1) checks the local resource utilization level and saves the readings, 2) calculates the average of resource utilizations over a certain number of previous consecutive readings. The value obtained from the second task is compared with resource utilization threshold value, defined in the test configuration. If the calculated average is above a set threshold value of 80 percent, then it means that the slave node is about to saturate and the master will be notified. When a slave is getting saturated, its current number of generated users is kept constant, and additional slaves will be delegated to generate the more load.

Reporter Module

All the data that has been gathered during the load generation is dumped into files. The Load Generator creates a separate data file for each user; it means that the total number of simulation data files would be equal to the total number of concurrent users. In order to reduce the communication delay, all these data files are packed into a zip file, and sent to the master at the end of the test session.

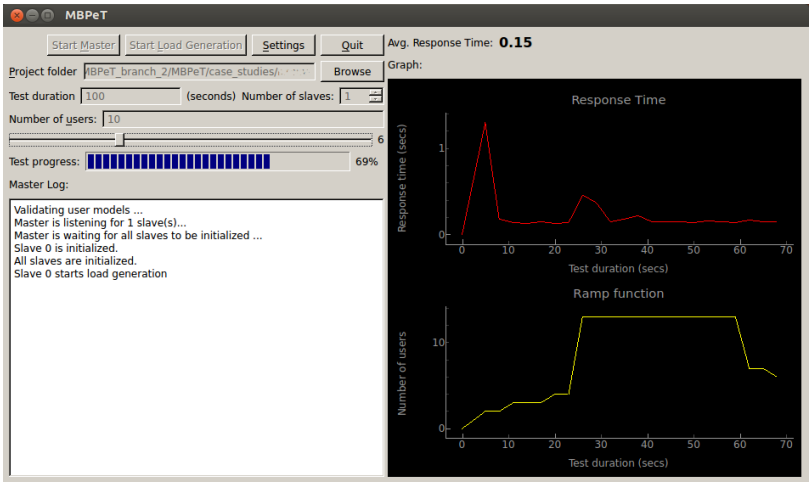


Figure 6.5: Main window of the GUI

6.4.3 Graphical User Interface

The MBPeT tool can be run both in command line and via a graphical user interface (GUI) as shown in Figure 6.5. Feature-wise the GUI is almost identical to the command-line version except for two features:

- The GUI implements the number of users as a slider function. This implies that the number of parallel user can be increased and decreased in real time using the slider, as an alternative to predefining a ramp function at beginning of the test session;
- The average response observed by all slave nodes is plotted in real-time. The response time graphs can be configured to display either one average response time plot for all actions (as currently depicted in Figure 6.5) or one average response time plot for each individual action type.

Additionally, from the GUI, one can specify basically all the test session settings previously described in Section 6.4.1

6.5 Model Creation

In this section we will introduce the load models used for generating load and describe how they are constructed. We will also in theory describe how

load is generated from these models.

6.5.1 Workload Characterization

Traditionally, performance analysis starts first with identifying key performance scenarios, based on the idea that certain scenarios are more frequent than others or certain scenarios impact more on the performance of the system than other scenarios. A performance scenario is a sequence of actions performed by an identified group of users [13]. In some cases, key performance scenarios can consist of only one action, for example "browse", in the case of a web-based system. In the case of Amazon online store, examples of key performance scenarios could be: searching for a product, then adding one or more products into the shopping cart and finally pay for them. In the first example, only one action is sent to the system, namely "browse". In the second example, several actions would have to be sent to the server, e.g. "login", "search","add-to-cart","checkout", etc.

In order to build the workload model, we start by looking and analyzing the requirements and the system specifications, respectively. During this phase we try to get an understanding of how the system is used, what are the different types of users, and what are the key performance scenarios that will impact most on the performance of the system. A user type is characterized by the distribution and the types of actions if performs.

The main sources of information for workload characterization are: Service Level Agreements (SLAs), system specifications and standards, and server execution logs [11]. By studying these sources we identify the inputs of the system with respect to types of transactions (actions), transferred files, file sizes, arrival rates, etc. following the generic guidelines discussed in [3]. In addition, we extract information regarding the KPIs, such as the number of concurrent users the system should support, expected throughput, response times, expected resource utilization demands etc. for different actions under a given load.

We use the following steps in analyzing the workload:

1. Identify the actions that can be executed against the system.
   (a) Analyze what are the required input data and output data for each action. For instance, what is the request type, its parameters, etc.
   (b) Identify dependencies between actions. For example, a user can not execute a logout action before a login action.
2. Identify what classes (types) of users execute each action

3. Identify the most relevant user types.

4. Define the distribution of actions that is performed by each user type.

5. Define an average *think time* per action for each user type.

Table 6.1 shows an example of a user type specification, its actions, action dependencies, and think time ordered in a tabular format. Based on this information we build a *workload model* described as a *probabilistic timed automata* or *PTA*.

Action	Dependency	User Type 1		User Type 2	
		Think time	Frequency	Think Time	Frequency
a_1		t_1	f_1	t_2	f_2
a_2	a_1	t_3	f_3		
a_3	a_1			t_4	f_4
a_4	a_2	t_5	f_5		
a_5	a_4	t_6	f_6	t_7	f_7
a_6	a_3			t_8	f_8

Table 6.1: Example of user types and their actions

6.5.2 Workload Modeling Using PTA

The results of the workload characterization are aggregated in a workload model similar to the one in Figure 6.6, which mimics the real workload under study. One such workload model is created for each identified user type. Basically, the model will depict the sequence of actions a user type can perform and their arrival rate, as a combination of the probability that an action is executed and the think time of the user for that action. In addition, we also identify the user types and their probabilistic distribution. A concrete example will be given in Section 6.7.

All the information that is extracted from the previous phase is aggregated in a workload model which is describes as a probabilistic timed automaton (PTA). A PTA is similar to a state machine in the sense that a PTA consists of a set of locations connected with each other via a set of transitions. However, a PTA also include the notion of time and probabilities. Time is modeled as an invariant clock constraint on transitions and increase at the same rate as real time.

A *probabilistic timed automaton* (PTA) is defined [9] as T = (L,C,*inv*, *Act*, E, δ) where:

- *a set of locations L;*

- *a finite set of clocks C;*
- *an* invariant condition *inv :* $L \rightarrow Z$;
- *a finite set of* actions *Act;*
- *an* action enabledness function $E : L \times Act \rightarrow Z$;
- *a* transition probability function $\delta : (L \times Act) \rightarrow D(2^C \times L)$.

In the above definitions, Z is a set of clock zones. A clock zone is a set of clock values, which is a union of a set of clock regions. Δ is a probabilistic transition function. Informally, the behavior of a probabilistic timed automaton is as follows: In a certain location l, an action a can be chosen when a clock variable reaches its value with a certain probability if the action is enabled in that location l. If the action a is chosen, then the probability of moving to a new location l' is given by δ[l,a](C',l'), where C' is a particular set of clocks to be reset upon firing of the transitions. Figure 6.6 gives an example of a probabilistic timed automata.

The syntax of the automata is as follows: Every transition has an initial location and an end location. Each location is transitively connected from the initial location. The transitions can be labeled with three different values: a probability value, an action, and a clock. The *probability* indicates the chance of that transition being taken. The *action* describes what action to take when the transition is used, and the *clock* indicates how long to wait before firing the transition. Every automaton has an *end location*, depicted with a double circle, that will eventually be reached. It is possible to specify loops in the automaton. It is important to notice that the sum of the probabilities on all outgoing transitions from a given location must be equal to 1. For example, consider location *2* in Figure 6.6: for the PTA to be complete the following must apply: p1 + p2 + p3 = p4 + p5 = 1.

6.6 Performance Testing Process

In this section we describe the performance testing process. Figure 6.7 shows the three steps involved in the process. In the following, we will discuss the three steps in more detail.

6.6.1 Test Setup

Every test run starts with a test setup. In each test setup, there is one master node that carries out the entire test session and generates a report. The

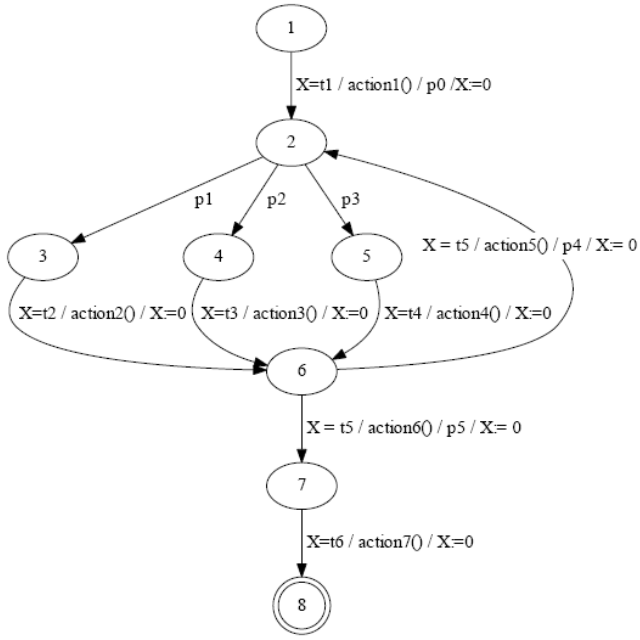


Figure 6.6: Example of a probabilistic timed automaton.

user only interacts with the master node by initializing it with the required parameters (mentioned in the Section 6.4.1) and getting the test report at the end of the test run. The parameter given to the master is the project folder. This folder contains all the files needed for load generation, such as the adapter code, the settings file (if command line mode is used) and other user specific files.

The adapter file and the settings file are the most important. The adapter files explains how the abstract actions found in the load models are translated to concrete actions. The settings file contain information about the test session, such as the location of the load models, IP-address to the SUT, the ramp function, test duration, etc. The same information can also be set from the GUI via the *Settings* button, see Figure 6.8. In here, the user is required to enter the same information as given in the settings file. Additionally, the path to the adapter file and the load models have to be given.

As one may notice in Figure 6.8, the user has the option of defining an average *think time* for the models and its *standard deviation*. If these options are used, the individual think time specified in the models for each action

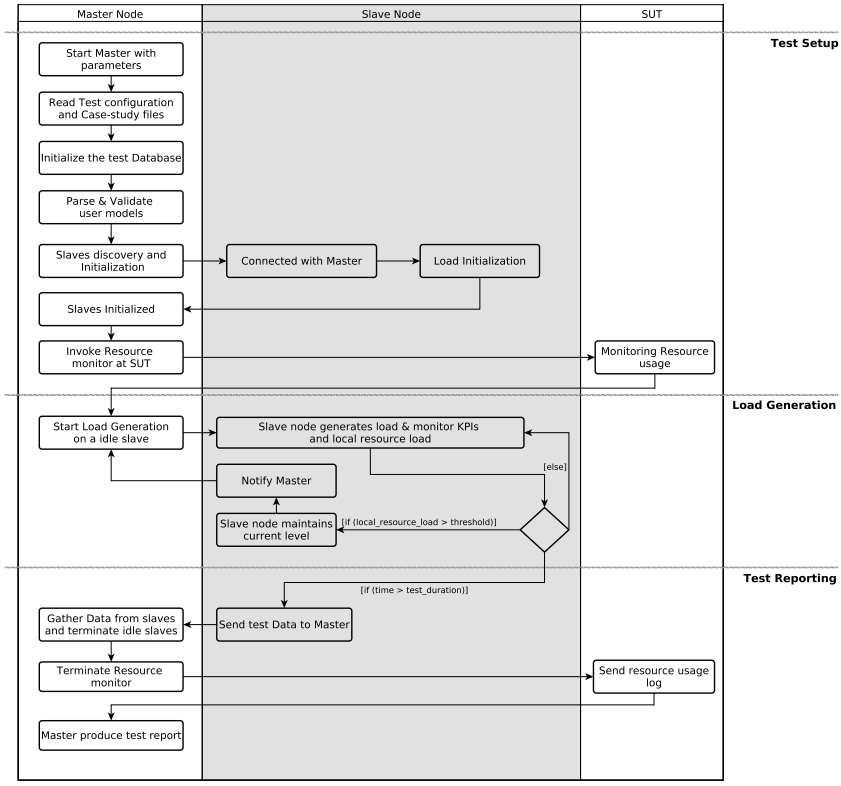


Figure 6.7: MBPeT tool activity diagram

will be ignored and the one specified in the GUI will be used.

Once the required information has been given, the master node sets up the test environment. After that, it invokes the *Model Validator*. This module validates the syntax of user models. If the validation fails, it gives the user a choice whether the user wants to continue or not to load generation. If the user decides to continue or the validation was successful, then the master enters into the next phase.

6.6.2 Load Generation

Load is generated for the models based on the same principles as described in section 6.5.2. The load generation is based on a deterministic choice with a probabilistic policy. This introduces certain randomness into the test process

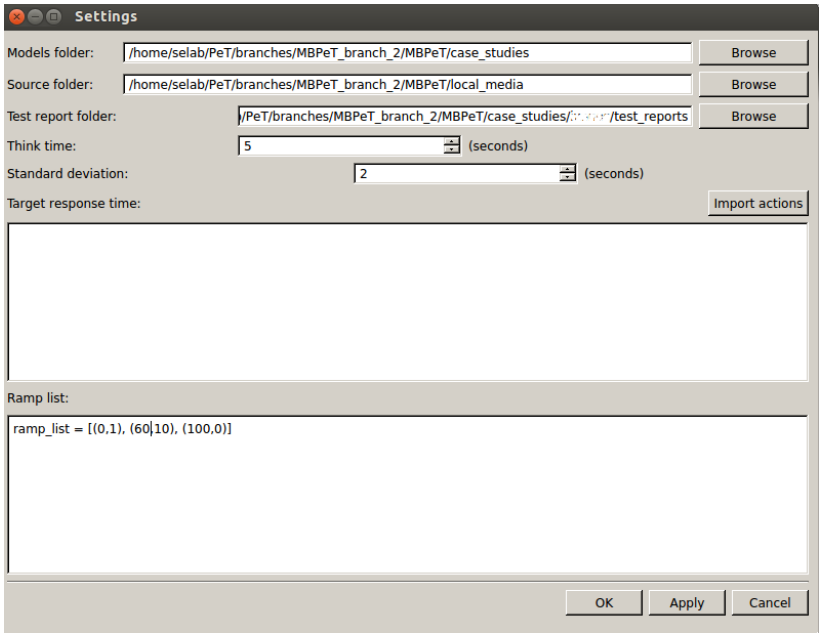


Figure 6.8: Settings window of the GUI

and that can be useful for uncovering certain sequences of actions which may have a negative impact of the performance. Such sequences would be difficult or maybe impossible to discover if static test scripts are used, where a fixed order of the actions is specified, and repeated over and over again. Every PTA has an *exit* location which will eventually be reached. By modifying the probability for the *exit* action, it is possible to adjust the length of the test.

The attributes of PTA models make them a good candidate for modeling the behavior of VUs, which imitate the dynamic behavior of real users. Actions in the PTA model corresponds to an action which a user can send to the SUT and the clocks present the user *think time*. In our case, the PTA formalism is implemented using the DOT notation.

Load is generated from these models by executing an instance of the model for every simulated VU. Whenever a transition with an action is fired, that action is translated by the MBPeT tool and sent to the SUT. This process is repeated and run in parallel for every simulated user throughout the whole test session. During load generation, the MBPeT tool monitors the SUT the whole time.

6.6.3 Test Reporting

After each test run the MBPeT tool generates a test report based on the monitored data. It is the slave nodes that are responsible for the monitoring and they report the values back to the master node which later creates the report.

Every slave node will monitor the communication with the SUT and collecting the data needed for test report. The slave node will start a timer every time and action is sent to the system. When a response is received, the timer is stopped and the response code together with the action name and response time is stored. This data is later sent to the master node which will aggregate the data and produce a report.

The slave node will also monitor its own resources so it does not get saturated and becomes the bottleneck during load generation. The slave node monitors is own CPU, memory, and disk utilization and sends the information to the master node. The master node the data is plotted in graphs and included in the test report.

It is the test report creation module of the master node that is responsible for creating test report. This module performs two tasks: aggregating data received from the slave nodes and creating a test report. Data aggregation consists of combining data received from the slave nodes together into and internal representation. Based on the received data, different kinds of statistical values are computer, e.g. mean and max response times, throughput, etc. Values such as response time and throughput plotted as graphs so the tester can see how the different values varies over time. Figures of the test report will later be shown throughout Section 6.7.

The final task of the test report creation module is to render all the values and graphs into a report. The final report is rendered as a HTML document.

6.7 Experiments

In this section we will describe a set of experiments carried out with the MBPeT tool on a case study. The system tested in the case study is an HTTP based auction web service.

6.7.1 YAAS

YAAS is a web application and a web service for creating and participating in auctions. An auction site is a good example of a service offered as a web application. It facilitates a community of users interested in buying or selling diverse items, where any user including guest user can view all the auctions

and all authenticated users, except seller of an item, can bid on the auction against other users.

The web application is implemented in Python language using the Django[2] web-framework. In addition to HTML pages, YAAS also has a RESTful [10] web service interface. The web service interface has various APIs to support different operations, including:

Browse API It returns the list of all active auctions.

Search API It allows to search auctions by title.

Get Auction This API returns an auction against the given Auction-ID.

Bids It is used to the get the list of all the bids have been made to a particular auction.

Make Bid Allows and authenticated user to place a bid on a particular auction.

6.7.2 Test Architecture

A setup of the test architecture can be seen in Figure 6.9. The server runs an instance of the YAAS application on top of an Apache web server. All nodes (master, slaves, and the server) feature an 8-core CPU, 16GB of memory, 1Gb Ethernet, 7200 rpm hard drive, and Fedora 16 operating system. The nodes were connected via a 1Gb ethernet over which the data were sent.

A populator script is used to generate input data (i.e., populate the test databases) on both the client and server side, before each test session. This ensures that the test data on either sides is consistent and easy to rebuild after each test session.

6.7.3 Load Models

The test database of the application is configured with a script to have 1000 users. Each user has exactly one auction and each auction has one starting bid.

In order to identify the different type of users for the YAAS application, we have used the AWStats[3] tool. This tool analyzes the Apache server access logs to generate a report on the YAAS application usage. Based on that report, we discovered three types of users; aggressive, passive and non-bidder.

[2]https://www.djangoproject.com/
[3]http://awstats.sourceforge.net

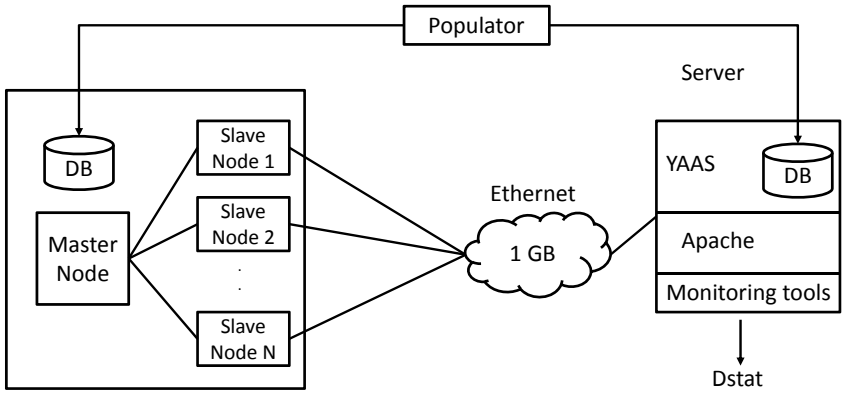


Figure 6.9: A caption of the test architecture

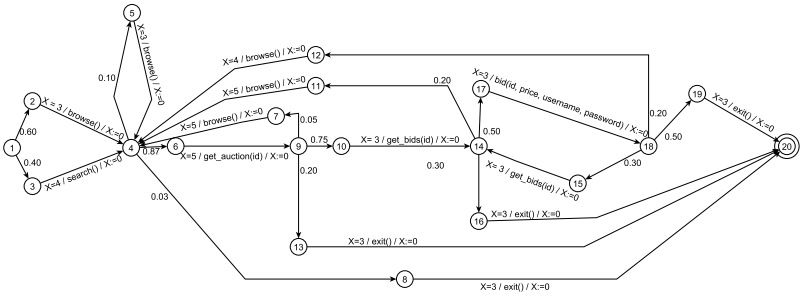


Figure 6.10: Aggressive User type model

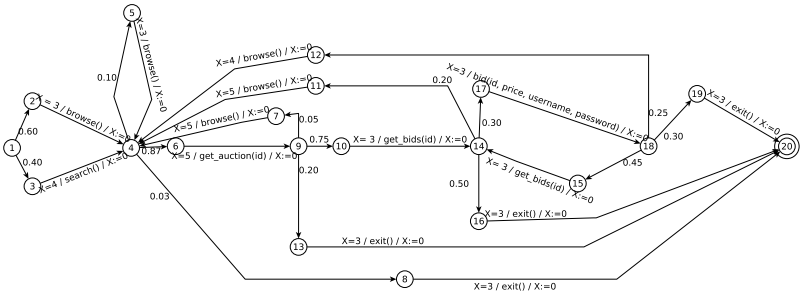


Figure 6.11: Passive User type model

Action	Dependency	Aggressive User		Passive User		Non-Bidder User	
		Think time	Frequency	Think Time	Frequency	Think Time	Frequency
search()		4	0,40	4	0,40	4	0,40
browse()		3	0,60	3	0,60	3	0,60
browse()	browse(),search()	5	0,10	3	0,10	3	0,10
get_auction()	browse(),search()	5	0,87	5	0,87	5	0,87
exit()	browse(),search()	3	0,03	3	0,03	3	0,03
browse()	get_auction()	5	0,05	5	0,05	5	0,05
get_bids()	get_auction()	3	0,75	3	0,75	3	0,75
exit()	get_auction()	3	0,20	3	0,20	3	0,20
browse()	get_bids()	5	0,20	5	0,20	5	0,60
bid()	get_bids()	3	0,50	3	0,30		
exit()	get_bids()	3	0,30	3	0,50	3	0,40
get_bids()	bid()	3	0,30	3	0,45		
browse()	bid()	4	0,20	4	0,25		
exit()	bid()	3	0,50	3	0,30		

Table 6.2: Think time and distribution values extracted from the AWStats report

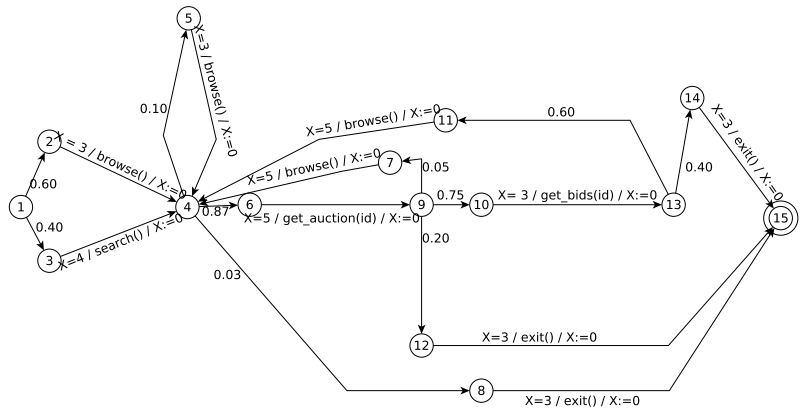


Figure 6.12: Non-bidder User type model

Table 6.2 shows the think time and distribution of actions for the three different types of users.

For each user type, a load model was created as describe in section 6.5. The aggressive type (Figure 6.10) of users describes those users, who make bids more frequently as compared to other types of users. The passive users (Figure 6.11) are less frequent in making bids, see for instance the locations 14 or 18 in the referred figures. The third type of users are only interested in browsing and searching for auctions instead of making any bids and are known as non-bidders (Figure 6.12). The root model of the YAAS application, shown in Figure 6.13, describes the distribution of different user types.

Based on the AWStats analysis, we determined that the almost 30% of total users who visited the YAAS, were very frequently in making bids, whereas rest of 50% users made bids occasionally. The rest of the users were not interested in making bids at all. This distribution is depicted by the model in Figure 6.13.

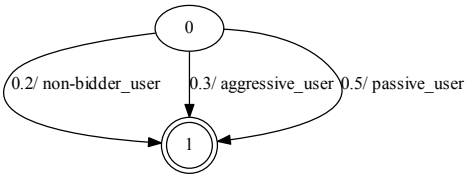


Figure 6.13: YAAS Root model

The models of all these user types were provided to the MBPeT tool to simulate them as virtual users. For example, the model of an *aggressive user type*, shown in Figure 6.10, shows that the user will start from the location *1*, and from this location the user will select either *browse* or *search* action based on a probabilistic choice. Before performing the action, the slave will wait for the think time corresponding to the selected action. Eventually, the user will reach the final location (i.e. location *20*) by performing the *exit* action and terminate the current user session. Similarly, the other models of *passive* and *non-bidder* user type have the same structure but with different probabilities and distribution of actions.

6.7.4 Experiment 1

The goal of this experiment was to set the target response time for each action and observe at what point the average response time of the action exceed the target value. The experiment ran for 20 minutes. The maximum number of concurrent users was set to 300 and the ramp up value was 0.9 that the tool would increase the number of concurrent users with the passage of time to achieve the value of 300 concurrent users when the 90% of test duration time has been passed.

The resulting test report has various sections, where each section presents the different perspective of the results. The first section, shown in Figure 6.14, contains the information about the test session including, test started time, test duration, target number of concurrent of users, etc. The *Total*

###################### Master Stats ######################

This test was executed at: 2013-07-01 16:54:47
Duration of the test: 20 min
Target number of concurrent users: 300
Total number of generated users: 27536
Measured Request rate (MRR): 27.68 req/s
Number of NON-BIDDER_USER: 6296 (23.0)%
Number of AGGRESSIVE_USER: 9087 (33.0)%
Number of PASSIVE_USER: 12153 (44.0)%
Average number of action per user: 91 actions

Figure 6.14: Test Report 1 - Section 1: General information

######## AVERAGE/MAX RESPONSE TIME per METHOD CALL ##########

	NON-BIDDER_USER (23.0 %)		PASSIVE_USER (44.0 %)		AGGRESSIVE_USER (33.0 %)	
Method Call	Average (sec)	Max (sec)	Average (sec)	Max (sec)	Average (sec)	Max (sec)
GET_AUCTION(ID)	3.04	23.95	2.85	23.67	2.93	24.71
BROWSE()	5.44	21.25	5.66	21.7	5.68	21.29
GET_BIDS(ID)	3.59	27.37	3.63	25.8	3.65	24.87
BID(ID,PRICE,USERNAME,PASSWORD)	0.0	0.0	8.26	33.44	8.11	36.84
SEARCH(STRING)	3.36	12.86	3.26	15.84	3.47	15.79

Figure 6.15: Test Report 1 - Section 2: Average and Maximum response time of SUT per action or method call

number of generated users in the report describes that the tool had simulated 27536 numbers of virtual users. The *Measured Request Rate (MRR)* depicts the average number of requests per second which were made to the SUT during the load generation process. Moreover, it also shows the distribution of total number of user generated which is very close to what we have defined in the root model (Figure 6.13). This section is useful to see the summarized view of the entire test session.

In the second section of the test report, we could observe the SUT performance for each action separately, and identify which actions have responded with more delay than the others, and which actions should be optimized to increase the performance of the SUT. As from the table in Figure 6.15, it appears that the action *BID(ID, PRICE, USERNAME, PASSWORD)* has larger average and maximum response time than the other actions. The *non-bidder* users do not perform the *BID* action that is why we have zero response time in the column of *NON-BIDDER_USER* against the *BID* action.

Section three (shown in Figure 6.16) of the test report presents a comparison of the SUTs desired performance against the measured performance. As we had defined the target response time for each action in the test config-

AVERAGE/MAX RESPONSE TIME THRESHOLD BREACH per METHOD CALL

Action	Target Response Time		NON-BIDDER_USER		PASSIVE_USER		AGGRESSIVE_USER		Verdict
	Average (secs)	Max (secs)	Average users (secs)	Max users (secs)	Average users (secs)	Max users (secs)	Average users (secs)	Max users (secs)	Pass/Fail
GET_AUCTION(ID)	2.0	4.0	70 (251)	84 (299.0)	70 (251)	95 (341.0)	70 (250)	95 (341.0)	Failed
BROWSE()	4.0	8.0	84 (299)	97 (345.0)	84 (299)	113 (403.0)	84 (299)	113 (403.0)	Failed
GET_BIDS(ID)	3.0	6.0	84 (298)	112 (402.0)	83 (296)	112 (402.0)	96 (344)	112 (401.0)	Failed
BID(ID,PRICE,USERNAME,PASSWORD)	5.0	10	Passed	Passed	97 (346)	113 (405.0)	112 (402)	135 (483.0)	Failed
SEARCH(STRING)	3.0	6	95 (341)	134 (479.0)	96 (342)	112 (402.0)	83 (296)	133 (476.0)	Failed

Figure 6.16: Test Report 1 - Section 3: Average and Maximum response time of SUT per action or method call

uration, in this section we could actually observe how many concurrent users were active when the target response time was breached. The table in this section allows us to estimate the performance of current system's implementation. For instance, the target average response time for the *GET_AUCTION* action was breached at 250 seconds for the *aggressive* type of users, when the number of concurrent users was 70. Further, this section demonstrates that the SUT can only support up to 84 concurrent users before it breaches the threshold value of 3 seconds for *GET_BIDS* action for the *passive* type of users. In summary, all the actions in Figure 6.16 have breached the target response time except the *BID* action in *NON-BIDDER_USER* column because *non-bidder* users do not bid.

Figures 6.17 and 6.18 display the resource load at the SUT during load generation. These graphs are very useful to identify which resources are being utilized more than the others and limiting the performance of SUT. For instance, it can be seen from Figure 6.17 that after 400 seconds the CPU utilization was almost equal to 100% for the rest of the test session, it means that the target web application is CPU-intensive, and it might be the reason of large response time.

Figure 6.19 illustrate that the response time of each action for the aggressive user type increases proportionally to the number of concurrent users. The figure also points out which actions response time is increasing much faster than the other actions and require optimization. Similar patterns was observed for the two other user types: passive users and non-bidder, respectively.

For example the response time of action *BID(ID, PRICE, USERNAME, PASSWORD)* for *aggressive* and *passive* user types increases more rapidly than the other actions. It might be because the *BID* action involves a write operation and in order to perform a write operation on the database file, the

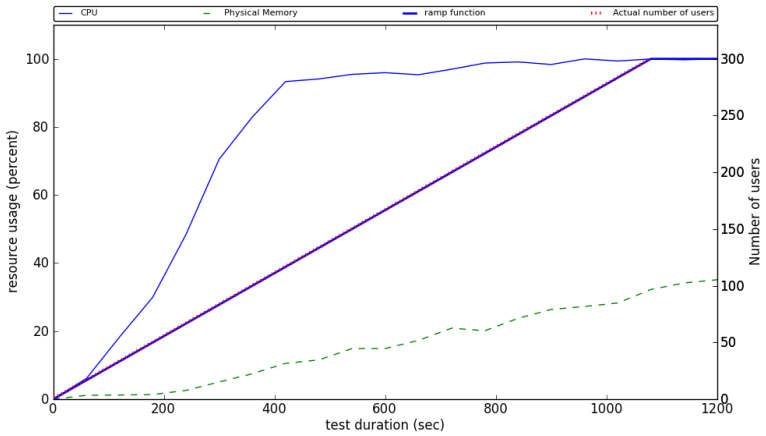


Figure 6.17: Test Report 1 - SUT CPU and memory utilization

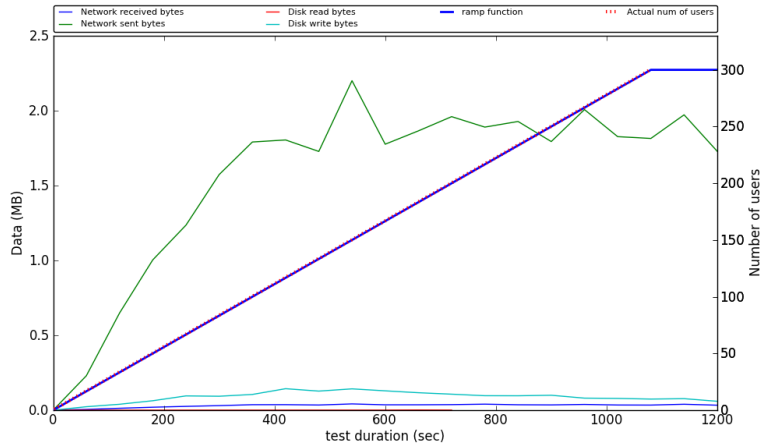


Figure 6.18: Test Report 1 - SUT network and disk utilization

SQLite[4] database has to deny the all new access requests to the database and wait until all previous operations (including read and write operations) have been completed.

Section four of the test report provides miscellaneous information about

[4]http://www.sqlite.org/

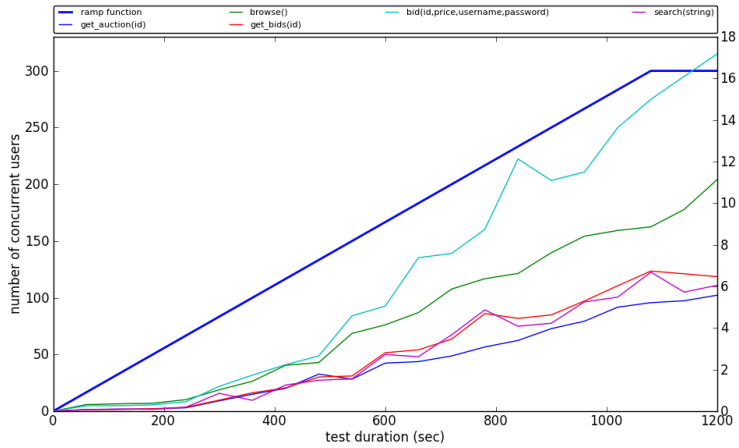


Figure 6.19: Test Report 1 - Response time of aggressive user type per action

the test session. For example, the first erroneous response was recorded at 520 seconds (according to Figure 6.20) and at that time the tool was generating load at the maximum rate, that is 1600 actions/seconds, shown in Figure 6.21. Similarly, Figure 6.20 displays that there was no error until the number of consecutive users exceeded 150, after this point errors began to appear and increased steeply proportional to the number of consecutive users.

A further deep analysis of the test report showed that the database could be the bottleneck. Owning to the fact a *sqlite* database has been used for this experiment, the application has to block the entire database before something can be written to it. It could explain the larger response time of *BID* actions compared to other actions. This is because the web application had to perform a write operation to the database in order to execute the *BID* action. Further, before each write operation, *sqlite* creates a rollback journal file, an exact copy of original database file, to preserve the integrity of database [17]. This could also delay the processing of a write operation and thus cause a larger response time.

6.7.5 Experiment 2

In the second experiment, we wanted to verify the hypothesis, which we proposed in the previous experiment: *database could be the performance bottleneck*. We ran the second experiment for 20 minutes with the same test

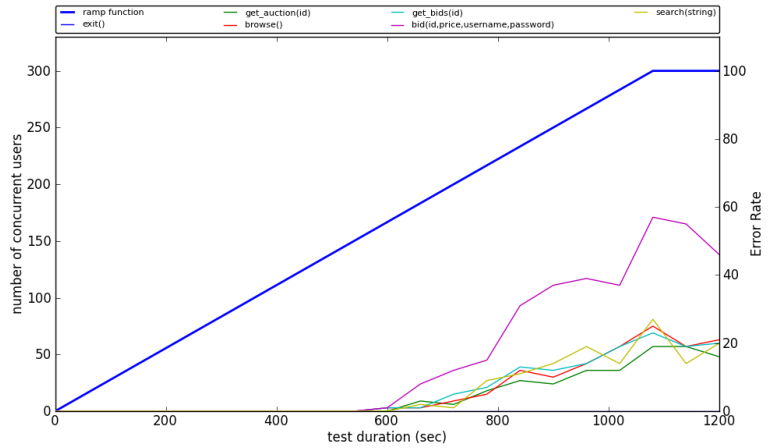


Figure 6.20: Test Report 1 - Error rate

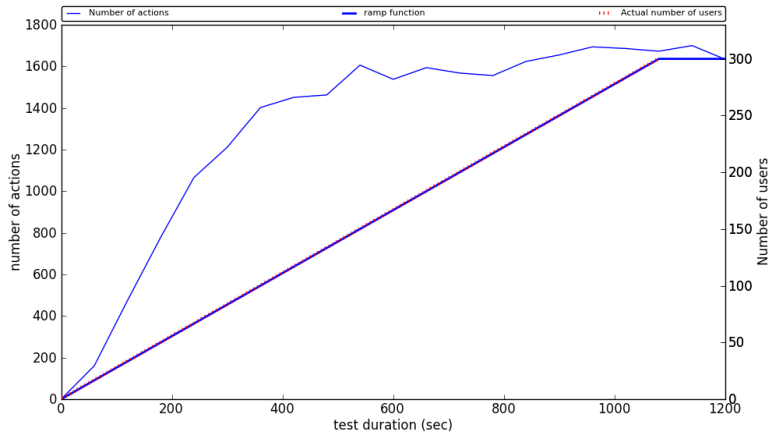


Figure 6.21: Test Report 1 - Average number of actions

configuration of the previous experiment. However, we did make one modification in the architecture. In the previous experiment, the *SQLite 3.7* was used as database server, but in this experiment, it was replaced by the *PostgreSQL 9.1* [5]. The main motivating factor of using the PostgreSQL

[5]http://www.postgresql.org

#################### Master Stats ######################

This test was executed at: 2013-07-01 17:37:38
Duration of the test: 20 min
Target number of concurrent users: 300
Total number of generated users: 35851
Measured Request rate (MRR): 39.21 req/s
Number of AGGRESSIVE_USER: 11950 (33.0)%
Number of NON-BIDDER_USER: 7697 (21.0)%
Number of PASSIVE_USER: 16204 (45.0)%
Average number of action per user: 119 actions

Figure 6.22: Test Report 2 - Section 1: global information

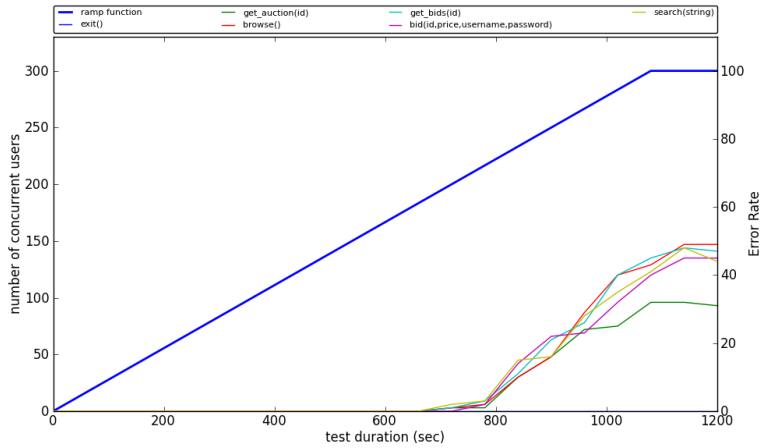


Figure 6.23: Test Report 2 - Error rate

database is that it supports the better concurrent access to the data than the SQLite. The PostgreSQL database uses the Multiversion Concurrency Control (MVCC) model instead of simple locking. In MVCC, different locks are acquired for the read and write operations, it means that the both operations can be performed simultaneously without blocking each other [14].

In the section 1 of Test report 2 (Figure 6.22) shows that the *Measured Request Rate (MRR)* increased by 42%. Additionally, each user performed averagely 30% more actions in this experiment.

Similarly in the second section (Figure 6.24), the average and maximum response time of all action decreased by almost 47%. Moreover, the error rate section (Figure 6.23) depicts that there was no error until the number

######## AVERAGE/MAX RESPONSE TIME per METHOD CALL ##########

	AGGRESSIVE_USER (33.0 %)		PASSIVE_USER (45.0 %)		NON-BIDDER_USER (21.0 %)	
Method Call	Average (sec)	Max (sec)	Average (sec)	Max (sec)	Average (sec)	Max (sec)
GET_AUCTION(ID)	1.18	15.58	1.1	15.95	1.25	15.8
BROWSE()	4.99	23.61	5.13	23.47	5.23	23.6
GET_BIDS(ID)	1.51	15.25	1.54	15.56	1.63	15.02
BID(ID,PRICE,USERNAME,PASSWORD)	3.25	18.65	3.25	18.37	0.0	0.0
SEARCH(STRING)	1.48	14.66	1.54	14.83	1.43	15.43

Figure 6.24: Test Report 2 - Section 2: Average and Maximum response time of SUT per action or method call

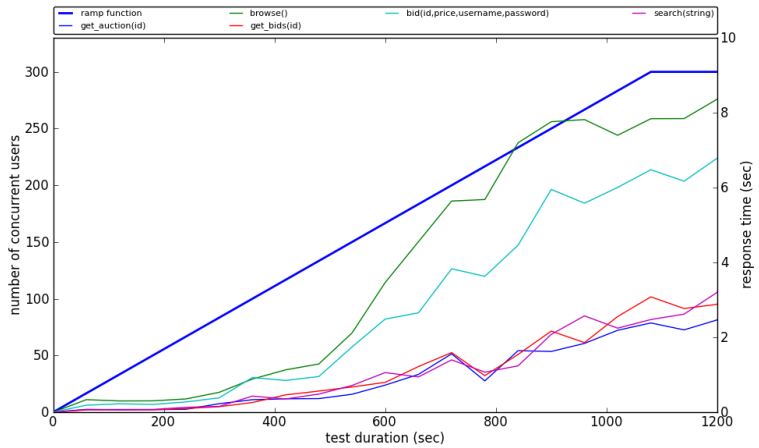


Figure 6.25: Test Report 2 - Response time of aggressive user type per action

of concurrent users was below 182, that is 21% more users than the last experiment.

Figure 6.25 shows that the response time of aggressive type of users is decreased by 50% approximately in comparison with the previous experiment in Figure 6.19. In summary, all of these indicators suggest significant improvement in the performance of SUT.

6.8 Conclusions

In this chapter, we have presented a tool-supported approach for model-based performance testing. Our approach uses PTA models to specify the probabilistic distribution of user types and of actions that are executed against

the system.

The approach is supported by the MBPeT tool, which has a distributed scalable architecture, targeted to cloud-based environments allowing it to generate load at high rates. The tool generates load in online mode and monitors different KPIs including the resource utilization of the SUT. It can be run both in command line and in GUI mode, respectively. The former facilitates the integration of the tool in automated test frameworks, whereas the latter allows the user to interact with the SUT and visualize in real-time its performance depending on the number of concurrent users.

Using our modeling approach, the effort necessary to create and update the user profiles is reduced. The adapter required to interface with the SUT has to be implemented only once and then it can be reused. As shown in the experiments, the tool allows quick exploration of the performance space by trying out different load mixes. In addition, preliminary experiments have shown that the synthetic load generated from probabilistic models has in general a stronger impact on the SUT compared to static scripts.

We have also showed that the tool us sufficient enough in finding performance bottlenecks and that the tool can handle large amounts of parallel virtual users. The tool benefits from its distributed architecture in the sense that it can easily be integrated in a cloud environment where thousands of concurrent virtual users need to be simulated.

Future work will be targeted towards improving the methods for creating the user profiles from historic data and providing more detailed analysis of the test results. So far, the MBPeT tool has been used for testing web services however, we plan also to address also web applications, as well as other types of communicating systems.

References

[1] Ashlish Jolly. *Historical Perspective in Optimising Software Testing Efforts*. 2013. URL: `http://www.indianmba.com/Faculty\_Column/FC139/fc139.html`.

[2] C. Barna, M. Litoiu, and H. Ghanbari. "Model-based performance testing (NIER track)". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 872–875. ISBN: 978-1-4503-0445-0. DOI: `10.1145/1985793.1985930`.

[3] M. Calzarossa, L. Massari, and D. Tessera. "Workload Characterization Issues and Methodologies". In: *Performance Evaluation: Origins and Directions*. London, UK, UK: Springer-Verlag, 2000, pp. 459–481. ISBN: 3-540-67193-5.

[4] G. Denaro, A. Polini, and W. Emmerich. "Early performance testing of distributed software applications". In: *Proceedings of the 4th international workshop on Software and performance*. WOSP '04. Redwood Shores, California: ACM, 2004, pp. 94–103. ISBN: 1-58113-673-0. DOI: 10.1145/974044.974059.

[5] E. Gansner, E. Koutsofios, and S North. *Drawing draphs with dot*. Online at http://www.graphviz.org/Documentation/dotguide.pdf. 2006. URL: http://www.graphviz.org/Documentation/dotguide.pdf.

[6] Hewlett-Packard. *httperf*. retrieved: October, 2012. URL: http://www.hpl.hp.com/research/linux/httperf/httperf-man-0.9.txt.

[7] HP. *HP LoadRunner*. 2013. URL: http://www8.hp.com/us/en/software-solutions/software.html?compURI=1175451\#.URz7wqWou8E.

[8] ITEA 2. *ITEA 2 D-MINT project result leaflet: Model-based testing cuts development costs*. 2013. URL: http://www.itea2.org/project/result/download/result/5519?file=06014_D_MINT_Project_Leaflet_results_oct_10.pdf.

[9] M. Jurdziński et al. "Concavely-Priced Probabilistic Timed Automata". In: *Proc. 20th International Conference on Concurrency Theory (CONCUR'09)*. Ed. by M. Bravetti and G. Zavattaro. Vol. 5710. LNCS. Springer, 2009, pp. 415–430.

[10] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media. 2007.

[11] D. A. Menasce and V. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN: 0130659037.

[12] D. Mosberger and T. Jin. "httperfa tool for measuring web server performance". In: *SIGMETRICS Perform. Eval. Rev.* 26.3 (Dec. 1998), pp. 31–37. ISSN: 0163-5999. DOI: 10.1145/306225.306235. URL: http://doi.acm.org/10.1145/306225.306235.

[13] D. C. Petriu and H. Shen. "Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications". In: Springer-Verlag, 2002, pp. 159–177.

[14] PostgreSQL. *Concurrency Control*. retrieved: March, 2013. URL: http://www.postgresql.org/docs/9.1/static/mvcc-intro.html.

[15] G. Ruffo et al. "WALTy: A User Behavior Tailored Tool for Evaluating Web Application Performance". In: *Network Computing and Applications, IEEE International Symposium on* 0 (2004), pp. 77–86. DOI: http://doi.ieeecomputersociety.org/10.1109/NCA.2004.1347765.

[16] M. Shams, D. Krishnamurthy, and B. Far. "A model-based approach for testing the performance of web applications". In: *SOQUA '06: Proceedings of the 3rd international workshop on Software quality assurance*. Portland, Oregon: ACM, 2006, pp. 54–61. ISBN: 1-59593-584-3. DOI: http://doi.acm.org/10.1145/1188895.1188909.

[17] SQLite. *File Locking And Concurrency In SQLite Version 3*. retrieved: March, 2013. URL: http://www.sqlite.org/lockingv3.html.

[18] Sun. *Faban Harness and Benchmark Framework*. 2013. URL: http://java.net/projects/faban/.

[19] The Apache Software Foundation. *Apache JMeter*. Retrieved: October, 2012. URL: http://jmeter.apache.org/.

7 Cloud Communication Service

Michael Cochez[1], Sami Helin[2], and Jiawen Chen[1]

[1]Department of Mathematical Information Technology
University of Jyväskylä, P.O. Box 35 (Agora), 40014, Jyväskylä, Finland
Email: michael.cochez@jyu.fi, jiawen.chen@student.jyu.fi

[2]Steeri Oy
Tammasaarenkatu 5,00180, Helsinki, Finland
Email: sami.helin@steeri.fi

Abstract–Cloud computing has opened the path to more on-line service oriented business models. Customers are interacting with enterprises' digital systems trough a multitude of interfaces. We regard each of these possible interfaces as a communication channel. When an organization owns multiple systems with each its own communication channels, a user might get a fragmented experience and use of the channels is likely sub-optimal. From our past research we will summarize a basis for a communication framework and how measuring quality of the semantic models in that framework can be achieved. Then we give a compact overview of techniques we proposed to process the knowledge extracted from high velocity communication streams, i.e. Big Knowledge, using a system which borrows concepts from biological evolution. From our current research focus, we present techniques which should help to reach a good user experience. We mainly look at how we could filter out unwanted messages and how we could make recommendations to users who might be interested in certain types of messages. Finally, we will suggest user interfaces for the proposed solutions.

Keywords-Cloud computing, communication, semantic web, big knowledge.

7.1 Introduction

In this chapter we present the current state of the achievements in the Cloud Communication Service business case which is part of the Cloud Software Program [12]. The results presented here summarize prior published work and extend it with our current efforts.

The idea behind the Cloud Communication Service (CSS) is to solve problems which arise when communication oriented cloud services interact with customers and other systems. The components needed for this communication are typically replicated amongst the different systems which are in use by an organization. These components often work strictly in parallel, not allowing for any flow of information from the one to the other. The overall result is that the systems will not make efficient use of the available communication channels and the customer will realize this fragmentation when interacting with the company. One example could be a customer who has bought a new smart phone. When this person contacts the help desk with a certain problem at a later point, it would be helpful if the person who answers this request can help with knowledge about previous interactions. It would also be beneficial to know, or not have to care about, the way this customer prefers to be contacted. A broader analysis of the problem and potential benefits of solving it was given by Nagy [14], we will summarize its main points below. During previous research we have also devised a basic multi-channel communication framework for solving part of above mentioned challenges. [13] In parallel, we have started to build a model for evaluation of the quality of the ontologies used in this system. [6] We also worked on a way to extract the core information from the stream of data, which the system has to process at high velocity. [10] In this chapter we will give an integrated view on these parts and extend with our current work on human interaction.

The chapter is structured as follows: first, we introduce a business scenario in which the Cloud Communication Service would function. Then we give an overview of the past work followed by our current efforts. Finally, we present some of the user interfaces which were developed for the Cloud Communication Service.

7.2 Business Scenarios

In modern society, on-line shopping and cooperating are major trends in business. To buyers, it brings plenty of benefits, such as convenience and lower prices due to increased global competition. To companies, on-line shopping

stimulates the activities of business, which can now sell on the global market to customers who were unreachable before. The on-line trade has however introduced a new challenge, namely the lack of communication and comprehension towards customers. For example, certain buyers do not plan to buy more as a single item from your web shop. In a face-to-face situation, the shopkeeper could advise the customers other related products or guide them to relevant services. This is more difficult in an on-line sale and many companies have spent great effort towards the delivery of targeted advertisement, often based on customer preferences and shopping history.

Further, customer buying processes have evolved to utilize the modern variety of channels available. This has made understanding customers and providing consistent customer service and communication more challenging. When thinking about the interaction with customers, they must be put at the core. Then, the channel and content selection has to be based on the individuals' profile which includes their settings and history of previous interactions. One of the goals is to reduce negative emotional effects to customers, such as these caused by trash mail.

An example scenario of a dynamic and customer focused communication would be a shopper wanting to buy a new TV. He is a member of an electronic retail chain's customer program. He then seeks information on the store website and downloads a brochure. Then he goes on and asks for referrals from social media websites. During the purchase process he starts receiving more TV focused ads in the newsletter and promotional discounts. The on-line website gets personalized for a more convenient shopping experience around topics of interest. And if he finally decides to go to store to make the actual purchase, he also receives suggestions relevant to his needs. After the purchase process he receives a simple feedback query from the company by SMS asking the customer about his experience with service and whether he would promote the shop to his friends.

In this example case, the complete buying process was made relevant with coordinated utilization of several channels:

1. Social Communication website followers
2. Surfing on-line on the web shop
3. An SMS notification
4. A newsletters via electronic or postal mail

We suggest that to achieve consistent multichannel customer dialog the company should:

- manage content for multiple channels; ranging from encountering the customer in person to digital channels with varying capabilities,
- split the content to atomic pieces for it to be possible to dynamically draw the finally delivered content based on the customer needs and channel choice,
- understand customer behavior and interaction in different channels to coordinate customer contacts and generate customer insights,
- instead of providing a set of static messages draw relevant content blocks to match customer interaction points, and
- measure how well different operations help to reach the goals set by the company.

These topics map to research questions related to:

- automated understanding of messages to increase the throughput in the system without losing quality,
- Big data related issues when trying to understand and measure behavior,
- the definition of a communication framework including messaging, profile management, and content and channel selection,
- Recommender Systems to handle dynamic content creation which benefits from similar tasks in the history of the system.

The research and development effort in cloud communication service tries to address these challenges. Some research results regarding these topics can be found in the next sections.

7.3 Past work

Up till now three main facets have been investigated. First, we created a framework to handle the complexity of the communication system. This framework is summarized in subsection 7.3.1. Then, we looked at how we can define quality of an ontology which is one of the building blocks for the framework. This effort is described in subsection 7.3.2. Finally, a book chapter [10] about evolving knowledge ecosystems for big data understanding was published. That chapter encompasses several facets related to this research topic. An overview of these topics can be found in subsection 7.3.3.

7.3.1 Framework

The results of the main effort towards solving some of the issues of the multi-channel communication problem can be found in the framework paper by Nagy [13]. At the current moment there is no complete implementation of the framework. Rather, the theoretical framework is used as a guideline for further extension of the current implementation described in section 7.5.

An overview of the framework can be seen in Figure 7.1. The structure consists of two main building blocks. The first one, the knowledge base, is depicted in the grayed box and is divided in several ontology fragments and sub-knowledge bases. The second one, the message conversion engine, is responsible for handling incoming and outgoing messages.

The ontology in the knowledge base is subdivided in 5 parts: the *customer ontology* to model users and the way they can be contacted, the *action ontology* used for the description of high level goals, the *message ontology* used for the classification of messages, the *commodity ontology* used to described goods and services, and finally the *channel ontology* used for classifying channels and their properties. These ontologies are defined as part of the framework. The article identifies that the ontology can be expanded depending on specific business needs. Further arguments for the use of ontologies from the original framework proposal [13] include:

Expressiveness: When using ontologies it is possible to represent the concepts of the framework and the ones from the business domain using the modeling language. Ontologies also allow for later extension when the business needs change.

Soundness and completeness: Several languages for ontologies have been formalized. These formalizations include ways to deduce new information from existing facts. This deduction, also called reasoning, is sound, i.e., correct and also complete, i.e., all facts which can be found will be found.

Computational complexity: The sound and complete reasoning happens with a reasonable computation complexity.

Tool support: There are several mature tools available for ontology design, management and reasoning.

The message conversion engine first converts messages templates into abstract messages. A message template is like a blueprint for a message and contains placeholders for information which the engine will fill. The engine then chooses a destination channel and the abstract message is converted to a concrete message with a structure depending on the destination channel.

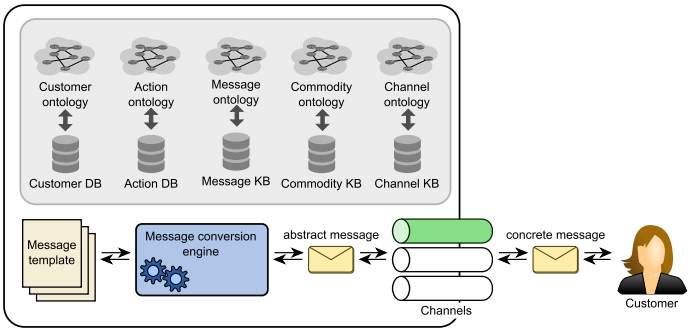


Figure 7.1: Multi-channel communication framework overview (Picture credit: Michal Nagy)

7.3.2 Quality of an Ontology

When creating an ontology, there are numerous ways of defining its quality. The main contribution of the work performed in the frame of the Cloud Communication Service is that it states the finding of an ontology with the highest quality possible as a Context-dependent dynamic multi-objective optimization problem. [6] A minor contribution is the recognition of fuzzy ontologies as a way to state inexact knowledge.

First, we discussed different features of an ontology as can be found in the literature. These features include coverage, cohesion, and coupling which are metrics representing how well concepts and relations between them are covered, how well concepts in the ontology belong together, and how strong the linking between this and external ontologies is, respectively.

These features can be measured from a given, possibly fuzzy, ontology. When the features are combined with the context in which the ontology is used, it is possible to state a quality of the ontology. The context comprises many factors. Examples include the performance when used in a system, the memory needed, how easy it is to scale the ontology, and how well it integrates into frameworks and interfaces.[1]

Next, the article shows that the quality of an ontology can not always be considered to have a single dimension only. This because of the fact that there is no total order on the quality of an ontology in a context. Therefore, it is necessary to consider the search for an optimal ontology for a given context as a multi-objective optimization problem.

The article then defines the optimization problem as follows:

First, $opt'(F(t))$ was denoted to be the optimization problem with function $F(t)_1$ and constraints $F(t)_2$. Let C be the set of contexts, O be the set of

ontologies and Q be the set of qualities. Now we can define the optimisation as:

Definition The function *sol* is the solution of the context-dependent dynamic multi-objective problem of finding an optimal ontology for a given context $\Leftrightarrow sol : C \rightarrow O$ and $sol(c) = opt'(F(c))$

Where $F\,(C) \rightarrow (O \rightarrow Q)$ a function which maps the context c to a function which incorporates the context when evaluating the quality of an ontology and its associated domain.

7.3.3 Cloud Communication System as a Big Data Problem

When the Cloud Communication System is used to handle the complete flow of information in an organization, the amount of data handled becomes such that it will he hard to process with current technology. However, we would like our system to even make sense of all the data it processes. Hence, we arrive in the field of Big Data analytics. According to Fisher et al. [11] "Fundamentally, big data analytics is a workflow that distills terabytes of low-value data (e.g., every tweet) down to, in some cases, a single bit of high-value data (Should Company X acquire Company Y? Can we reject the null hypothesis?). The goal is to see the big picture from the minutia of our digital lives."

Our previous work [10] looked at how to solve the problem of handling streams of tokens arriving at high rate and altogether representing a huge amount of data. The system described in this chapter is a concrete example of such a system. In the following subsections we will, based on the previous work, describe the balance between volume and velocity, how big data can lead to big knowledge, and how a system inspired by the mechanisms of natural evolution could provide a solution.

Volume vs. Velocity

Volume, the size of the data, and velocity, the amount of time allowed for its processing are clearly the main factors when talking about Big Data. Both of them manifest themselves when a high number of messages has to be handled within a reasonable time. When the system tries to extract information or knowledge from the data it does this effectively if no important facts are overlooked, i.e the analysis is complete, and the facts found are useful further inference, i.e they are expressive and granular. The ratio of the effort the

system spends on finding a given result to its utility can be interpreted as the efficiency of the system. Note that when one tries to improve the effectiveness of the system, the computational complexity will be increased and hence the efficiency of the system might drop. Hence, if we would like to make a deeper analysis of the message stream, we would have a less efficient system.

Big Knowledge

When we have a vast amount of data and try to extract all knowledge from it, we might end up with an unmanageable amount. From that observation we identified some aspects which should be taken into account while working with Big Data. We called this approach *3F+3Co* which stands for Focusing, Filtering, and Forgetting + Contextualizing, Compressing and Connecting. It should be noted here that these terms are not novel in the sense that they have been used in different domains and interpretations, see for example [8]. We gave an occasionally overlapping meaning to each of these terms in the context of Big Data analysis as follows.

Focusing is mainly concerned with the order in which the data is processed. An optimal focus will only scan the data which is absolutely needed to come to an answer for the question which is at hand and will hence lead to a higher efficiency. This facet will most likely play a less significant role in the messaging system since the data is arriving continuously and hence the focus will most likely be on the information which freshly arrives to the system.

Filtering is ignoring anything which is, hopefully, not of importance for future analysis. We use *hopefully* since deciding whether information is relevant or not can in most cases not be done with a hundred percent certainty. One way to filter is to only focus on specific features of the data, which also reduces the variety and complexity of the data. Similar to the focusing perspective, it is not possible to make the filter upfront since it is not feasible to accurately guess the future data.

Forgetting is a further step from filtering where data or knowledge derived from it is completely removed from the system. This trashing can remove potentially valuable information. It is very difficult to decide which part of the data can be removed. In the work which we did around Evolutionary Knowledge Systems (see section 7.3.3), we use the technique of "forgetting before storing ". This means that there has to be reason before anything is stored at all in the knowledge base.

Contextualizing comprises not only the change of sense of statements in different contexts, but also judgments, assessments, attitudes, and sentiments. There are various facets which contribute to the context of data. Examples include the origin of the data, the tools used, and the place in which the result will be used.

Compressing stands for both lossy and lossless compression. Where lossy compression is similar to Forgetting which was discussed above. The lossless compression might be very effective because the high amount of data leads to a high probability that repetitive or periodical patterns are present.

Connecting can be done if information is added to an already existing body of data. The whole body is build incrementally. The benefit of linking the data before processing it further is that data and knowledge mining, knowledge discovery, pattern recognition, etc can be performed more effectively and efficiently. A good example of this connecting used for building an index of the world wide web can be found in [16].

Evolving Knowledge Ecosystems

When messages arrive to the CCS, the system tries to forward messages, which is still in abstract form, to the correct receivers over the preferred channel. These includes both inbound and outbound messages. The actual content is, however, likely to evolve over time due to many external factors. Examples include the variation in activity of customers or the company using the system, the economical situation, the season, and so on. To anticipate these changes the CSS should be able to change its inner working, if possible automatically. In the chapter [10] we proposed an Evolving Knowledge Ecosystem which is able to adapt to changes in the environment. This Ecosystem would, when implemented, assist in the understanding of the external world. It should, again, be noted that the proposed system is more general as the parts which could be used in the CCS. In this section, however, the focus will be on the relevant parts.

The core idea behind the Ecosystem is that

> The mechanisms of knowledge evolution are very similar to the mechanisms of biological evolution. Hence, the methods and mechanisms for the evolution of knowledge could be spotted from the ones enabling the evolution of living beings.

Starting from this idea, we derived that we could model the knowledge evolution inside the system using ideas from natural evolution. One of the core

ideas is that, similar to the idea of natural selection proposed by Darwin [7], knowledge which is more fit for its environment, has a higher chance to survive as less fit knowledge. The environment here is formed by the incoming information to the system. The following concepts, borrowed from modern evolutionary biology, were further elaborated in relation to the system which processes a stream fo incoming messages. They are also illustrated in Figure 7.2.

Knowledge organism (KO) are the components which carry all knowledge in the system. They are an analog to living beings in nature.

Environment is the place where the KO reside. The environment is limited in resources and hence only KO which can consume the resources available can survive in the given place of the environment. These places are called environmental contexts.

Knowledge genome is the part of the KO which represents its terminological component, also called Terminological Box or TBox [15].

Knowledge body is the assertional component of the KO. It is similar to a ABox [15] with an ontology from the knowledge genome.

Knowledge tokens are influences from the environment, comparable to mutagens in evolutionary biology. These mutagens come either from analysis of the message streams or are excreted by a KO. They can then subsequently be consumed by a KO if that KO has capability to consume it, i.e. has a knowledge body which has enough similarity to the token.

Morphogenesis is a change in the knowledge body of a KO caused by the consumption of knowledge tokens.

Mutation is a change in the knowledge genome of a KO, which in most cases, leads to a change in the knowledge body.

Recombination is a process in which two or more KOs are combined into a new KO. This entity has a knowledge genome composed of parts of the parents. The newly created KO might be, but is not necessarily, more fit to the environment than any of its parents.

Excretion is a process by which a KO can expose of parts of its knowledge body or genome. These unused parts will be placed in back in the environment.

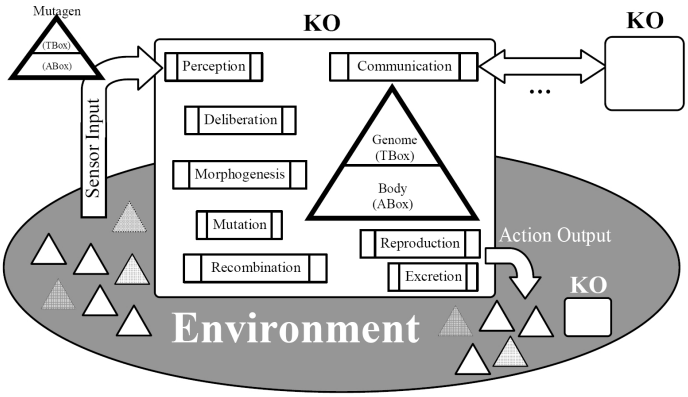


Figure 7.2: A Knowledge Organism: functionality and environment. Small triangles of different transparency represent knowledge tokens in the environment consumed and produced by KOs. These knowledge tokens may also referred to as mutagens as they may trigger mutations. (Picture credit: [10])

7.4 Human Interaction

As we mentioned above, the *3F*, focusing, filtering, and forgetting, was used earlier in different contexts. One of them is the context of management [8] where the terms are used as techniques to cope with the information overload which many people experience in our modern society. We did not investigate how much information the Cloud Communication System can send to its users before they would get a negative feeling about the system or perhaps about the company using it, which more of a social science topic. (See for instance [2]) We however want to look at how we can reduce the amount of times the user is contacted by the system, leading to a reduction of the load on humans interacting with it.

We looked into several techniques which have been used in other fields before. In the following subsections, these are described in the context of the system we are designing. First, we look from the perspective of Information Filtering and Recommender systems. Then we discuss filtering techniques which can be applied. The overall idea is that the CCS filters content out when it is irrelevant and merges what is related.

7.4.1 Information Filtering and Recommender Systems

Generally speaking, Information Filtering (IF) is a technique which could be used to automatically remove or add information according to preferences or behaviors. An IF system could be used to perform tasks like the creation of abstracts, the classification of information, and summarization. Also, IF primarily deals with unstructured or semistructured data, its most common use case is the classification of e-mail. [3] An Information Filtering system has a mediator role between the resources and its users.

There are several issues which could be solved using Information Filtering, like for instance finding a good route for messages which pass trough the system, removal of unwanted messages like spam, and the classification of messages based on their meta-data.

Recommender systems are a specific type of Information Filtering systems which can, given a set of items, propose other related items. This type of IF has been proposed in the mid-'90s [18] and is commonly used to offer on-line shoppers suggestions based on their shopping history and what other similar customers bought. Another example is the articles proposed to readers in accordance with the interests set on their profile page. In our system we could use this type of system finding related messages which could be merged together, finding users with similar interest to target messages correctly, and finding users with similar requests or history in order to correctly route a new incoming request in an optimal way. The framework which we proposed in section 7.3.1 uses semantic storages for all data in the system. Using recommender systems in combination with this kind of data stores has been studies in [19].

Information Filtering and specifically Recommender Systems can use several filtering techniques. They can be roughly divided into Content-based Information Filtering, which takes mainly the content of an individual item into account while filtering, and Collaborative Information Filtering, which looks at similar context for the query, like for instance users with similar interests as the current user. There are also approaches which combine both techniques. A good introduction to the topic can be found in [17]. These two classes of filtering techniques are further elaborated in the next subsections.

7.4.2 Content-based Information Filtering

Content-based Information Filtering uses mainly attribute of items while searching for recommendations. For example, if a visitor of the NBA.com website has read a number of articles related to the Lakers basketball team,

then the system could suggest articles from the database which are tagged with the 'Lakers' tag.

A content-based filtering system often uses a search profile which contains characteristics of the interests, in other words, it tries to relate properties of items with factors found in the search context. In our message merging case, we could use this technology to filter incoming and outgoing message. The incoming messages could be filtered based on similarity with customer profiles in the database and when an outgoing message is sent to a more abstract receiver, we could limit the actual reception to the interested parties only. Most likely, these use cases would also benefit from collaborative filtering described in the next section.

7.4.3 Collaborative Filtering

Collaborative information filtering offers suggestions according to similarity between users and items. Software like email, calendar and social bookmarking often make use of this type of techniques. [4]

Traditionally, collaborative filtering is used to gather and analyze information from users' behaviors instead of using properties from items, in other words, it is utilized to measure similarity between items by taking other users' preferences and actions in relation to the items into account. [17] For example, collaborative filtering could be explained as follows. If the preferences or characteristics of a single user X are similar to the ones of the members of a user group A, then the system will to recommend items to user X if they have been appreciated, i.e. bought, used, read, and so on, by the members of the group A. In our Cloud Communication Service, the message merging will probably benefit from the use of this technique. We could, for instance, add or propose users to groups and decide what the content of a message is about based on similarity between its sender and other users of the system.

7.4.4 Knowledge Based Recommendation

Recommender systems are usually classified into categories based on the technique used. Besides the content based and collaborative type introduced above, there is another type, namely, the knowledge based recommender system. This type of recommender system firstly sets up a knowledge foundation which consists of a model of the processed items.

This model can, for instance, consist of users, business items, etc. The system then makes recommendations through reasoning with the data combined with the model. If, for instance, users and items are matched or certain requirements are fulfilled then matching items are proposed.[5] According

to [9] Knowledge based recommendation should be preferred in marketing over other recommender systems. Some features of this type of recommender system include

1. Simplicity: a large amount of data is not necessarily required in the knowledge based type.

2. Quickness: new users with a detailed personal profile could receive recommendations at once.

3. Humanity: the system knows what the user needs and why the user needs this item.

A concrete example would be a system which matches users with products which are for sale in a web shop. A new customer fills a short questionnaire during his registration on the website. From these data preferences are extracted and stored in a profile in the database. This step is called *data collection*. Next, there is the *knowledge foundation*. This knowledge foundation will contain the products and their associated tags. The tags are given in accordance with the products' attributes and popularity.

With the Knowledge Foundation and the profile information in place it is possible to generate recommendations for the customer. This recommendation process is often augmented with the visiting record of the customer.

7.5 Implementation and User Interfaces of the Cloud Communication System

As mentioned in the introduction, the primary goal of the cloud software program is to improve the performance of the Finnish industry. Therefore, in parallel with the efforts to create scientific artifacts related to the business case, we also worked on a concrete implementation of at least part of the ideas. The scientific parts, including the theoretical framework, described above are used as guideline for further extension.

In this section we show some of the user interfaces for channel preference management and multichannel communication coordination which are in use in the application which is under development at Steeri Oy.

An abstract overview of the components of the service are depicted in Figure 7.3. The framework described in section 7.3.1 overlaps greatly with the service presented here. A short description of its key elements follows.

The *Outbound communication service* enables dynamic content creation. It is also used to set channel specific tracking mechanisms. In section 7.4.1

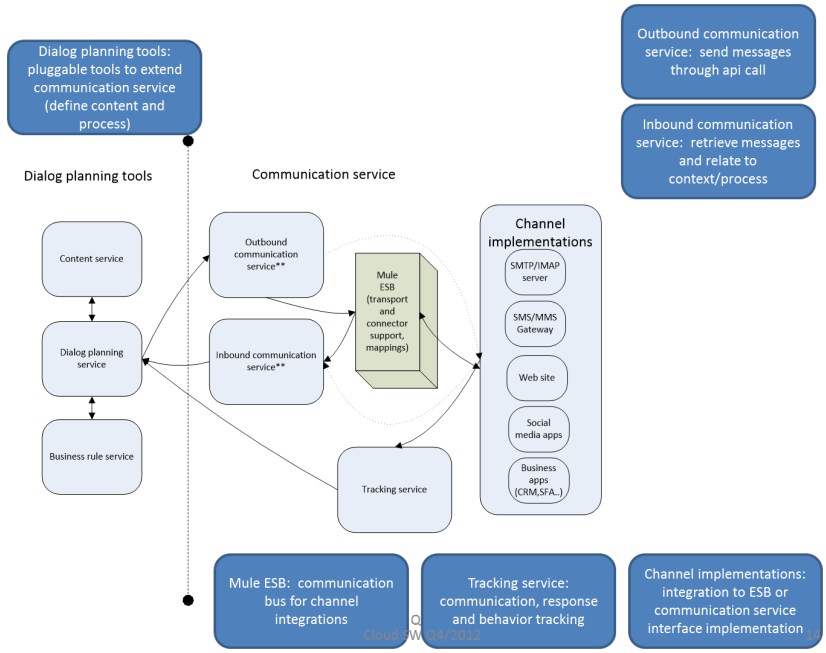


Figure 7.3: Overall architecture

information filtering and recommender systems were discussed. These technologies aim at improving the experience the customer gets from the Outbound communication service.

The *Inbound communication service* maps the incoming communication to the right customers and communication rules. It is in this component where the techniques described in sections 7.3.3 can be used in the future. The automatic recognition of the meaning of the messages could improve the handling efficiency and its correctness.

The *Mule Enterprise Service Bus (ESB)* [1] and *Channel implementations* are technical tools for implementing the set of interaction channels. Thus, they play a key role in the service, but contain only little or no logic at all from the communication perspective. The logic resides at the level that understands the customer behavior, communication rules and is able to relate the dialog conducted to a specific set of company goals and plans.

To be able to orchestrate the communication with the customer a set of dialog planning and coordination tools has been devised. The key part of

[1] http://www.mulesoft.org/

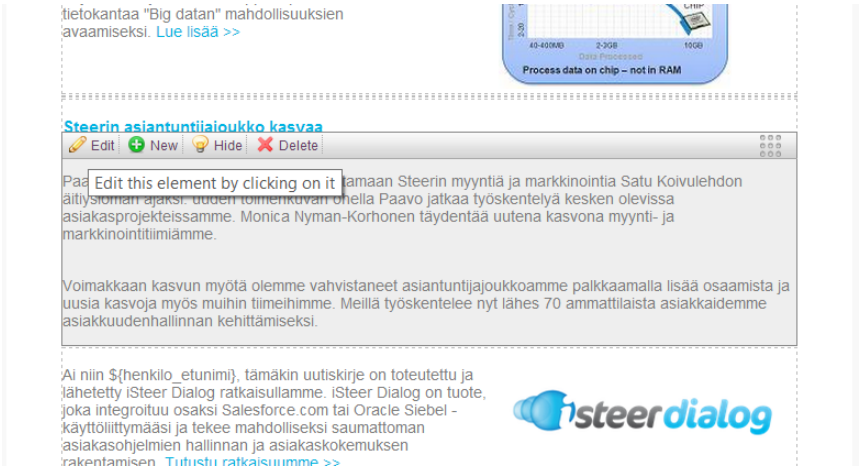


Figure 7.4: Defining web content from content blocks

these tools is the *Content service*, which enables writing channel independent and specific content blocks. The *Content blocks* relate to specific objectives the company has for its communication. They can, for example, provide offers for customers, inform about new services, or suggest ways of utilizing purchased products. The actual implementation could range from a short text message delivered to the mobile phone to a full story on how to present the information in a phone call. A user interface for the definition of web content for a Content Block is shown in Figure 7.4.

Using the *Business rule service* the company can define communication plans, i.e. it can create rules about situations in which it wants to take actions and define which customers should be involved. An example of how a business rule could be created can be found from Figure 7.5. These rules can be applied upon actions and communication from the customer or they can be instigated by goals the company wants to proceed in general.

The *dialog planning service* provides means for defining the multichannel communication process. It connects business rules and content together, and also enables the definition of the steps in the communication process. Each step may have its own rules. An interface for editing the communication steps can be seen in figure 7.6. Different actions can be performed depending on the communication channel as shown in the screenshot in Figure 7.7.

The dialog planning service also provides tools for customer preference management and managing communication rules from a specific customers point of view. Managing customer preferences may take place on channel

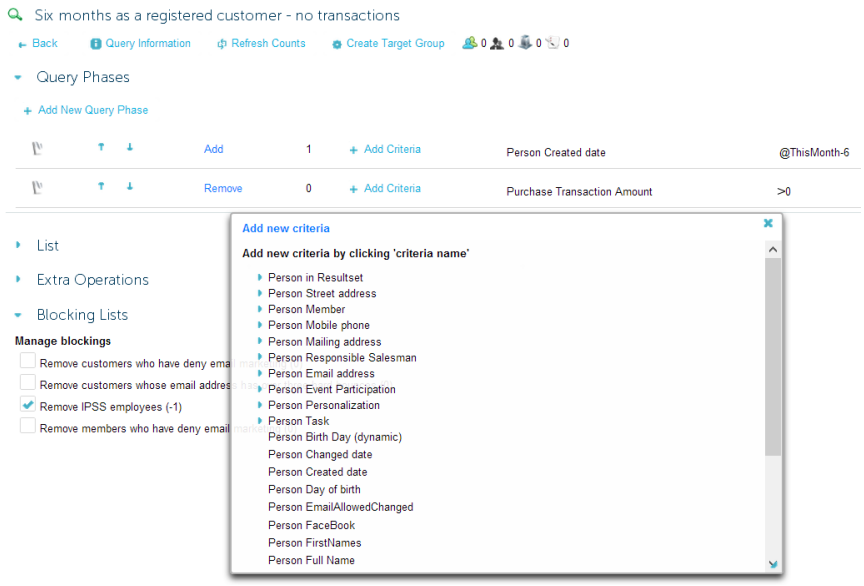


Figure 7.5: Interface for the query phase of a business rule. This query consists currently of two phases: In the first phase all persons which have been created 6 months ago (@ThisMonth-6) are added to the result set. In the second phase all persons who have a transaction amount greater as 0 are removed.

level and based on the type of communication topics one is interested in. The customer is responsible for prioritizing topics of interest and opting in to optional customer care processes. Customers also have the possibility to cancel their subscription to mailings they do no longer which to receive.

Communication plans are run at a scheduled interval. As a result, it is possible to show which customers will receive which information based on their current profile and behavior, as well as the communication rules. As in shown in Figure 7.8, it is possible to see which planned actions would be targeted to which customers in the future. Furthermore, it is possible to define rules for managing future conflicts like for instance overwhelming the customer with too many messages, possibly handling conflicting offers in different channels, etc... This is illustrated in Figure 7.9. The techniques used here are concrete examples of the methods for filtering and recommendation described in section 7.4.

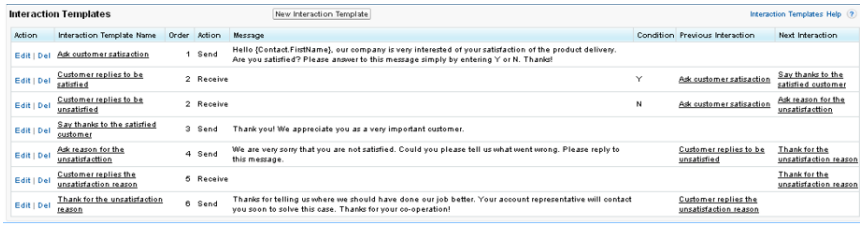


Figure 7.6: An interaction template for different communication steps and conditions

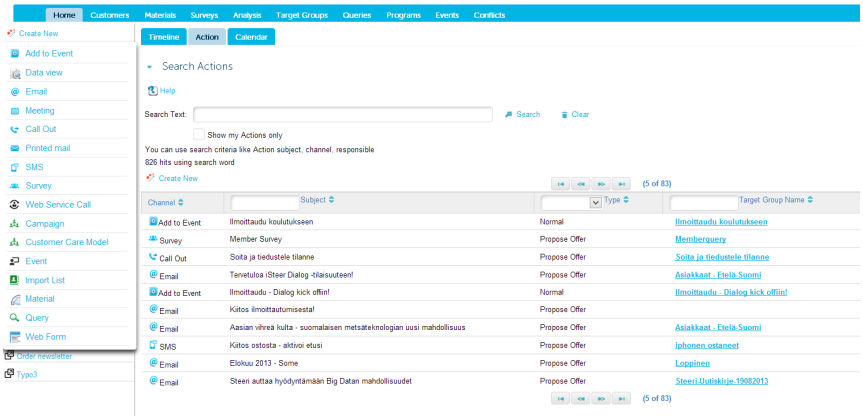


Figure 7.7: Defining actions to different channels

Figure 7.8: Future communication view from one customer's perspective

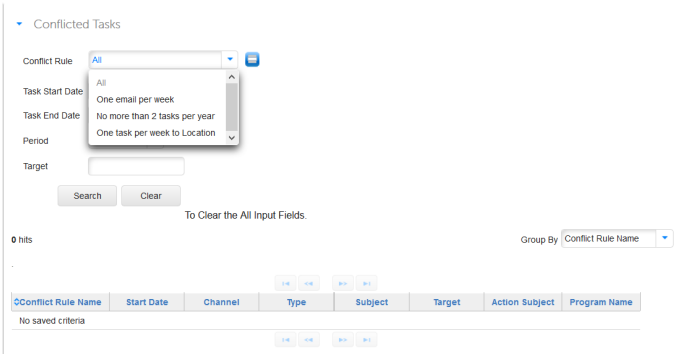


Figure 7.9: Conflict manager UI for managing communication hygiene rules

7.6 Conclusion

Enterprises have often a multitude of systems in place to communicate with internal and external partners, like customers, students, and employees. The communication is usually fragmented over these systems and it is hard to obtain an integrated communication experience. In order to get closer to this target, we proposed the Cloud Communication System (CCS). This system, which is delivered as a SAAS, uses cloud computing technologies and has been elaborated in the frame of the Cloud Software Program.

In this chapter we presented some of our previous research topics. First we showed a general framework for the communication system which uses a semantic data storage to integrate the data which flows trough the system. Then, we discussed how the finding of an ontology with an optimal quality can be defined. This ontology would be stored in the semantic data storage as defined in the framework. Finally we discussed the CCS as a Big Data problem which works on streams of data with a high velocity and propose an Evolving Knowledge Ecosystem to manage the data stream. The knowledge which the system collects from this stream should be kept within feasible limits. We proposed several approaches to limit the amount of stored knowledge which we refer to as *3F+3Co*. This abbreviation stands for by focusing, filtering, and forgetting + Contextualizing, Compressing and Connecting. The core idea behind the Evolving Knowledge Ecosystem is that we can use methods similar to the mechanics of biological evolution for managing knowledge which dynamically changes over time, i.e. which evolves.

From our current research focus we showed our work on human interaction with the system by showing how research in Information Filtering and

Recommender Systems can help us to solve specific problems in the CCS. In the last section we showed a selection of user interfaces for the management of communication.

Acknowledgments

This chapter was written as a part of our activities in the Cloud Software Program organized by the Strategic Centre for Science, Technology and Innovation in the Field of ICT (TiViT Oy) and is financially supported by the Finnish Funding Agency for Technology and Innovation (TEKES). We would further like to thank the technical staff of Steeri Oy for their excellent work in implementing the first prototypes and the members of the Industrial Ontologies Group (IOG) of the university of Jyväskylä for their support in the research. We would also like to thank the reviewer for helpful comments and improvement suggestions.

References

[1] T. Aruna, K. Saranya, and C. Bhandari. "A Survey on Ontology Evaluation Tools". In: *Process Automation, Control and Computing (PACC), 2011 International Conference on*. 2011, pp. 1 –5. DOI: 10.1109/PACC.2011.5978931.

[2] D. Bawden and L. Robinson. "The dark side of information: overload, anxiety and other paradoxes and pathologies". In: *Journal of information science* 35.2 (2009), pp. 180–191.

[3] N. J. Belkin and W. B. Croft. "Information filtering and information retrieval: two sides of the same coin?" In: *Communications of the ACM* 35.12 (1992), pp. 29–38.

[4] T. Bogers and A. Van den Bosch. "Collaborative and content-based filtering for item recommendation on social bookmarking websites". In: *Submitted to CIKM* 9 (2009).

[5] R. Burke. "Hybrid recommender systems: Survey and experiments". In: *User modeling and user-adapted interaction* 12.4 (2002), pp. 331–370.

[6] M. Cochez and V. Terziyan. "Quality of an Ontology as a Dynamic Optimisation Problem". In: *ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer* (2012), p. 249.

[7] C. Darwin. "On the origins of species by means of natural selection". In: *London: Murray* (1859).

[8] D. Dean and W. Caroline. "Recovering from information overload". In: *McKinsey Quarterly. organization practice* (Jan. 2011). URL: http://www.mckinsey.com/insights/organization/recovering_from_information_overload.

[9] D. DellAglio, I. Celino, and D. Cerizza. "Anatomy of a Semantic Web-enabled Knowledge-based Recommender System". In: *Proceedings of the 4th international workshop Semantic Matchmaking and Resource Retrieval in the Semantic Web, at the 9th International Semantic Web Conference*. 2010.

[10] V. Ermolayev et al. "Big Data Computing". In: *Big Data Computing*. Ed. by R. Akerkar. Taylor & Francis group - Chapman and Hall/CRC, to appear in September 2013. Chap. Towards Evolving Knowledge Ecosystems for Big Data Understanding.

[11] D. Fisher et al. "Interactions with big data analytics". In: *interactions* 19.3 (May 2012), pp. 50–59. ISSN: 1072-5520. DOI: 10.1145/2168931.2168943. URL: http://doi.acm.org/10.1145/2168931.2168943.

[12] J. Järvinen. *Cloud Software Program*. http://www.cloudsoftwareprogram.org/cloud-software-program Last accessed on May 31, 2013.

[13] M. Nagy. "A Multi-channel Communication Framework". In: *ICT in Education, Research, and Industrial Applications*. Springer, 2013, pp. 72–88.

[14] M. Nagy. "On the Problem of Multi-Channel Communication". In: *ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer* (2012), p. 128.

[15] D. Nardi and R. J. Brachman. "The description logic handbook: theory, implementation, and applications". In: *The description logic handbook: theory, implementation, and applications*. Ed. by F. Baader et al. Cambridge: Cambridge University Press, 2003. Chap. An introduction to description logics, pp. 1–40.

[16] D. Peng and F. Dabek. "Large-scale Incremental Processing Using Distributed Transactions and Notifications". In: *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*. 2010.

[17] A. Rajaraman and J. D. Ullman. In: *Mining of massive datasets*. Cambridge University Press, 2011. Chap. Recommendation Systems, pp. 277–309.

[18] F. Ricci, L. Rokach, and B. Shapira. "Introduction to recommender systems handbook". In: *Recommender Systems Handbook*. Springer, 2011, pp. 1–35.

[19] C.-N. Ziegler. "Semantic Web Recommender Systems". In: *Current Trends in Database Technology - EDBT 2004 Workshops*. Ed. by W. Lindner et al. Vol. 3268. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 78–89. ISBN: 978-3-540-23305-3. DOI: 10.1007/978-3-540-30192-9_8. URL: http://dx.doi.org/10.1007/978-3-540-30192-9_8.

8 HTML5 in Mobile Devices – Drivers and Restraints

Antero Juntunen, Eetu Jalonen, and Sakari Luukkainen
Department of Computer Science and Engineering
Aalto University, Espoo, Finland
Email: {antero.juntunen, eetu.jalonen, sakari.luukkainen}@aalto.fi

Abstract–Application stores have played a crucial role in the proliferation of applications for smartphones and other mobile devices. However, web-based mobile applications are challenging the application store model by allowing developers to directly reach the end users. These web-based applications are enhanced by the HTML5 standard, which provides additional capabilities for the use of developers and brings the performance of mobile web applications closer to that of native applications. In this chapter, we analyze the potential of HTML5 and identify drivers and restraints that affect the future of the technology.

Keywords-HTML5, application store, web application.

8.1 Introduction

Today's mobile phone landscape is increasingly dominated by smartphones with advanced computing capabilities and features. Smartphone sales surpassed feature phone (normal phone) sales for the first time in the second quarter of 2013, with smartphone sales accounting for almost 52% of all mobile phone sales worldwide [24]. In the United States, the number of smartphone users already exceeded the number of feature phone users in May 2012, with two thirds of new users choosing smartphones [35]. Currently, the majority of smartphones are sold in emerging markets, with Asia/Pacific which are forecasted account for 65% of all smartphone sales in 2013 [29]. This increase in the technical capabilities of mobile phones has coincided with a

proliferation of applications available on these devices. In fact, the main selling point of mobile phones has changed from the hardware capabilities of the devices to the applications they can run. A significant reason for the abundance of mobile applications has been the emergence of application stores. Application stores have simplified the process of finding and installing the applications for the end users, increasing the demand for mobile applications. In addition, a more open policy by device manufacturers has allowed small developers and hobbyists to develop and publish their applications for mobile devices, thus increasing the supply of mobile applications.

The application store model works with native mobile applications, which are downloaded into the user's mobile phone and stored and executed locally. Native applications can fully use the capabilities of the mobile devices, require no Internet connectivity, and can be distributed through the application store, leading to increased visibility among the end users. However, there are certain problems with the native application model. First, the mobile space is fragmented and native applications are tied to a specific platform (such as Apple's iOS or Google's Android). As a result, a mobile developer targeting a larger user base has to create applications for many different operating systems (OS), significantly increasing the time and resources required in application development. Moreover, fragmentation is an issue even within a single OS, as new versions of an OS may not support old applications. Second, application developers are tied to the revenue sharing terms set by the application store provider. These terms (typically a 70/30 % split for the developer) may not be suitable for all developers and all applications. Third, while the performance of mobile devices has increased, native applications are still limited by the constraints of the devices such as limited computing resources and battery power.

These issues can be partially addressed by using web-based mobile applications instead of native applications. OS fragmentation can be addressed if the user can access the web-based application through a standard mobile browser, forgoing the need for the developer to tailor the application to each platform. Web-based applications can also bypass the revenue sharing constraints of application stores, with the developer establishing a direct billing relationship with the end user. In addition, the Mobile Cloud Computing (MCC) model can help address the hardware limitations of mobile devices by running a part of the computation in the cloud. MCC can be used with pure web applications [31] or with native applications, the processing of which is partly offloaded into the cloud [30]. MCC is also sometimes used to describe a mobile device cloud [39], in which mobile devices form a cloud by pooling their resources.

Web-based mobile applications are enhanced by the HTML5 standard,

which provides some of the features from traditional desktop-style software to the browser. HTML5 is currently being developed by two standards bodies, the Worldwide Web Consortium (W3C) [49] and the Web Hypertext Application Technology Working Group (WHATWG) [27]. Both standards are still in a draft stage, with the WHATWG standard being the more fast-changing or fluid of the two. Support for HTML5 is predicted to grow from 336 million mobile phones with HTML5 browsers sold in 2011 to one billion sold devices in 2013 [42], while further estimates put the number of mobile phones with HTML5 browsers in 2016 to 2.1 billion devices [1]. Among notable recent developments for the HTML5 technology is Mozilla's new operating system for mobile devices called Firefox OS [33], which has HTML5 and web technologies at its core. There are no native applications in Firefox OS, because all applications in the operating system are web applications.

In this chapter, we used exploratory research to examine the HTML5 technology and evaluate its potential. Our research goal was to identify drivers and restraints that affect the technology evolution of HTML5.

We based our work on general literature of HTML5, which was chosen by identifying academic articles focusing on HTML5 and mobile applications from databases ScienceDirect, ACM Digital Library, ProQuest ABI/Inform Complete, IEEE Xplore Digital Library, JSTOR, EBSCO Business Source Complete, and Google Scholar. Due to the scarcity of academic articles addressing HTML5 and mobile applications, we supplemented this material with an expert interview with a representative of a large European mobile network operator and with a more general web search. In order to avoid becoming overwhelmed with the available data, we used our own research framework as a research focus [16], basing the research framework on relevant existing business literature. We analyzed the literature using the research framework, which allowed us to classify the data within the different dimensions of the framework. These dimensions were then compared with the research target, which produced the drivers and restraints of HTML5 under each dimension.

The remainder of the chapter is structured as follows: Section 8.2 describes the framework and its theoretical background and Section 8.3 provides an overview of the HTML5 technology and Firefox OS. We analyze HTML5 using the framework in Section 8.4, summarize and discuss the results in Section 8.5, and give our conclusions in Section 8.6.

8.2 Theoretical Background

8.2.1 Technology Evolution

Industries evolve through a sequential development of technology cycles. These cycles are initiated by technological discontinuities [4] that emerge through scientific advance or through a unique convergence of existing complementary technologies, which eventually substitutes existing products [3]. At some point, diminishing returns begin to surface as the technologies start to reach their limits and new, substitute technologies emerge [4]. The threat of substitute products depends on several factors including relative price, new features and added value, performance, and switching costs [38].

The success of many new entrants has lead to coining a phenomenon called the "attackers' advantage". This term refers to those new entrants who are better than the incumbents in developing and commercializing emerging technologies because the new entrants are smaller in size, have limited path-dependent history, and are not commitment to the value networks of the previous technology [10, 23]. New entrants can be successful despite the incumbents' greater resources and experience with the existing technology. However, industries have barriers to entry, which protect the profit levels of the incumbents and hinder the market entry of new entrants. Barriers to entry are unique to each industry and include factors such as cost advantage, economies of scale, brand identity, switching costs, capital requirements, learning curve, regulation, access to inputs or distribution, and proprietary products [38].

Christensen [9] states that the incumbents improve their technological performance on an existing trajectory and finally exceed even the most demanding customers' needs. Simultaneously, new, more cost-effective technologies are developed by new entrants, first for the needs of the customers of other industries. These new technologies start to increase their market share among less-demanding customer segments and will later enter the existing mainstream market. Christensen refers to these technologies and the related innovations as 'disruptive', which can be seen as an extension to the concept of technological discontinuity. Similar to technological discontinuities, disruptive innovations significantly change the current market structures, customer usage patterns, and value propositions. If the markets of disruptive technologies develop fast, new entrants gain advantages due to economies of scale. If the development is slower, the incumbents will have more time to react on the new entrants.

Rogers [40] considers the most important factor affecting innovation diffusion to be the relative advantage (price and performance) over competing

technology substitutes. Among other factors highlighted by Rogers, trialability relates to how easily the product can be experimented with. Easy trialability for the early adopters enhances the diffusion of an innovation. This is also supported by Gaynor [26], who emphasizes the importance of experimentation, especially in times of great market uncertainty and Thomke [48], who stresses the role of experimentation with new technologies.

The product platform is a concept that allows a company to build a series of related products around a set of common components [32]. An industry platform differs from a product platform in that these components are likely to come from different companies called complementors and that the industry platform has relatively little value to users without these complements [25]. Eisenmann, Parker, and Van Alstyne [17] define platforms as products or services that bring together two distinct groups of users in two-sided markets. They consider four different roles in platform-mediated networks: demand-side platform users (end users), supply-side platform users (complementors), platform providers (users' primary point of contact with the platform), and platform sponsors (who determine access to platform) [18]. Platform openness can differ for each role, leading to varying strategies for managing openness.

8.2.2 Research Framework

Based on the above literature review on technology evolution, we created the following framework for the empirical part of this study. The most important factors affecting the technology evolution of HTML5 are summarized in Table 8.1. The 'Added value' category emphasizes the value of the HTML5 technology over existing solutions and focuses on the viewpoints of the main actors end users and application developers. Relevant theoretical concepts in this dimension are added value [38] and relative advantage [40]. 'Ease of experimentation' concentrates on the ability of developers to adopt HTML5 and to use the technology to create new applications and services. Relevant theoretical concepts include trialability [40] and experimentation [26, 48].

The category 'Complementary technologies' examines supporting technologies, which can be especially important in the emergence of technological discontinuities [3] and in the case of platforms [18, 25]. 'Incumbent role' focuses on the roles of major incumbent actors, including device manufacturers, mobile OS providers and mobile network operators. This dimension can be especially relevant when considering the effect of new entrants [23] on the market, particularly in the case of disruptive innovations [9]. 'Technological performance' compares the performance of HTML5 to substitutes, which relates to the concept of a sufficient level of performance [38, 40]. The chosen

Table 8.1: Research Framework

Dimension	Meaning
Added value	The relative advantage over existing technologies
Ease of experimentation	The threshold of end users or third parties (developers) to experiment with new services
Complementary technologies	The interdependence between complementary technologies
Incumbent role	The product strategy of existing players
Technological performance	The performance or capability of the technology

categories were considered especially useful for a developing technology and the categories arose from both the literature on technology evolution and HTML5.

8.3 Technology Overview

8.3.1 HTML5

HTML5 is both an evolution of the previous HTML version, but also a response to the change in the way that content is used and viewed on the web. Application developers providing multimedia-rich and interactive services have previously relied on solutions provided by third parties, primarily Adobe Flash, and to a lesser extent, Microsoft Silverlight. HTML5 standardizes some of the core aspects of the previously mentioned technologies, allowing the browser to directly provide those features without the need for additional drivers or plug-ins. These new capabilities also bring HTML5-based solutions closer to the traditional realm of desktop or native applications, thus lowering the barrier between the traditional and web-based solutions [44].

Although HTML5 is a standard itself, it is also used as a blanket term for other related technologies such as Cascading Style Sheets version 3 (CSS3) and JavaScript (JS). Roughly speaking, HTML5 is used for content, CSS3 for presentation and JS for defining the behavior of the other two.

Table 8.2 contains a selection of the most relevant features when considering using HTML5 on mobile devices.

Mobile applications built on HTML5 usually rely on different frameworks for cutting down development time and cost. In general, these frameworks can roughly be divided by the input they take and the end product they pro-

Table 8.2: HTML5 Features [50]

Feature	**Comment**
Multimedia	*<video>* and *<audio>* tags, support for both media formats without 3rd party plug-ins.
Hardware integration	Access to mobile device features such as *GPS*, *accelerometer*, *microphone*, *camera*, etc.
Device adaptation	Modifying the page based on the device's *screen size*, *keyboard type*, etc.
User interactions	Support for *touch* and *speech* interaction, also *haptic feedback* (vibration).
Data storage	Data can be stored *offline* within the *browser* or on the underlying *filesystem*, though there is also a simple *key-value* based *database*.
Network	Cross-domain requests with *XMLHttpRequest*. *Server-Sent Events* or *Push Events* for sending data to HTML5 applications even when the page is not active on the browser. *WebSocket* [21] allows for more efficient data transfer, based on a TCP stream (two-way).
Widgets	HTML5 applications can be run off-line with the *ApplicationCache* feature, but also shared as archive files that can be unpacked and deployed in the same way as more traditional applications as per the *W3C Widgets* family of specifications.

duce. Basic mobile HTML5 frameworks such as LungoJS, jqMobi, Sencha, Jo and others use HTML5, CSS3, and JavaScript, but also offer added library functions that help in the development of the application. The end product is a page, site, application, or any other target the developers were aiming for that is then usually run inside a web browser on the targeted platforms.

PhoneGap is similar to the previously mentioned frameworks, but instead of running applications within the browser of the device, PhoneGap [37] outputs a stand-alone application for the selected and supported mobile platforms. The source is HTML5 and it can include parts of other HTML5 frameworks, JavaScript libraries, or even native code. The end result is an application that runs inside browser view that in turn runs inside the aforementioned PhoneGap stand-alone application or container.

Titanium SDK [5] is another step towards native applications from PhoneGap, as its only input is JavaScript that is then cross-compiled to the selected mobile platforms. The output is platform-specific code and the end result is in a sense a true native application. The limitation of this approach is that

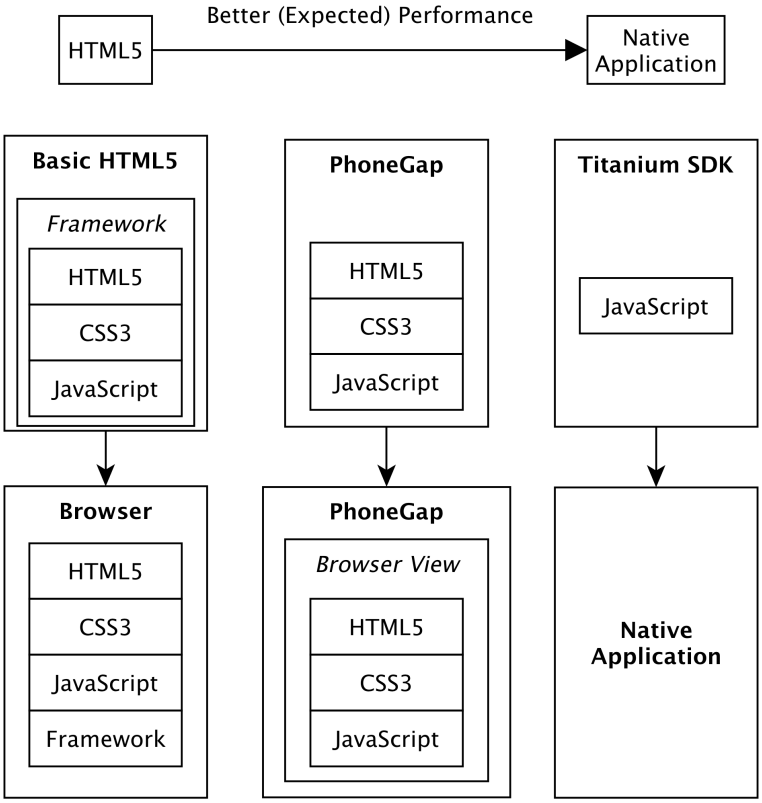


Figure 8.1: HTML5 Frameworks Input/Output and Application Scale.

the framework is restricted to the libraries provided by Appcelerator.

Figure 8.1 displays the inputs and outputs of the different frameworks, and shows how these frameworks fit into the scale between HTML5 applications and native applications.

8.3.2 Firefox OS

Firefox OS is a mobile device operating system built on the Linux kernel and it uses a Gecko-based runtime engine to run Open Web Applications built purely on HTML, CSS and Javascript [33]. By leveraging the web as an application platform, the hope is that developers would not have to write the same application several times for multiple platforms, but only

once for the web with the possibility to install that application for offline use on an end user device. All applications on the Firefox OS platform are web applications, which means the term native application on this platform becomes moot.

Firefox OS interfaces with device hardware and external services through Web APIs. These APIs are the same ones used by the system applications, so in theory every aspect of the operating system is available to external applications as well, though device vendor or other restrictions may apply. Overall, Firefox OS is highly modifiable and as a platform it is convergent to its cousin, the Firefox browser. Applications written for the Firefox browser should run with little or no modifications on Firefox OS as well.

8.4 Analysis

In this section, we apply the research framework of Table 8.1 to HTML5. Section 8.4.1 examines the added value provided by HTML5 to both developers and end users, while Section 8.4.2 evaluates the ease of experimentation with HTML5 from the point of view of third-party application developers. In Section 8.4.3, we analyze the role of complementary technologies, and Section 8.4.4 examines the role of incumbent actors such as application store providers. Finally, Section 8.4.5 examines the technological performance of HTML5-enabled applications.

8.4.1 Added Value

End users can benefit from web-based applications and HTML5 in several ways. First, the users do not need to manually install or update their applications, as is the case with native applications. Because web applications use the mobile browser as the run-time environment, the user always has access to the newest version of the application without explicit installation or update [45]. Second, users who have multiple devices such as mobile phones, tablets, and laptops on several platforms may have to use different applications on different devices. Web-based applications can offer a unified user experience regardless of the device or platform used.

Third, HTML5 provides both platform-specific and custom user interfaces depending on the device in question or the needs of the application. Applications developed directly for a certain phone model by leveraging its native programming interfaces and programming model can have a better usability than traditional web applications. Native applications can take a stronger advantage of the user interface controls such as certain gestures on

touch screen devices and the placement of control buttons around a display in keypad-operated mobile phones. In addition, users may not be comfortable in using and installing applications the user interface of which greatly differs from the rest of the mobile device, thus creating confusion. However, HTML5-enabled applications can offer better interactivity for the user and more closely mimic the behavior of native applications. In addition, many JavaScript frameworks provide UI elements designed to imitate the look and feel of native applications of a particular platform. For example, HTML5-enabled applications can be designed not to look like web pages by disabling the traditional web page elements such as tabs, URLs, and back/forward buttons [46].

Fourth, web applications normally require a network connection to function properly, but HTML5 provides offline data caching, which enables applications to be developed to function at least partially even when the connection is unavailable. In addition, the application can also function completely offline and to only exchange data with the host server when required.

In addition to end users, developers can benefit from HTML5-enabled and web-based applications in multiple ways. First, web applications can help overcome the fragmentation of the mobile space. Application developers only need to develop one web application that will be used through the browser rather than provide applications for each platform they want to target. This cross-platform development not only significantly reduces application development costs, but precludes the need to have the programming expertise necessary to develop an application for each platform. One major driver for the development of web technologies has always been the interoperability of different operating systems and computer architectures, thus making it a natural choice for developing applications for the heterogeneous mobile environment. Increasing diversification and uncertainty about the future direction of the mobile operating system market may drive more developers to adapt web technologies in developing their applications.

Second, developers may find it financially lucrative to challenge the revenue sharing terms of the applications store monopolies. With the application store providers typically retaining 30% of the application revenues, developers may wish to bypass the application store and sell the application directly to the end users. This approach has been used by content providers such as Financial Times, who withdrew their application from Apple's AppStore and launched an HTML5-enabled application [14], eventually resulting in increased revenue and more subscribers [41].

8.4.2 Ease of Experimentation

The trialability or ease of experimentation of HTML5 depends on how easy it is for application developers to start using the technology and how HTML5 affects the software development process. First, use of HTML5 builds on existing knowledge with web technologies such as JavaScript, meaning that web developers should be comfortable moving to HTML5. In general, web development is considered more economical and faster than traditional software development as the technologies and tools are usually easier to start with than their native counterparts. With web development, the results of your work are almost immediately visible, unlike native development where more additional code and effort is needed to produce notable results. This all leads to a lower threshold for HTML5-based software development and increases the ease of experimentation of the technologies in question.

Second, some of the intrinsic advantages of running applications on the web include the ease of deployment as well as the speed and ease of updating the applications. Application stores are typically vendor-locked to their respective operating systems, which means that developers have to first develop, compile, and submit the application to the store before it can be downloaded by the users. On the other hand, a web application is usually distributed as source code that the browser interprets, resulting in a more dynamic process. Code can be updated on the fly, with users downloading the changes while they are using the application. Thus, the process of software deployment is greatly hastened. For more popular applications, this means having the necessary server hardware and bandwidth to host your application, instead of relying on an application store's hosting service, a trade-off between the ease of deployment and the ease of upkeep [11].

Most of the intrinsic advantages of web development should apply to Firefox OS as well, but as the operating system is relatively new there are unfortunately some platform-specific concerns to address. These are mainly related to the fact that the roots of Firefox OS are in the Firefox browsers for desktop computers, but Firefox OS is intended to run on more restricted devices. This had lead to some changes, for example in the way files are handled. On the other hand, due to the way Firefox OS applications are built, the development tools for it and general web development are practically identical. As an innate benefit, the Firefox OS application should translate to a generic web application for other mobile devices with little extra work. Naturally the reverse, transforming an existing application to a Firefox OS one, is also possible.

8.4.3 Complementary Technologies

As was already mentioned in section 3, HTML5 is a blanket term for several related web standards and technologies, and HTML5 together with CSS3 and JavaScript represent the complete package or idea that is HTML5. Underneath the surface there are several related APIs to provide the multitude of functionalities that are currently available on mobile devices, but ultimately it is up to the browser to implement these standards. A listing of how mobile browsers support different HTML5 features is available on the web [22] and no mobile browser offers complete as of September 2013. Adequate browser support is a prerequisite for HTML5 adoption and the dichotomy with the platform vendors is that of native applications versus web applications. The platform vendors are striving for a strong ecosystem around their respective platforms, but the ultimately correct path is still unclear. The level of support they want to provide for the two options, native applications and browsers/HTML5, is a balancing act.

8.4.4 Incumbent Role

The current native application market is mainly controlled by the platform vendors and their ecosystems, the largest two being Google (Android) and Apple (iOS) [28]. Their respective native application stores are the primary way in which users on these platforms find, download, and update their applications. As native applications already have access to the hardware features of the devices through a multitude of APIs, platform vendors also provide help and documentation on using these features as well as the necessary software tools to develop the applications. The only limiting factor for developers with this model is the vendor lock-in caused by the vendors themselves.

The problem of how to cross-develop applications for multiple platforms and ecosystems has been left unanswered and this opening is something HTML5-based solutions are able to exploit. The wide variety of HTML5-based frameworks allow for solutions based on it to adapt to a fairly large number of situations. Even if a particular method of application deployment might prove in the long run to be unsuccessful, it is more likely that a large quantity of the development already done can be transferred to another application, minimizing the amount of waste in development.

One of the main benefits of the application store model comes from the simplicity of monetizing applications, but at the same time it ties down the application developer, both technically and legally. To be able to publish in an application store, the application developer has to follow the guidelines set by the store and also accept the 70/30 % revenue split. With HTML5,

the application itself can be hosted as a traditional website, but it can also be deployed as a more traditional application, via a mobile platform vendor's application store, or in some cases even by simply downloading it from a website. This wider set of options for deployment for the applications publisher might be a key factor in switching over to HTML5.

The areas where HTML5 is lagging behind its native competitors are performance, ease of use, and added value. The performance aspect is viewed in detail in Section 8.4.5 , but the ease of use and added value factors have to also be addressed. If the benefit proposal is clearly biased in the favor of the application developer, that is, the user experience of an HTML5 application compared to a native application would only be marginally better, the same or even worse, then the user has very little incentive to switch over from an existing native application unless forced. For new services with no existing native applications the situation might be simpler, but overall, if the HTML5 user experience cannot reach the level of native applications, it can be considered a serious hindrance for its wider adoption on mobile devices.

Although many Mobile Network Operators (MNOs) have launched their own application stores, these initiatives have gained little success compared with those of handset manufacturers or operating system providers. Moreover, because MNOs normally provide subsidized handsets for numerous platforms, creating and maintaining an application store for each platform can be costly. With web-based applications opening up the application store model, MNOs may be able to reach new relevance by mediating between content providers, advertisers, and end users. MNOs could bring value to these actors by helping users find relevant applications, and by providing application developers with more flexible billing models and revenue sharing than current application store providers. In addition, MNOs could utilize their access to anonymous user data like device type, location, and behavioral data to better target applications for users. Similarly, because HTML5 uses the client-server paradigm, such applications are well suited for utilizing open APIs offered by MNOs [12]. These APIs can provide developers with additional capabilities such as user location, billing, and SMS messaging [31] while fitting seamlessly to the client-server paradigm of HTML5.

Mozilla's Firefox OS has the potential to drive the proliferation of HTML5-enabled applications if it becomes a viable alternative platform to Android, iOS, Windows Phone, and other platforms. Several MNOs have taken an active role in developing the operating system and they see potential benefits in the Firefox OS platform. These benefits include the openness of the platform, the ability to substantially contribute to its development, and the customizability of the operating system. Among other actors, MNOs can also develop their own applications for Firefox OS, including NFC appli-

cations, which unlike in iOS or Android, can access the secure element from the browser. Another key actor related to Mozilla is Google, which provides the majority of Mozilla's funding in exchange for Mozilla setting Google as the default search engine in Firefox browsers [43]. This currently mutually beneficial relationship could change in the future, if Firefox OS were to have an effect on Android's market share. It is possible that even if Firefox OS cannot attain a substantial market share, it could have a significant positive impact on the development of mobile browsers and the proliferation of mobile web applications. This could ultimately lead to the mobile web application model overtaking the application store model in popularity.

8.4.5 Technological Performance

As previously mentioned, HTML5 is still a work in progress, even though some of the new features it defines have already been implemented in certain browsers. Problems with mobile use include the adaptation of the web applications view to the quirks of the particular platforms conventions, considering for example the extra buttons on an Android device compared to an iOS device. Browser compatibility is another issue, as not all browsers treat the same piece of code in the same way. Browser performance must also be taken into account, as the browser itself adds another layer of complexity between the application and the hardware. Even though there has been progress in the last few years on the execution speeds of JavaScript on browsers compared to native code, JavaScript still has to be downloaded, parsed, and only then can it be executed, adding a time penalty.

There are a few frameworks available to address some of the issues, such as the previously mentioned PhoneGap and Titanum SDK. Both frameworks allow access to most of the internal APIs of their supported mobile platforms, but provide them in a platform-independent way for cross-platform development and deployment. As Firefox OS is essentially a web browser, it also suffers from the same issues as all other browsers when running HTML5 applications. Fortunately, performace improvements to the Javascript runtime engine for the Firefox browser should also translate downstream into Firefox OS (cross-platform improvement). These frameworks, HTML5, Firefox OS, and web-based applications in general exchange application execution smoothness and responsiveness for flexibility and a more universal deployment scheme when compared to native applications.

Table 8.3: Drivers and restraints of HTML5

Dimension	**Driver**	**Restraint**
Added value	Cross-platform compatibility (D1)	User experience compared to native apps (R1)
Ease of experimentation	Cheaper, more flexible development and deployment (D2)	
Complementary technologies		Browser support (R2)
Incumbent role	No reliance on restrictive policies (D3), Flexible revenue models (D4)	Infrastructure and marketing expenses (R3)
Technological performance		Performance compared to native apps (R4)

8.5 Summary of Results and Discussion

8.5.1 General Results

In our analysis, we identified several factors that act as both drivers and restraints to the diffusion of HTML5. Table 8.3 displays these drivers and restraints and how they relate to the theoretical framework dimensions presented in Table 8.1.

The added value [38] or relative advantage of HTML5 over substitutes [40] is crucial for the success of the technology. In the case of HTML5, the most important driver in this dimension is cross-platform compatibility (D1), which allows developers of mobile web applications to more easily target various mobile platforms, thus minimizing the negative effects of mobile OS fragmentation. On the other hand, the end user benefits of HTML5 and web applications are somewhat limited and the user experience (R1) of these applications can, in fact, be inferior compared to native applications.

Ease of experimentation [26, 48] or trialability [40] in this context refers to the ability of mobile developers to adopt HTML5 and to develop applications using HTML5. An important driver is the cheaper and more convenient development of HTML5 applications (D2) compared to native applications. In addition, the deployment of these applications is much more flexible, and the developers can choose to use different HTML5 frameworks (see Figure 8.1), allowing them to tailor the application deployment to their needs (D2).

Complementary technologies can play an important role in the emergence

of technological discontinuities [3] and in the case of platforms [18, 25]. In the context of HTML5, the most important complementary technology is mobile browsers, and their support for HTML5, which is still incomplete (R2). The crucial issue is the incentives that mobile browser providers have in developing the browsers and the effectiveness of browser standardization.

The role and actions of incumbents is important in determining what effect new entrants can have on the market [23], especially in the case of disruptive innovations [9]. In the case of HTML5 and web applications, the most important incumbents are the current application store providers or platform vendors, such as Google and Apple. Their current restrictive policies on application approval (D3) and revenue sharing (D4) can function as an incentive for developers to move to HTML5 and web applications. On the other hand, the platform vendors provide an infrastructure for the deployment of applications and a marketing venue for application developers, which would not be available for pure web applications (R3).

A sufficient level of performance [38, 40] of HTML5 is a precondition for the success of the technology. Mobile web technologies, such as the execution of Javascript in browsers, have seen considerable progress, but mobile web applications still suffer from a performance gap compared to native applications (R4), which can in part lead to a worse user experience. However, these performance issues can partly be reduced by opting for a hybrid solution between native and web applications, using frameworks such as PhoneGap.

Not all the drivers and restraints presented in Table 8.3 are necessarily relevant to all kinds of actors in the value network of HTML5 applications. Table 8.4 evaluates the importance of the drivers and restraints of HTML5 for three different actors: large "enterprise" developers, small developers which can mean small companies or individual developers, and end users. Cross-platform compatibility (D1) is especially important for small developers, who may not have the resources or skill sets necessary to develop their application for multiple platforms. Enterprise developers typically have the capability to target multiple platforms, but they can still receive substantial savings from the cross-platform support of HTML5 applications. In addition, end users can benefit from a more unified user experience by having access to the same application on multiple platforms.

Cheaper and more flexible development and deployment of applications (D2) is relevant to all developers, but especially to smaller developers who can easily start to utilize web development tools and practices. Being able to bypass the rigid policies and revenue models of applications stores (D3, D4) can be particularly important for enterprise developers, allowing them more freedom to experiment with different pricing models and giving them the opportunity to retain more of their revenues. Smaller developers may

Table 8.4: Drivers and restraints of HTML5 for different actors

Drivers / Restraints	**Enterprise Developers**	**Small Developers**	**End users**
D1: Cross-platform compatibility	Medium	High	Medium
D2: Cheaper, more flexible development and deployment	Medium	High	N/A
D3: No reliance on restrictive policies	High	Medium	N/A
D4: Flexible revenue models	High	Low	N/A
R1: User experience compared to native apps	High	High	High
R2: Browser support	Medium	High	Medium
R3: Infrastructure and marketing expenses	Low	High	N/A
R4: Performance compared to native apps	High	High	High

find it more beneficial to utilize the application store model, which limits the infrastructure and marketing expenses they have to face (R3). Thus, the application store model offers an easier way to monetize applications, especially for smaller developers. The importance of browser support (R2) highly depends on the application in question and whether it uses browser capabilities that are not equally well supported among all mobile browsers. Small developers may be quicker to seek a competitive advantage by utilizing new browser features, which may be supported differently in different browsers. The user experience (R1) and performance (R4) of HTML5 applications can be considered very important to end users and, by extension, to all developers who consider these factors as key elements in the value offerings of their applications.

8.5.2 Practical Examples

The adoption of new technologies can be accelerated by the example of successful products or services using these technologies. For technologies aimed at end users, so-called "killer applications" can be instrumental in driving end-user adoption. However, because HTML5 benefits mostly developers and not end users, examples of successful HTML5-enabled applications are more likely to be relevant in demonstrating the benefits of the technology for developers.

The following real-life examples highlight some of the drivers mentioned in Table 8.3 and provide a quick overview of how HTML5 has and currently is impacting the mobile application market.

- (D3) Grooveshark offers a HTML5 client [19], as the native mobile application was pulled from both Google's and Apple's application stores due to ongoing legal issues [8] between Grooveshark and the music labels EMI, Sony, Universal, and Warner.
- (D1, D2) OpenAppMkt [36] is a marketplace for mobile HTML5 applications, with a client available for iOS and Android.
- Mozilla's Firefox Marketplace is the de facto web application store for the whole Firefox platform [34]. It also has installable applications for desktop operating systems, although these applications are running inside a separate Gecko runtime engine process.
- (D3, D4) Financial Times completely switched from the AppStore to an HTML5-based application [14].
- (D1, D2) According to PhoneGap [37] and Titanium SDK [5], several companies and other groups, such as the Wikimedia Foundation, eBay, and NBC, have released applications built on their frameworks.
- (D1) Facebook [20], Amazon (Kindle Cloud Reader) [2], and Dropbox [15] all offer HTML5-based applications for services for which they also provide native mobile applications.
- Firefox OS is a practical example of two distinct factors. First, it shows the extent to which HTML5 and web technologies can be utilized, i.e. they can be used as a mobile operating system. Second, the cross-platform web-application support between the Firefox Browser, Firefox Browser for Android and Firefox OS demonstrates the viability of the web as an application platform.

The examples presented in the list above do not all rely on a particular benefit that HTML5 provides, but accentuate the multitude of new opportunities the technology brings. Financial Times, for example, has taken the route of removing its application completely from Apple's AppStore, while Facebook has done the opposite and deepened its relation with Apple's mobile operating system, iOS [6]. Facebook's iOS application used to be a HTML5 solution wrapped as a native application, but the company released

a completely new native iOS application [13] written in Objective-C in August 2012, citing "when it comes to platforms like iOS, people expect a fast, reliable experience and our iOS app was falling short." Although a single application moving away from HTML5 might not be that significant, it highlights the fact that HTML5 still has issues to solve, mainly related to performance. In December 2012, Sencha released their own HTML5 mobile Facebook client, Fastbook. Sencha claims that their HTML5 client compares favorably in regards to performance and usability with Facebook's own native client [7].

What Facebook and Financial Times have in common is that they both provide a viable HTML5-based application for their service. In the case of Facebook, the HTML5-based mobile version of Facebook [20] is used as a tool to reach a wider audience, that is, users that are not using native mobile applications for one reason or another. From the perspective of Financial Times, HTML5 is used as a replacement for a native application because of the restrictive policies of a single platform provider [41], as Financial Times does provide an Android application [47].

The HTML5 mobile application stores by Mozilla and OpenAppMkt present an attempt to create a cross-compatible application store for all compatible mobile platforms (Open Web Applications). The number of potential customers is naturally higher than in any platform-specific store, but at the same time they suffer from a much larger pool of differing hardware and software combinations. Ensuring an adequate user experience for so many different devices will undoubtedly require more resources, negating a part of the easier development aspect of HTML5 but fully taking advantage of the technology's cross-platform capabilities.

PhoneGap and Titanium SDK both offer an enticing solution to the native versus web-application question. Developers can, to a varying degree, use the same codebase for both versions and leave out native development altogether. In addition, the switching cost on the developers' side to either version in the future is minimal compared to fully porting a native application to HTML5 or vice versa.

8.6 Conclusions

Although HTML5 applications are not equal in performance levels of native applications, the lower cost and cross-platform availability of a web application might prove crucial for vendors or organizations looking to support the largest number of customers possible, without writing platform-specific implementations of their application. Political or financial decisions to opt out

of the mobile platform vendors' ecosystems can also motivate companies and individuals to directly adopt HTML5 and web-based applications. Which actor or actors would provide the largest push away from native applications and towards HTML5 remains to be seen.

Mobile browser support for HTML5 features is a key factor in the diffusion of the technology, and efforts to adopt and integrate HTML5 standards into a growing number of browsers are ongoing. For certain types of applications, HTML5 will surely be a viable option, but at the same time it is unknown if platform vendors would simply halt the development of their own mobile application platforms and let web applications take over.

Firefox OS is an operating system that has no comparable "native applications" when compared to Android, iOS, or Windows Phone. It relies entirely on web applications, thereby elevating the web browser to an operating system. However, there are still open questions regarding the overall viability of a browser as an operating system as well as application performance and end-user acceptance of such a platform. Hopefully, in the near future, Firefox OS will provide a real-world case study into the extent to which web technologies can be extended on mobile devices, and whether it is a viable alternative for most users when compared to more traditional mobile operating systems and their ecosystems.

The final issue is that of end user preference, in which the largest obstacle comes from the fact that if HTML5 cannot offer a level of usability and added value that users currently receive from native applications, then there is no pull from their part to adopt the technology. A level of parity is needed, which requires support from the platform and hardware vendors to open up their systems for the browsers to utilize with HTML5. Taking all this into consideration, performance still remains a key issue, with the native applications having the advantage. However, the performance gap between native and HTML5 applications is shrinking, and the performance of HTML5 applications is already suitable for many end user needs.

To summarize, if the application vendor's main requirements currently are either performance or easy monetization, native applications might be more enticing. If low initial capital expenditure and a wide cross-platform market share are key issues, HTML5 applications are the more attractive choice.

In this chapter, we provided an initial analysis on how HTML5 affects different actors in the mobile phone ecosystem. In the future, more research is needed to clarify the effect of HTML5 especially on the role of mobile network operators and on the potential impact of Firefox OS in the mobile market.

References

[1] ABI Research. *2.1 Billion HTML5 Browsers on Mobile Devices by 2016 says ABI Research*. July 2011. URL: http://www.abiresearch.com/press/3730-2.1+Billion+HTML5+Browsers+on+Mobile+Devices+by+2016+says+ABI+Research (visited on 09/15/2013).

[2] Amazon.com, Inc. *Kindle Cloud Reader*. 2013. URL: https://read.amazon.com/about (visited on 09/15/2013).

[3] P. Anderson and M. L. Tushman. "Technological Discontinuities and Dominant Designs: A Cyclical Model of Technological Change". In: *Administrative Science Quarterly* 35.4 (Dec. 1990), pp. 604–633. ISSN: 00018392.

[4] P. Anderson and M. L. Tushman. "Technology Cycles, Innovation Streams and Ambidextrous Organizations". In: *Managing Strategic Innovation and Change*. New York: Oxford University Press, 1997.

[5] Appcelerator Inc. *Titanium SDK | Mobile App Development*. 2013. URL: http://www.appcelerator.com/titanium/titanium-sdk/ (visited on 09/15/2013).

[6] Apple Inc. *Apple Previews iOS 6 With All New Maps, Siri Features, Facebook Integration, Shared Photo Streams & New Passbook App*. June 2012. URL: https://www.apple.com/pr/library/2012/06/11Apple-Previews-iOS-6-With-All-New-Maps-Siri-Features-Facebook-Integration-Shared-Photo-Streams-New-Passbook-App.html (visited on 09/15/2013).

[7] J. Avins and J. Nguyen. *The Making of Fastbook: An HTML5 Love Story*. Dec. 2012. URL: http://www.sencha.com/blog/the-making-of-fastbook-an-html5-love-story (visited on 09/15/2013).

[8] BBC News. *EMI terminates Grooveshark streaming deal*. Apr. 2012. URL: http://www.bbc.co.uk/news/technology-17610811 (visited on 09/15/2013).

[9] C. M. Christensen. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Boston: Harvard Business School Press, 1997.

[10] C. M. Christensen and R. S. Rosenbloom. "Explaining the attacker's advantage: Technological paradigms, organizational dynamics, and the value network". In: *Research Policy* 24.2 (Mar. 1995), pp. 233–257. ISSN: 0048-7333. DOI: 10.1016/0048-7333(93)00764-K. URL: http://www.sciencedirect.com/science/article/pii/004873339300764K (visited on 12/10/2011).

[11] L. Corral et al. "Evolution of Mobile Software Development from Platform-Specific to Web-Based Multiplatform Paradigm". In: *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*. ONWARD '11. New York, NY, USA: ACM, 2011, 181183. ISBN: 978-1-4503-0941-7. DOI: 10.1145/2048237.2157457. URL: http://doi.acm.org/10.1145/2048237.2157457 (visited on 06/10/2012).

[12] M. Crossey. *HTML5 - The Catalyst for Network as a Service?* 2012. URL: http://www.telco2.net/blog/2012/07/html5_-_the_catalyst_for_netwo.html (visited on 09/14/2013).

[13] J. Dann. *Under the hood: Rebuilding Facebook for iOS*. Aug. 2012. URL: https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-ios/10151036091753920 (visited on 09/16/2013).

[14] K. Doctor. *FT Declares Independence (from Apple) Day*. June 2011. URL: http://newsonomics.com/ft-declares-independence-from-apple-day/ (visited on 09/15/2013).

[15] Dropbox. *Dropbox Mobile HTML5 Client*. URL: https://m.dropbox.com (visited on 09/15/2013).

[16] K. M. Eisenhardt. "Building Theories from Case Study Research." In: *Academy of Management Review* 14.4 (Oct. 1989), pp. 532–550. ISSN: 03637425. DOI: 10.5465/AMR.1989.4308385.

[17] T. Eisenmann, G. Parker, and M. W. V. Alstyne. "Strategies for Two-Sided Markets". In: *Harvard Business Review* 84.10 (Oct. 2006), pp. 92–101. ISSN: 00178012.

[18] T. R. Eisenmann, G. Parker, and M. W. V. Alstyne. "Opening Platforms: How, When and Why?" In: *SSRN eLibrary* (Aug. 2008). URL: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1264012 (visited on 09/05/2012).

[19] Escape Media Group. *Grooveshark Mobile*. URL: http://html5.grooveshark.com/ (visited on 09/15/2013).

[20] Facebook. *Facebook Mobile HTML5 Client*. 2013. URL: https://m.facebook.com/ (visited on 09/15/2013).

[21] I. Fette and A. Melnikov. *The WebSocket Protocol*. Dec. 2011. URL: https://tools.ietf.org/html/rfc6455 (visited on 09/15/2013).

[22] M. Firtman. *Mobile HTML5 compatibility on iPhone, Android, Windows Phone, BlackBerry, Firefox OS and other mobile devices*. English. Sept. 2013. URL: http://mobilehtml5.org/ (visited on 09/20/2013).

[23] R. N. Foster. *Innovation: The Attacker's Advantage*. New York: Summit Books, 1986.

[24] Gartner, Inc. *Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time*. Aug. 2013. URL: http://www.gartner.com/newsroom/id/2573415 (visited on 09/15/2013).

[25] A. Gawer and M. A. Cusumano. *Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*. 1st. Harvard Business Review Press, Apr. 2002. ISBN: 1578515149.

[26] M. Gaynor. *Network services investment guide: maximizing ROI in uncertain times*. Indianapolis, Indiana: Wiley Publishing, 2003.

[27] I. Hickson. *HTML Living Standard*. Sept. 2013. URL: http://www.whatwg.org/specs/web-apps/current-work/multipage/ (visited on 09/15/2013).

[28] IDC. *Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC*. Aug. 2013. URL: http://www.idc.com/getdoc.jsp?containerId=prUS24257413 (visited on 09/15/2013).

[29] IDC. *Smartphones Expected to Grow 32.7% in 2013 Fueled By Declining Prices and Strong Emerging Market Demand, According to IDC*. June 2013. URL: http://www.idc.com/getdoc.jsp?containerId=prUS24143513 (visited on 09/15/2013).

[30] A. Juntunen, M. Kemppainen, and S. Luukkainen. "Mobile Computation Offloading - Factors Affecting Technology Evolution". In: *2012 International Conference on Mobile Business*. Delft, The Netherlands, June 2012. URL: http://aisel.aisnet.org/icmb2012/9.

[31] A. Juntunen et al. "Innovation in Mobile Clouds - Analysis of an Open Telco Application". In: *CLOSER 2011 - 1st International Conference on Cloud Computing and Services Science*. Noordwijkerhout, The Netherlands, May 2011.

[32] M. H. Meyer and J. M. Utterback. "The Product Family and the Dynamics of Core Capability". English. In: *Sloan Management Review* 34.3 (1993), p. 29. ISSN: 0019848X. URL: http://search.proquest.com.libproxy.aalto.fi/docview/224965799/abstract/138F9D73F246CD524D0/19?accountid=27468 (visited on 09/05/2012).

[33] Mozilla Developer Network. *Firefox OS*. Sept. 2013. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS (visited on 09/15/2013).

[34] Mozilla Foundation. *Firefox Marketplace*. 2013. URL: https://marketplace.firefox.com/ (visited on 09/15/2013).

[35] Nielsen. *Smartphones Account for Half of all Mobile Phones, Dominate New Phone Purchases in the US | Nielsen Wire*. Mar. 2012. URL: http://blog.nielsen.com/nielsenwire/online_mobile/smartphones-account-for-half-of-all-mobile-phones-dominate-new-phone-purchases-in-the-us/ (visited on 09/15/2013).

[36] OpenAppMkt. *OpenAppMkt*. 2011. URL: http://openappmkt.com/ (visited on 09/15/2013).

[37] PhoneGap. *PhoneGap | About*. 2013. URL: http://phonegap.com/about (visited on 09/14/2013).

[38] M. E. Porter. *Competitive Advantage*. New York: Free Press, 1985.

[39] M. Raatikainen et al. "Towards Mobile Device Cloud". In: *Communications of the Cloud Software* 1.1 (2011).

[40] E. M. Rogers. *Diffusion of Innovations*. 5th. New York: Free Press, 2003.

[41] J. Saba. *FT Web-based app more popular than app sold in Apple store*. Sept. 2011. URL: http://www.reuters.com/article/2011/09/22/us-ft-idUSTRE78L49Q20110922 (visited on 09/13/2013).

[42] Strategy Analytics. *One Billion HTML5 Phones to be Sold Worldwide in 2013*. Dec. 2011. URL: http://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5145 (visited on 09/15/2013).

[43] K. Swisher. *Google Will Pay Mozilla Almost $300M Per Year in Search Deal, Besting Microsoft and Yahoo*. Dec. 2011. URL: http://allthingsd.com/20111222/google-will-pay-mozilla-almost-300m-per-year-in-search-deal-besting-microsoft-and-yahoo/ (visited on 09/15/2013).

[44] A. Taivalsaari and T. Mikkonen. "The Web as an Application Platform: The Saga Continues". In: *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. Sept. 2011, pp. 170 –174. DOI: 10.1109/SEAA.2011.35.

[45] A. Taivalsaari et al. "The Death of Binary Software: End User Software Moves to the Web". In: *Creating, Connecting and Collaborating through Computing (C5), 2011 Ninth International Conference on*. Jan. 2011, pp. 17 –23. DOI: 10.1109/C5.2011.9.

[46] Telco 2.0 Research. *HTML5: market impact and telco strategies*. SubHub Site. May 2012. URL: http://www.telco2research.com/articles/EB_HTML5-what-does-it-mean-for-telcos_Summary (visited on 09/15/2013).

[47] The Financial Times Ltd. *Financial Times - The new FT app for Android*. 2011. URL: http://apps.ft.com/android/ (visited on 09/15/2013).

[48] S. Thomke. *Experimentation Matters*. Boston, MA: Harvard Business School Press, 2003.

[49] W3C. *HTML5, W3C Candidate Recommendation 6 August 2013*. Aug. 2013. URL: http://www.w3.org/TR/html5/ (visited on 09/15/2013).

[50] W3C. *Standards for Web Applications on Mobile: current state and roadmap*. Sept. 2013. URL: http://www.w3.org/Mobile/mobile-web-app-state/ (visited on 09/15/2013).

9 Collaborative Coding Environment on the Web: A User Study

Antti Nieminen, Janne Lautamäki, Terhi Kilamo,
Jarmo Palviainen, Johannes Koskinen, and Tommi Mikkonen
Department of Pervasive Computing
Tampere University of Technology
Email: {antti.h.nieminen, janne.lautamaki, terhi.kilamo,
jarmo.palviainen, johannes.koskinen, tommi.mikkonen}@tut.fi

Abstract–Today, techniques made popular by Web 2.0 enable massive cooperation of online users. In the spirit of Google Docs, where multiple editors can cooperate in real time to craft a single document, we believe that it is only a matter of time before software development takes the step towards real-time collaborative online editing and development instead of artificially forced interleaving implemented in version control. First, to study the different aspects of this approach, we have implemented a web-based collaborative programming environment that extends the capabilities of a code editor with numerous features commonplace in social media. Second, to gain the developers perspective to real-time collaborative development, we arranged a week-long experiment where participants composed a web application using our environment and report the results here. Towards the end of the chapter, we discuss the lessons we have learned about real-time collaborative software development. Furthermore, we list the features that developers find essential in such a system as well as compare our results with other reported research.

Keywords–Software development, online collaboration, user study.

9.1 Introduction

Today, software is being developed by teams of programmers who may be globally distributed, work in dissimilar environments and come from various

backgrounds. Contemporary development approaches such as pair programming along with other extreme programming practices [2], global software engineering [13] and collaborative software development [14] illustrate this trend. However, while the main designs and principal guidelines are often crafted in a collaborative fashion in meetings of different kinds, the final design of software artifacts that will later be compiled and run are usually designed and written by individual developers, working in relative isolation from one another and with limited face-to-face or any types of contact. With such an approach, it is not surprising that communication among the developers has been considered a problem for over two decades [23].

Recently, various fields of industry as well as consumer applications are experiencing a paradigm shift towards web-based systems. Web 2.0 allows online collaboration beyond any previous expectations, making the web the first truly global platform for cooperation and collaboration. Fundamentally, Web 2.0 technologies combine two important characteristics, collaboration and interaction. The former refers to the "social" aspects that allow a vast number of people to collaborate and share the same data, applications, and services over the Web. The second, equally important aspect of such technologies is that they enable building web services that behave much like desktop applications with the added benefit that the user does not have to install any software on their computer; any device with a web browser can be used to access the applications. We believe that these two properties are paving the way towards shifting software development to the web. Examples of such systems include Cloud9 IDE[1], Eclipse Orion[2], GitHub[3] and Codeanywhere[4]. From these premises, we have implemented a web-based real-time collaborative software development environment, CoRED, that has been introduced from the technical perspective in [17, 19].

In this chapter, we evaluate collaborative software development with a number of developers who jointly compose a web application using CoRED. With CoRED, several developers can access and modify the same code base in parallel, with no need to use separate version control, following the spirit of real-time collaborative software such as Google Docs. The contribution of this chapter is the introduction of the developers' view to the collaborative browser based integrated development environments (IDE), including both the experiment setup as well as the most important lessons we have learned during the experiment.

The rest of this chapter is structured as follows. In Section 9.2, we in-

[1] `https://c9.io/`
[2] `http://www.eclipse.org/orion/`
[3] `https://github.com`
[4] `https://codeanywhere.net/`

troduce our collaborative web-based development environment. In Section 9.3, we define the research approach and the experiment setup. Section 9.4 presents the data collected and introduces the main results we have learned from the experiment. Section 9.5 further discusses the research questions with discussion and lessons learned. Future work is presented in Section 9.6. Section 9.7 covers work related to our research, and finally, Section 9.8 concludes the chapter.

9.2 Collaborative Development: The CoRED Approach

Moving a software development environment to the web has at least two major benefits. First, developers get rid of the process of installing, configuring and updating the environment for developing, testing, running, and debugging the applications; everything is readily available on the web. Second, the web can enhance collaboration and communication. Most developers are already familiar with using various co-operation tools on the web. Integrating such tools into the development environment can make the collaboration more tightly combined with the process of writing code.

Previously, we have introduced our proof-of-concept system CoRED [17, 19]. In addition to editing the code in real-time collaboration, developers are able to communicate with each other using means familiar from social media. The architecture of our system is typical for a web application. While the client side running inside the browser enables interactions with the system, most of the business logic of the system runs on the server back-end. Our system is implemented with the Vaadin [10] framework. Vaadin provides user interface components, implemented as Google Web Toolkit [22] widgets, that are integrated with the server-side Java[5] code. With this approach, Vaadin applications can be written entirely in Java. The Vaadin framework takes care of connecting the server-side Java code to the user interface components loaded into the browser.

The architecture of CoRED is illustrated in Figure 9.1. The client side contains an open source Ace code editor[6]. Ace is implemented in JavaScript, wrapped in our system inside a GWT component which in turn communicates with the server using HTTP communication channels provided by Vaadin. Other IDE components are implemented in a similar way, some of which are provided by the standard Vaadin package while others are developed by us or other 3rd party developers. Furthermore, the server-side code utilizes Java

[5] `http://www.java.com`
[6] `http://ace.ajax.org`

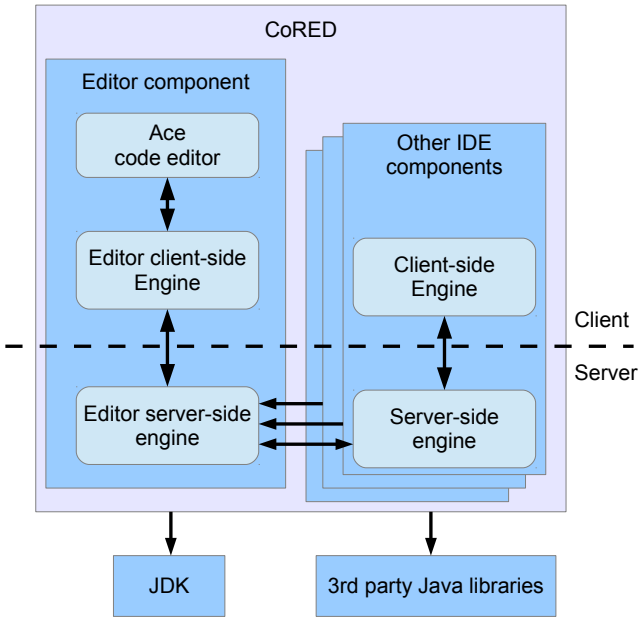


Figure 9.1: The architecture of CoRED

Development Kit for parsing and compiling Java code, and other 3rd party libraries for various tasks. For a more detailed description of the system, the reader is referred to [17, 19].

Figure 9.2 shows a screenshot of a CoRED project that is currently edited by two developers. On the left sidebar there is a list of files in the project (marker with number 1), a deploy button (2), and an avatar of each collaborator currently viewing the project (3). In the bottom left corner, there is a chat box for communication (4). Most of the screen space is reserved for the source file under edit. On the right, the users that view the file are shown (5). Additionally, the cursor positions as well a possible text selection of other users is shown within the opened file. The more rarely used features are accessible via the menubar on the top of the screen. A video clip demonstrating the main capabilities is available at `http://cored.cs.tut.fi`.

CoRED offers various features to support software development, most of which are familiar from traditional desktop IDEs. In addition, some features geared towards collaboration exist. Next, we briefly list the most relevant facilities.

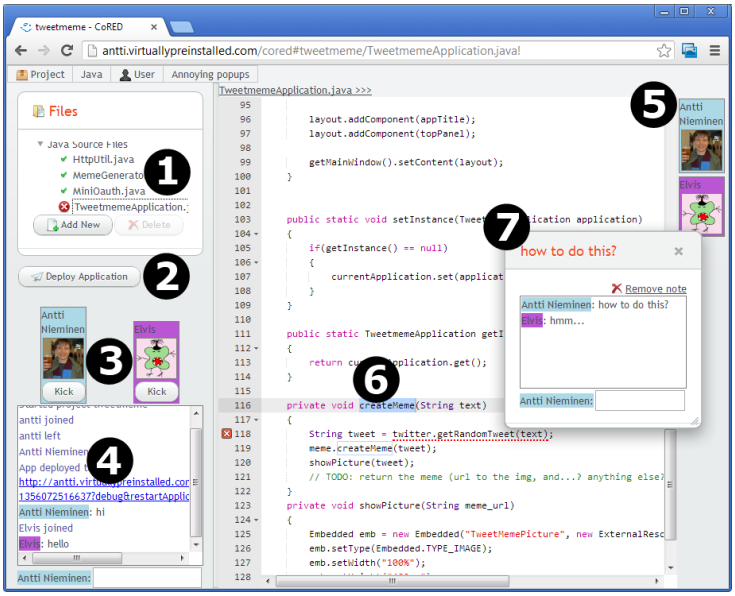


Figure 9.2: Screenshot of CoRED

Project support. In CoRED, projects are used to constitute compilable and runnable systems, similarly to many IDEs. Upon entering the CoRED web page, the user can select to join any of the projects that are currently under development.

Error checking, suggestions and code completions. Another feature seen also in many installable IDEs is the ability of CoRED to pinpoint Java errors in the code. In addition, the editor suggests possible code completions. The suggestions can be invoked in two ways, by using a special key combination or by typing a dot after, e.g., an interface or an object name.

Single click deployment. The developer can compile and deploy the project with a single click. The deployed system will be available online as a service, and a related URL (uniform resource locator) is provided to the user to test the system.

User identity. In order to have meaningful identities for users, it is possible to log in using a Facebook[7] account or a Gravatar[8] avatar. In addition, a guest login with a simple nickname is available.

Collaborative editing. To enable users to edit the code in real-time col-

[7] `http://facebook.com`
[8] `http://gravatar.com`

laboration, we have used the Differential Synchronization algorithm by Neil Fraser[7]. It is a robust collaborative editing algorithm with open source implementations available on several languages, including Java and JavaScript.

Pop up notes and chat. Besides editing the text itself, the users have the option to augment the code with pop up notes in a fashion normally associated with text rather than code. The goal of these facilities is to allow the separation of comments that are related to the actual code, and discussions that developers have over a particular solution. Users can place notes into the code (number 6 in Figure 9.1), and view the related discussion (number 7 in Figure 9.1) by moving the cursor on top of the note. In addition to the inline notes, CoRED also offers a project-wide chat.

Code locking. In CoRED, a single editor can lock a portion of the code for exclusive editing. For example, if a user wanted to make changes to a specific function without anybody interfering, he/she could lock a function by selecting the function and clicking a "Lock for me" button. After locking, the locked area cannot be modified by other users until it is released by the developer who originally locked the area.

9.3 Experiment

We set out to study how developers approached CoRED and real-time collaborative software development through an empirical experiment [1, 15]. In addition to the experiment that ran as a week long code camp, personal opinion surveys [16] on the participants' background and experiences were conducted. During the experiment we also conducted interviews and recorded development data into log files. Also, to get preliminary feedback on CoRED, two short proof-of-concept sessions were held before the actual code camp as precaution for latent problems in the setup.

9.3.1 Research Questions

The main motivation for the experiment was to get a developer's view to real-time collaborative coding and to evaluate how well CoRED would work when developing a web application in small developer teams. Additionally, there was a need to identify the features developers find necessary in real-time collaborative development environment.

The research questions of the study were:

Q1 How do developer teams use a collaborative IDE such as CoRED?

Q2 What features do developers expect from a real-time collaborative IDE?

To find answers to $Q1$, an experiment where teams of computer science students used CoRED to develop web applications was set up. Section 9.3.3 discusses the experiment setup in more detail. Additional data was gathered through personal opinion surveys (pre- and post-camp surveys and individual interviews) discussed further in Section 3.4. Section 9.4 answers to $Q1$ by presenting data from surveys, log files and interviews. Combined with the participants exposure to real-time collaborative programming, the interviews and surveys also provided answers for $Q2$, presented in Section 9.5.

9.3.2 Proof-of-concept Session

In order to manage risks, two short proof-of-concept sessions were conducted before the actual experiment. The purpose of the sessions was to validate the ability of our installation to support the planned number of developers and to find possible fatal bugs or deficiencies in CoRED in order to get any such issues fixed before the code camp. The proof-of-concept session was held twice, on consecutive days, as a part of a Web Services course at Tampere University of Technology. The participants, 11 on each session, were undergraduate software engineering students, allowed to split freely into groups of 1-3 people. The students familiarized themselves beforehand with the very basics of the underlying application platform Vaadin. The session consisted of a brief introduction, an hour of programming, and a small survey. The assigned task was to insert deleting functionality to an existing REST-based image search client.

After the session, many participants were happy about the easy access to IDE without installation, simplicity of deployment of applications, and the work being immediately available for other group members to see. Some considered CoRED to be a good tool for pair programming while some thought that it is probably only suitable for developing small applications.

No critical flaws that would prevent the conduction of the experiment came up during the proof-of-concept session. Anyhow, the participants reported deficiencies such as the lack of multiple editor views open at the same time, no indication that another person is editing the same file, and Java error checking not working in all cases. Fixing all these problems required no changes for the architecture and therefore we were able to fix them before the code camp, although the last one only partly: the user still may need to manually press a "compile all" button in some rare cases. Other comments about CoRED addressed the general immaturity of the tool, lack of IDE features such as "go to method definition" option and lack of proper debug support. These problems were not addressed for the code camp version of CoRED. For the sake of answering the research questions and not wanting

Day	Activity
1	Camp starts. Lectures on the tools and the REST architecture. The teams form and start designing their project.
2	Development starts. Teams use CoRED and are encouraged to work in the same classroom.
3	Development continues. As a short experiment, each team chooses a suitable, socially adept person to act as the project leader. The rest of the team is randomly assigned onto a different project for two hours. The project leaders are located into a separate room from which they communicate with their newly formed team by using CoRED.
4	Distributed development. The developers can choose their working environment freely and are encouraged to leave the classroom.
5	The teams finalize their projects. Presentations of the finished applications are given.

Table 9.1: Daily activities during the code camp

to let our own assumptions on developer needs to influence the study too much, we found it best to leave these points open for the code camp and let the experiment to bring light to where CoRED needs to be matured.

9.3.3 Experiment Setup

In order to extend the amount of data to analyze, we organized a one-week (December 17th to 21st 2012) code camp for implementing a web application using CoRED. The task for students was to design and implement a Vaadin application that utilizes one or more external RESTful [6] APIs of their choice. Table 9.1 presents the flow of the entire camp and the daily alterations that were made to keep the participants' motivation up and to make sure they put the collaborative features of CoRED to full use.

A total of 23 students participated on the first day of the code camp. Out of these, 19 students completed their group project and wrote the required assignment for passing the course. The size of the groups varied from two to four. As the code camp ran during an exam week and right before the Holidays four students dropped out during the first days of the camp. We were able enquire a couple of them for reasons for dropping out, and they mentioned lack of time and personal issues as the main reasons for not finishing the camp.

All the groups were able to produce a working web application, although some of the groups had to leave out certain features they had originally planned. Among the projects produced during the code camp was a system that generates funny meme pictures based on keywords or phrases, utilizing

Day	Survey or interview
1	Survey for investigating background knowledge, preconceptions and expectations
4-5	Six volunteers were picked and interviewed about their experiences in collaborative coding and CoRED
5	A survey for examined the gained knowledge, experiences from CoRED and the different collaborative programming experiences

Table 9.2: Surveys and interviews made during the code camp

APIs of Twitter[9] and Meme Generator[10]. Some of the other projects generated personal pony themed home pages, allowed users to share and browse old exam questions, or view venue information and other data provided by Foursquare[11].

9.3.4 Surveys

During the camp, the participants filled several surveys listed in Table 9.2. Furthermore, the participants got a delegated anonymization code that they used throughout the camp when filling surveys or taking part in an interview. The codes allowed us to track the evolution of the participants' knowledge.

The objective of the before and end of the camp surveys was to monitor if the participants felt that they have learned something or gained new knowledge during the code camp, as well as to get feedback from the system and collaborative way of working. Interviews were used to get more informal feedback and for probing issues the participants wanted to emphasize.

9.4 Results

Next, we present the data gathered during our experiment. The data comes from three main sources: surveys, data logs, and interviews. The surveys provide basic data on the participants as well as their views on CoRED and the whole code camp. The data logs are used to find out how the participants actually worked during the camp. Deeper understanding of the participants views on real-time collaborative development was gathered during interviews.

[9] `www.twitter.com`
[10] `http://memegenerator.net`
[11] `www.foursquare.com`

Skill	Initial	Std. dev. of Initial	"Learning"	Std. dev. of "learning"
Programming	4.4	0.6	-0,1	0.4
Java	3.1	0.8	0.4	0.6
Vaadin	1.6	0.7	1.5	1.1
REST	2.1	1.1	0.9	0.9

Table 9.3: How much the participants felt they learned during the code camp

9.4.1 Survey

Out of the participants, 21 – 16 master's degree and 5 postgraduate students – answered the pre-camp survey. Both the pre-camp and post-camp survey was answered by 14 of them. Table 9.3 presents the quantitative data collected from the questionnaires. In both questionnaires we asked the participants to give themselves a value from 1-5 to describe their knowledge of programming, Java, Vaadin and REST, respectively. The number one stands for no experience and five means that the developer is very experienced. Table 9.3 shows the average initial skill levels of the participants as well as the learning outcome during the course. As expected, the skills with the lowest start values improved the most. The overall result indicates that the participants feel they have learned during the code camp. During group work, people were able to compare their own and other students skills and we speculate that to be an explanatory factor behind the small drop in basic programming skill during the course. Based on observations made on raw data, it seems that the course was most rewarding for the people with basic knowledge of Java and no skills in Vaadin or REST.

In freeform feedback, the following themes repeated several times:

Deployment. "Deployment is hard while people editing." To be able to deploy and test an application, the code had to be compilable. In a group with several people, this requires some coordination and was therefore considered difficult.

Testing tools. "No test tools. Testing was sometimes hard because many people were editing the code at the same time", "No access to server logs" and "Desktop IDE is much better. Real time debugging and much more (shortcuts, smooth, dynamic); only downside maybe not seeing others code".

Version control. "No version control or timeline." Developers often have a need to be able to restore or study older versions of code. However, in CoRED this was not supported except by exporting project versions as zip files.

Group Id	# members	Avg. level	Score (smaller is better)	LOC	Collab. files	Collab. edits (%)	Most active developer (%)
A	3	3.3	3	691	3/11	11	67
B	3	2.7	5	747	5/13	8	63
C	4	2.9	5	1482	3/11	7	31
D	4	3.0	7	800	1/2	13	67
E	3	2.0	10	240	0/1	1	88
F	2	2.4	12	874	3/7	9	93

Table 9.4: Some metrics on the groups and collaboration

9.4.2 Log Data

CoRED recorded data on each groups work to a log file. By analyzing the logs we could get some insight on how the groups collaborated during the code camp. The most important item type recorded was *edits* by the group members. An edit is a small change to the project: addition, removal or a combination of both. The system sent an edit to be synchronized with the shared document after the user had paused typing for 500 milliseconds, in which point it was also recorded to the log. Thus, a typical edit is a line of code or less.

As confirmed by the logs, the participants developed their application mostly during "office hours" when the whole group was present in the same classroom. However, there were some instances when a group member contributed to the project in the evening on their home computer, or even in a bar at night (as one of the participants described in an interview). The chat of the development environment, which was also logged, was not used that much, mostly because all the group members were sitting next to each other most of the time. The only time when the chat was in heavy use was during the two-hour experiment when project leaders were physically separated from their teams.

Some group statistics as well as collaboration metrics are shown in Table 9.4. The first two columns are the group id and the number of group members. The next column is the average skill level of the group as reported by the group members themselves in the initial questionnaire. The projects were evaluated independently by two members of the course staff based on the originality of the idea and the quality of the implementation. The *score* column shows the group score, smaller number being better. The table is sorted by the score, the best group being at the top. The next column shows the total project size, measured as lines of code (LOC).

The remaining columns of Table 9.4 are intended for describing the collaboration aspects of the groups. We considered a file to be collaboratively edited if no one developer is responsible for over 80% of the files content. The *Collab. files* column shows the number of collaboratively edited files compared to the total number of files in the project. If another team member has edited the same file less than 30 seconds before the latter, the edit is considered collaborative. The *Collab. edits* column describes the percentage of the edits that were collaborative. The *Most active developer (%)* column describes the percentage of the characters of the total project produced by the most active member of the group.

As can be seen in the Table 9.4, in all the groups except one, there was one member who wrote over 50% of the total code. As an exception, the work in the group C was notably evenly distributed: each of the four members contributed approximately a quarter of the content. In two of the groups with the worst score, a single person wrote 90% of the code. This could be an indication of either poor collaboration within the group or lack of relevant skills of other project members. The latter conclusion is somewhat supported by the lower self-reported skill levels by the two groups compared to other groups. In both of these groups, the person with most Java experience did most of the coding.

The collaboration level, expressed in Table 9.4 by the *Collab. edits* metric does not seem to have a large correlation to the group performance. In most of the groups, 7-13% of the edits were "collaborative" as defined earlier. The only exception is group E, where there were almost no collaborative edits at all. The group did not produce that much code in terms of LOC, and the majority of the work was done by a single member. In other, somewhat larger projects, even though the group assumably try to structure their work in such a way as to avoid working on the same parts of the project, around 10% of the edits are collaborative. Some of them may be purposeful collaboration between project members while others are accidental. It is not easy to differentiate between the two cases based on the logs.

The column *Avg. level* in Table 9.4 shows the number that is the average of all the skills (Programming, Java, Vaadin, REST) of all the project members. We could make a hypothesis that instead of average, the *maximum* value for each skill level would better predict the performance of a closely working group such as the ones in our experiment. That is, a group with one member with a lot of experience on Vaadin, another with a lot of REST experience, etc. would outperform a group with each member having average skill in each of the categories. In this study, though, we did not find evidence in either direction; the averages of the maximum skill levels (not shown in the table) were quite close to the average level.

To get an overview of the typical collaborative work during the code camp, Figure 9.3 shows a four-hour long period of one of the projects. The edits are shown as data points, different symbols for different project members. The y-axis is the total number of characters in the project and the lines mark the file boundaries. That is, the edits between two lines are in the same file. There is nothing very surprising in how the groups worked on the projects. Most of the time a single person edits a part of the project alone with occasional closer collaboration here and there in some parts of the project. In the logs it was quite rare to see two or more people working on the same piece of the project for very long at a time.

At the end of the log analysis, it should be noted that the logs we analyzed may not contain all the coding work for all the members. We analyzed the logs of only the main project of each group. In addition to the main project, some groups may have used other test projects to try out new features that may be later included into the main project. The possible additional projects are not considered in this log analysis. However, in each of the groups, most of the work seemed to happen in the main project and in any case all the features had to eventually be integrated into it.

9.4.3 Interviews

During the code camp, a total of six semi-structured interviews were held. Interviews were conducted between Wednesday afternoon and Friday midday, taking a half an hour each. In principle, all the interviews followed the same structured pattern with 11 themes. However, the aim of the interviews was to get some free form feedback and therefore we did not try to get answers to all the questions if there was a good flow of conversation. Interviews mostly strengthen the findings of the survey, but also some new themes appeared during conversations. During the interviews it was also possible to get more detailed feedback.

Collaborative editing. Students liked to try out something new and mostly positive feedback was given concerning real-time collaborating. It was reported that with the collaborative IDE working felt more like group work compared to a traditional environment with asynchronous version controlling such as SVN or Git. It was reported that feeling of group work came from the ability to see in real-time what others are doing. Furthermore, they also considered it as a motivational aspect to know that others are able to see what one is actually implementing.

Concerning team size in real-time collaboration, the interviewees reported that a group of four people is too large for working on an ad-hoc basis. Therefore some structure for working was needed. In one group of four

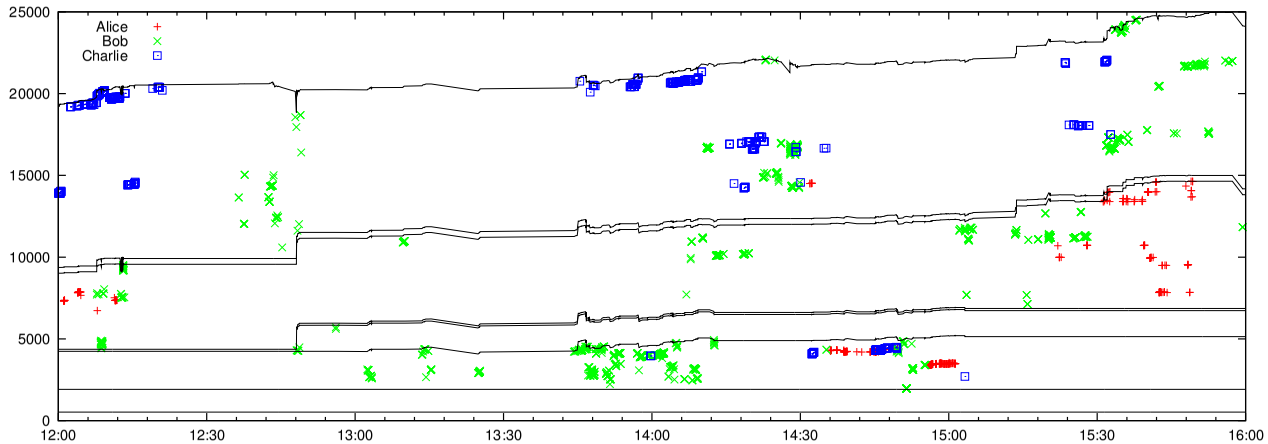


Figure 9.3: A four-hour long piece of a log of one of the projects. Edits are marked with different symbols for each developer, anonymized as Alice, Bob and Charlie. The vertical axis represents the character location where the edit has been done. The lines are file boundaries i.e. edits between the same lines are on the same file. The order of the files is arbitrary.

people, the problem was solved by splitting into two minigroups with tasks of their own. One example was that the less experienced half implemented user interface while the other half was doing business logic. Furthermore, in the minigroups the pairs often adopted two separate roles: a coder and a "googler" who was searching for solutions to problems.

Students seemed to quickly realize that one big strength of a browser based IDE is the ability to use any device in a location independent way. While most of the work was done during the office hours, the students liked that they had an option to continue work when and where ever they felt like.

Deployment. Deployment was generally considered the most severe problem in CoRED. Most of the interviewees mentioned that a lot of time was wasted while waiting other group members to finalize their tasks. In some groups this lead to situations that lots of code was commented away from compilation. As no connectivity to version controlling existed, the comment blocks were also used for storing obsolete code and later it was difficult to know that which parts of the code should be restored from the comment blocks and which can be removed. Some solutions for solving the problem were suggested, such as storing an error-free versions of each file and using them instead. However, the deployment problem also had some positive outcomes. It was reported that because of the problem, the developers mostly concentrated on one problem at a time and tried to finalize their code as quickly as possible.

Project management tools. To make work more effective, some project management tools were suggested to be integrated as a part of the system. For example, it was suggested that a project backlog for future tasks would be a good addition to CoRED. In addition, in the current implementation of CoRED, all the users logged into the system are able to read and write all the projects and files. During the camp, the problem was solved by allocating a private virtual machine to each of the groups and no problems concerning access policies were reported. However, it was suggested that each file and project should have an owner that would be able to share read and write permissions. Some refactoring tools were also asked to be added to CoRED.

Version control. A graphical tool for browsing old versions of a project was considered to be mandatory. One of the interviewees stated that currently most of the old code can not be deleted, but is moved inside a comment block and therefore it is later difficult to know which of the code blocks are still needed. Interviewees stated that certain problems concerning undo and redo functionalities exist in CoRED. A version control system would also ease these problems, as code from older states could be easily restored using version control if undo fails.

As miscellaneous feedback, it was suggested that two weeks would be a more suitable length for a code camp instead of one week. Furthermore, the possibility to include resource files like images and css style sheets to the project was requested. Some of the interviewees exposed that a collaborative IDE can also cause feelings of confusion: it is sometimes impossible to understand the state the project and what are the responsibilities of each group member as the target is constantly changing. In addition, it was stated that collaboration would be easier in a group with no high skill level differences.

9.5 Developer Expectations for Real-time Collaborative IDEs

Based on the surveys, interviews and observing the teams working during the code camp, valuable information concerning real-time collaboration was collected. Above we described how the teams approached real-time collaborative programming, thus answering to the research question $Q1$. In this section, we answer the second question $Q2$ by presenting the features important for a real-time collaborative IDE.

The second research question was:

Q2 What features do developers expect from a real-time collaborative IDE?

Being able to test a web-based real-time collaborative integrated development environment, even with certain deficiencies, gives more perspective for analyzing and reasoning on what features developers expect from such a system. Based on the experience from the code camp we have collected a list (see Table 9.5) of the main features that should be included in a real-time collaborative IDE. In addition to these findings, we speculate that by fixing all the features and organizing a new code camp it is probably possible to find new demands for the system.

The real-time collaboration aspect brings additional twist to deploying and running the application under development. In the basic case, if files contain errors or unfinished code blocks, the compilation fails and the application can not be deployed or tested. During our experiment, it turned out that in collaborative work, most of the time projects contain unfinished features and therefore deploying the application requires some coordination. This issue did not come up in the two-hour proof-of-concept sessions as the task at hand was much smaller, and the team members could thus more easily coordinate their actions, but was a major hindrance in larger projects. An important realization confirmed in our experiment was that even though

Feature	Implemented in CoRED
Edits should be visible in real-time	yes
Possibility to deploy a project containing errors	no
Project management tools	no
Single click deployment	yes
Sufficient awareness between the users	partly
Support for undo/redo	partly
Tools for communicating	partly
Tools for debugging and testing	no
Version control support	no

Table 9.5: Important features for a collaborative web based IDE, in alphabetical order

the developers edit the same code concurrently, the running and debugging of the application should be separate for each developer.

Many developers are used to traditional desktop based IDEs such as Eclipse[12] and Visual Studio[13] and have got used to debug tools provided by these environments. Thus, they have rather high standards for what a web-based environment should offer. Many of the participants considered features such as debugging tools and an ability to set breakpoints to the code to be vital.

Functional undo/redo is also a feature that users expect of a code editor. Collaborative editing makes undo and redo more difficult to understand, as it needs to be designed what is undoable and redoable. Are the users only undoing their own edits, or does the system allow to undo edits made by other users? How should the overlapping edits be undone?

Even in the realm of real-time collaboration, version control is a highly requested feature. Even though version control system is not needed to share the code, it is useful because it allows the developers to revert back to earlier revision as well as start separate development branches. There could be several ways to integrate a version control system into a real-time collaborative environment. It would be simple to just connect the development environment to one of the available version control systems and add a commit button to the user interface. However, how should the old versions be restored? In real-time collaboration, the restoring of old states would affect all the users and therefore it is probably not a good idea to allow. As an alternate option, we are considering a timeline approach that allows users to drag a slider, and have a read-only access to history. That would allow users to copy features

[12] `http://www.eclipse.org`

[13] `http://www.microsoft.com/visualstudio`

from the older versions to the current. However, branches, the possibility to create forks and full restorations would probably not be available in the user interface in such a case.

A different approach to version control could be to purposefully keep the responsibility of version control outside of the web-based collaborative development environment. In this approach, the developers could use any editor they want, but additionally have the possibility to start collaboration sessions on the web. At the start of a collaboration session, the code is retrieved from a version control system to be edited in real-time collaboration. At the end of the session, the code could be committed back to the version control system, and, if needed, another session would be started. Thus, the real-time collaborative environment would just simply be used in between two commits. This approach would have at least two benefits. Firstly, the developers who do not want to use real-time collaboration could use any environment they like and just collaborate via the version control system. This would also allow a development team to be split into smaller collaborating units of, for example, two people. Secondly, there is no need to implement complicated version control features into the web-based environment because most of the responsibility for branching and other version control management would be elsewhere.

Communication is essential in software development. In our experiment the role of the communication tools did not play that big a part because the teams were mostly co-located and could simply talk to each other. During the two hour experiment the sole communication channel between the project leader and their team was the chat box in the environment, which, based on the interviews, succeeded in doing its job. It may not be necessary to try to include all the communication channels to the development environment. Teams could, for example, use an external Voice over IP software for communication. In our experiment, one of the groups reported using a separate chat program for communicating, mostly because they were used to that.

In a collaborative IDE that is open for everybody to use, some project management tools are needed. In a relatively small scale use, such as our experiment, the system can be based on the trust between the users. However, in a system with a large user base, tools for sharing and restricting read and writes to projects and files are necessary.

In real-time collaborative tools, *awareness* is an important concept. There has been lot of research done on, as well as multiple definitions of awareness. Basically awareness means "knowing what is going on" [5]. When people work face-to-face, they exchange a lot of information with their speech, facial expressions, gestures, and other actions as well as by manipulating their shared environment. A large part of that information is lost when using

software tool for collaboration. That is why the tool must deliberatively add features to improve the awareness of other collaborators. Information such as who are currently available for collaboration, what are others looking at, and what are they doing should be transmitted via the collaboration tool.

9.6 Future Work

As already discussed, CoRED is a research artifact and does not fully meet developers' expectations. Via our experiment, we gathered hints on which additional features would bring the most benefit for our system as well as directions for future research on real-time collaborative software development. Before moving on to the ideas for future work, we briefly evaluate the current state of CoRED based on the essential features of a collaborative development environment envisioned above.

The last column of Table 9.5 shows how CoRED currently fulfills the requirements we discussed earlier. When considering the features that are not available in CoRED, no major technical issues are foreseen in their implementation, as long as we can rely on already existing concepts and implementations. We are presently evaluating the relative importance of these from the developer perspective. So far, the ability to deploy a partially erroneous project and the support for version control have been considered as the most important ones.

Deploying and running the collaboratively developed applications turned out to be one of the most apparent hindrances in CoRED. To be able to run the application, the whole project has to be compilable, a condition often not satisfied during real-time collaboration. One way of solving this problem is given in [9], discussed in more detail in Section 9.7. Another deficiency related to running the application was the lack of debugging support. Creating a mechanism that allows the developers to print debug text that would be shown in the IDE would be useful and relatively simple to implement. Setting breakpoints in the code to stop the execution of the application at certain position and inspect the values of variables is a feature seen in many traditional IDEs. Such feature in CoRED would be possible although laborious to implement.

Integrating a version control system into real-time collaborative software development is another subject for future work. As discussed in Section 9.5 and additionally in [19], there could be various ways on how to integrate version control into a system such as CoRED. We plan to further develop models for using version control systems in a real-time collaboration, implement such support in CoRED and try them out in later experiments. As a

related issue, the undo/redo feature does not work optimally in the current version of CoRED in the face of collaborative edits. We need to rethink the undo/redo behavior, and possibly incorporate into some kind of version control or timeline approach. The role of revision control in the light of collaborative software development has also been studied by Magnusson et al. [18]. We assume that the same mechanisms will be reusable in the context of our work, where micro-scale revisions that are created in a collaborative fashion form the technical artifact.

When implementing new features to a real-time collaborative IDE, heavy emphasis needs to be put on the awareness features of the system. Supporting awareness of other project members and the state of the project would be even more important if the developers are physically separated from each other, as opposed to what was the case most of the time in our experiment. In the current version of CoRED, some steps for improving awareness have been taken. CoRED shows who have opened the project, who edits which files as well as their cursor positions. Still, some of the interviewees said that they had trouble keeping track of what is going on. Awareness could be further improved by more clearly indicating where the others are looking at and when they wrote new code. Introducing a version control system with the possibility of branching would pose even more challenges to the awareness aspects of the user interface.

Our goal is to expand the research on real-time collaborative programming in two dimensions. Firstly, we will implement some of the features discussed above and study how they affect the collaborative software development process. Secondly, we seek to broaden the scope of the collaborative programming experiments to a distributed setting by organizing an intercontinental code camp. A development team where the members do not have the benefit of physical proximity to each other would pose greater challenges to the cooperation and collaboration aspects of the tools, as well as possibly expose aspects of real-time collaborative programming not yet captured in the mostly co-located setting described in this chapter.

9.7 Related Work

Computer-based tools that support collaboration, often called *groupware* [4], have been created, used and studied heavily in the multidisciplinary field known as *computer supported cooperative work* [21, 11] for decades. Also in the domain of software development, collaborative processes and tools have been a subject of study. Gutwin et. al. [12] have studied distributive collaborative software development from the perspective of group awareness.

DeFranco-Tommarello et. al. [3] also explore collaborative software development and analyze a list of tools that allow software developers to communicate, coordinate and design software artifacts collaboratively. Even though these kinds of collaboration tools for improving the software development process have been extensively used, the act of writing code in real-time collaboration is a more rarely seen phenomenon. Even though implementations for real-time collaborative code editors have existed much earlier, they have really gained popularity only along with the web becoming a viable application platform.

A number of real-time collaborative software development environments have appeared on the web in recent years. Cloud9[14] is one of the most popular of such environments. It contains a runtime environment for Node.js[15] applications as well as an integration with various version control systems. Another real-time collaborative web-based IDE called DevTable[16] supports creating HTML5 as well as Python applications that can be run in Google AppEngine[17]. Codenvy[18] have also added some real-time collaborative features in their IDE. Many other environments, such as Stypi[19] and Collabedit[20] offer a more limited system without possibility to run the applications in the environment.

There has also been some environments for research purposes. Collabode [9] is a collaborative web-based IDE, implemented as an Eclipse plugin. The most notable feature of Collabode is *error-mediated integration*. The goal of error-mediated integration is to prevent other collaborators' erroneous code to interfere with other developers. It works by keeping a separate copy of a file for each developer, and additionally an error-free copy to which edits are only applied if that can be done to create an error-free file. Thus, even though everybody sees what the other developers write in real-time, everybody has their own version of the project that can be run whether the other developers are in the middle of writing or not. The need for this kind of technique became apparent also during our experiment. Collabode has also been used as a platform for user studies on real-time collaborative programming. Compared to our experiment, their study was smaller in scope (30 or 40 minutes, single file, Java console application) and mostly concentrated on validating their error-mediated integration algorithm.

[14] `https://c9.io/`
[15] `http://nodejs.org/`
[16] `https://devtable.com`
[17] `https://developers.google.com/appengine/`
[18] `https://codenvy.com`
[19] `https://www.stypi.com`
[20] `http://collabedit.com/`

In our experiment, we did not instruct the developers to assume any roles (apart from the two-hour experiment on the third day); they were freely allowed to find a suitable way of working in a collaborative environment. A different approach, envisioned by Goldman [8], is that the real-time collaborative environment could offer for some novel development models to more effectively structure work and to advance close collaboration. They suggest three such models: *test-driven pair programming* where one person writes the code while another developer writes the tests, *micro-outsourcing* where the main developer could "outsource" small implementation tasks to other developers, allowing himself to remain on the same level of abstraction, and *mobile instructor* where one person takes the role of a teacher. As far as we know, a proper support for these kind of roles is not yet implemented in any IDE; they could offer additional interesting research directions.

Another development model often associated with real-time collaborative environments is *distributed pair programming*. That is, pair programming where the participants are not required to be co-located. Saros [25] is a real-time collaborative tool mostly intended for distributed pair programming, with various features for improving awareness. Unlike CoRED, Saros is not web-based but is run inside a desktop Eclipse client. As reported in the case of Saros, companies and open-source projects were reluctant to try out their tool for distriputed pair programming [24]. That could be taken as an indication that, more generally, it may not easy to get the software developers to adopt new development models offered by real-time collaborative tools.

Even if developers are not ready to use real-time collaboration in ordinary work, it could be utilized in some specific situations where close collaboration is essential. One such situation is merge conflicts occurring in a version control system when combining two incompatible branches of development. To resolve the conflict, all the parties involved in causing the conflict could participate in a real-time collaborative conflict-resolving session. A more detailed explanation of such process as well as a web-based tool enabling it is given in [20].

9.8 Conclusion

The current paradigm shift from desktop applications to the browser based applications makes it possible that software development can shift from desktop based IDEs to the web. Among many other good features, this shift makes it easier to enable collaboration between the developers as the developers are connected to the same system.

Earlier, we have implemented a real-time collaborative code editor named

CoRED. To study the software developers perspective of CoRED and collaborative coding in general, we ran a week-long experiment. During the experiment, six small groups of students used the tool for one week while we both guided and observed the students. We collected feedback via questionnaires and interviews. Based on the lessons learned during the code camp we continue to develop CoRED, but what is more important we analyzed what kind of features the developers assume to see in a collaborative web based IDE. Naturally the users would like to combine all the features well-known from desktop based IDEs with the benefits provided by a web based environment such as a low barrier for starting to use the tool as well as collaboration features.

Towards the end of the chapter, we pinpoint nine essential features that should be considered when creating a real-time collaborative development environment. In addition we speculate that more such features could be found by implementing these and by doing another round of experimentation. Based on the results of this study a collaborative tools such as CoRED can offer developers more powerful environments than the current norm to develop software in teams.

Acknowledgments

The authors wish to express sincere thanks to all the participants of the experiment. At the same time, we wish to apologize for the few remaining (but still annoying) bugs in the system. The code camp was part of the projects *TIVIT Digital Services*[21] and *ITEA2 EASI-CLOUDS*[22], supported by Vaadin inc.

References

[1] V. Basili, R. Selby, and D. Hutchens. "Experimentation in software engineering". In: *Software Engineering, IEEE Transactions on* SE-12.7 (1986), pp. 733 –743. ISSN: 0098-5589. DOI: 10.1109/TSE.1986.6312975.

[2] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[21] http://www.tivit-services.fi

[22] http://www.itea2.org/project/index/view/?project=10078

[3] J. DeFranco-Tommarello and F. Deek. "Collaborative software development: a discussion of problem solving models and groupware technologies". In: *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*. 2002, pp. 568 –577. DOI: 10.1109/HICSS.2002.993937.

[4] C. A. Ellis, S. J. Gibbs, and G. Rein. "Groupware: some issues and experiences". In: *Commun. ACM* 34.1 (Jan. 1991), pp. 39–58. ISSN: 0001-0782. DOI: 10.1145/99977.99987. URL: http://doi.acm.org/10.1145/99977.99987.

[5] M. R. Endsley. "Toward a theory of situation awareness in dynamic systems". In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 37.1 (1995), pp. 32–64.

[6] R. T. Fielding. "Chapter 5: Representational State Transfer (REST)". In: *Architectural Styles and the Design of Network-based Software Architectures, Dissertation* (2000).

[7] N. Fraser. "Differential synchronization". In: *Proceedings of the 9th ACM symposium on Document engineering*. DocEng '09. Munich, Germany: ACM, 2009, pp. 13–20. ISBN: 978-1-60558-575-8. DOI: 10.1145/1600193.1600198. URL: http://doi.acm.org/10.1145/1600193.1600198.

[8] M. Goldman. "Role-based interfaces for collaborative software development". In: *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*. UIST '11 Adjunct. Santa Barbara, California, USA: ACM, 2011, pp. 23–26. ISBN: 978-1-4503-1014-7. DOI: 10.1145/2046396.2046410. URL: http://doi.acm.org/10.1145/2046396.2046410.

[9] M. Goldman, G. Little, and R. C. Miller. "Real-time collaborative coding in a web IDE". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. UIST '11. Santa Barbara, California, USA: ACM, 2011, pp. 155–164. ISBN: 978-1-4503-0716-1. DOI: 10.1145/2047196.2047215. URL: http://doi.acm.org/10.1145/2047196.2047215.

[10] M. Grönroos. *Book of Vaadin*. Vaadin Limited, 2011.

[11] J. Grudin. "Computer-supported cooperative work: history and focus". In: *Computer* 27.5 (1994), pp. 19 –26. ISSN: 0018-9162. DOI: 10.1109/2.291294.

[12] C. Gutwin, R. Penner, and K. Schneider. "Group awareness in distributed software development". In: *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. CSCW '04. Chicago, Illinois, USA: ACM, 2004, pp. 72–81. ISBN: 1-58113-810-5. DOI: 10.1145/1031607.1031621. URL: http://doi.acm.org/10.1145/1031607.1031621.

[13] J. D. Herbsleb. "Global Software Engineering: The Future of Socio-technical Coordination". In: *2007 Future of Software Engineering*. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 188–198. ISBN: 0-7695-2829-5. DOI: 10.1109/FOSE.2007.11. URL: http://dx.doi.org/10.1109/FOSE.2007.11.

[14] M. Jonge, E. Visser, and J. M. Visser. *Collaborative software development*. Tech. rep. Amsterdam, The Netherlands, The Netherlands, 2001.

[15] B. Kitchenham et al. "Preliminary guidelines for empirical research in software engineering". In: *Software Engineering, IEEE Transactions on* 28.8 (2002), pp. 721 –734. ISSN: 0098-5589. DOI: 10.1109/TSE.2002.1027796.

[16] B. Kitchenham and S. Pfleeger. "Personal Opinion Surveys". In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, and D. Sjøberg. Springer London, 2008, pp. 63–92. ISBN: 978-1-84800-043-8. DOI: 10.1007/978-1-84800-044-5_3. URL: http://dx.doi.org/10.1007/978-1-84800-044-5_3.

[17] J. Lautamäki et al. "CoRED: browser-based Collaborative Real-time Editor for Java web applications". In: *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. CSCW '12. Seattle, Washington, USA: ACM, 2012, pp. 1307–1316. ISBN: 978-1-4503-1086-4. DOI: 10.1145/2145204.2145399. URL: http://doi.acm.org/10.1145/2145204.2145399.

[18] B. Magnusson, U. Asklund, and S. Minör. "Fine-grained revision control for collaborative software development". In: *Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*. SIGSOFT '93. Los Angeles, California, USA: ACM, 1993, pp. 33–41. ISBN: 0-89791-625-5. DOI: 10.1145/256428.167061. URL: http://doi.acm.org/10.1145/256428.167061.

[19] T. Mikkonen and A. Nieminen. "Elements for a cloud-based development environment: online collaboration, revision control, and continuous integration". In: *Proceedings of the WICSA/ECSA 2012 Companion Volume*. WICSA/ECSA '12. Helsinki, Finland: ACM, 2012,

pp. 14–20. ISBN: 978-1-4503-1568-5. DOI: 10.1145/2361999.2362003. URL: http://doi.acm.org/10.1145/2361999.2362003.

[20] A. Nieminen. "Real-time collaborative resolving of merge conflicts". In: *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*. 2012, pp. 540 –543.

[21] T. Palmer and N. Fields. "Computer supported cooperative work". In: *Computer* 27.5 (1994), pp. 15 –17. ISSN: 0018-9162. DOI: 10.1109/2.291295.

[22] B. W. Perry. *Google Web Toolkit for Ajax*. O'Reilly Media, 2007.

[23] M. Pikkarainen et al. "The impact of agile practices on communication in software development". English. In: *Empirical Software Engineering* 13 (3 2008), pp. 303–337. ISSN: 1382-3256. DOI: 10.1007/s10664-008-9065-9. URL: http://dx.doi.org/10.1007/s10664-008-9065-9.

[24] L. Prechelt. "Some non-usage data for a distributed editor: the saros outreach". In: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 48–48. ISBN: 978-1-4503-0576-1. DOI: 10.1145/1984642.1984651. URL: http://doi.acm.org/10.1145/1984642.1984651.

[25] S. Salinger et al. "Saros: an eclipse plug-in for distributed party programming". In: *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE '10. Cape Town, South Africa: ACM, 2010, pp. 48–55. ISBN: 978-1-60558-966-4. DOI: 10.1145/1833310.1833319. URL: http://doi.acm.org/10.1145/1833310.1833319.

Turku Centre for Computer Science
TUCS General Publications

1. **Joakim von Wright, Jim Grundy and John Harrison (Eds.)**, Supplementary Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics: TPHOLs'96
2. **Mikko Ruohonen and Juha Pärnistö (Eds.)**, Proceedings of the First European Doctoral Seminar on Strategic Information Management
3. **Christer Carlsson (Ed.)**, Exploring the Limits of Support Systems
4. **Mats Aspnäs, Ralph-Johan Back, Timo Järvi and Tiina Lehto (Eds.)**, Turku Centre for Computer Science, Annual Report 1996
5. **Wolfgang Weck, Jan Bosch and Clemens Szyperski (Eds.)**, Proceedings of the Second International Workshop on Component-Oriented Programming (WCOP '97)
6. Working Material from the School on Natural Computation, SNAC
7. **Mats Aspnäs, Ralph-Johan Back, Timo Järvi and Tiina Lehto (Eds.)**, Turku Centre for Computer Science, Annual Report 1997
8. **Reima Suomi, Paul Jackson, Laura Hollmén and Mats Aspnäs (Eds.)**, Teleworking Environments, Proceedings of the Third International Workshop on Telework
9. **Robert Fullér**, Fuzzy Reasonging and Fuzzy Optimization
10. **Wolfgang Weck, Jan Bosch and Clemens Szyperski (Eds.)**, Proceedings of the Third International Workshop on Component-Oriented Programming (WCOP '98)
11. Abstracts from the 10th Nordic Workshop on Programming Theory (NWPT'98)
12. **Edward M. Roche, Kalle Kangas and Reima Suomi (Eds.)**, Proceedings of the IFIP WG 8.7 Helsinki Working Conference, 1998
13. **Christer Carlsson and Franck Tétard (Eds.)**, Intelligent Systems and Active DSS, Abstracts of the IFORS SPC-9 Conference
14. **Mats Aspnäs, Ralph-Johan Back, Timo Järvi, Martti Kuutti, and Tiina Lehto (Eds.)**, Turku Centre for Computer Science, Annual Report 1998
15. **Tero Harju and Iiro Honkala (Eds.)**, Proceedings of the Secenth Nordic Combinatorial Conference
16. **Christer Carlsson (Ed.)**, The State of the Art of Information System Applications in 2007
17. **Christer Carlsson (Ed.)**, Information Systems Day
18. **Ralph-Johan Back, Timo Järvi, Nina Kivinen, Leena Palmulaakso-Nylund and Thomas Sund (Eds.)**, Turku Centre for Computer Science, Annual Report 1999
20. **Reima Suomi and Jarmo Tähkäpää (Eds.)**, Health and Welath through Knowledge
21. **Johan Lilius and Seppo Virtanen (Eds.)**, TTA Workshop Notes 2002
22. **Mikael Collan**, Investment Planning – An Introduction
23. **Mats Aspnäs, Christel Donner, Monika Eklund, Pia Le Grand, Ulrika Gustafsson, Timo Järvi, Nina Kivinen, Maria Prusila and Thomas Sund (Eds.)**, Turku Centre for Computer Science, Annual Report 2000-2001
24. **Ralph-Johan Back and Victor Bos**, Centre for Reliable Software Technology, Progress Report 2003
25. **Pirkko Walden, Stina Störling-Sarkkila, Hannu Salmela and Eija H. Karsten (Eds.)**, ICT and Services: Combining Views from IS and Service Research
26. **Timo Järvi and Pekka Reijonen (Eds.)**, People and Computers: Twenty-One Ways of Looking at Information Systems
27. **Tero Harju and Juhani Karhumäki (Eds.)**, Proceedings of WORDS'03
28. **Mats Aspnäs, Christel Donner, Monika Eklund, Pia Le Grand, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2002

29. **João M. Fernandes, Johan Lilius, Ricardo J. Machado and Ivan Porres (Eds.)**, Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software
30. **Mats Aspnäs, Christel Donner, Monika Eklund, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2003
31. **Andrei Sabelfeld (Ed.)**, Foundations of Computer Security
32. **Eugen Czeizler and Jarkko Kari (Eds.)**, Proceedings of the Workshop on Discrete Models for Complex Systems
33. **Peter Selinger (Ed.)**, Proceedings of the 2nd International Workshop on Quantum Programming Languages
34. **Kai Koskimies, Johan Lilius, Ivan Porres and Kasper Østerbye (Eds.)**, Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques, NWPER'2004
35. **Kai Koskimies, Ludwik Kuzniarz, Johan Lilius and Ivan Porres (Eds.)**, Proceedings of the 2nd Nordic Workshop on the Unified Modelling Language, NWUML'2004
36. **Franca Cantoni and Hannu Salmela (Eds.)**, Proceedings of the Finnish-Italian Workshop on Information Systems, FIWIS 2004
37. **Ralph-Johan Back and Kaisa Sere**, CREST Progress Report 2002-2003
38. **Mats Aspnäs, Christel Donner, Monika Eklund, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2004
39. **Johan Lilius, Ricardo J. Machado, Dragos Truscan and João M. Fernandes (Eds.)**, Proceedings of MOMPES'05, 2nd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software
40. **Ralph-Johan Back, Kaisa Sere and Luigia Petre**, CREST Progress Report 2004-2005
41. **Tapio Salakoski, Tomi Mäntylä and Mikko Laakso (Eds.)**, Koli Calling 2005 – Proceedings of the Fifth Koli Calling Conference on Computer Science Education
42. **Petri Paju, Nina Kivinen, Timo Järvi and Jouko Ruissalo (Eds.)**, History of Nordic Computing – HiNC2
43. **Tero Harju and Juhani Karhumäki (Eds.)**, Proceedings of the Workshop on Fibonacci Words 2006
44. **Michal Kunc and Alexander Okhotin (Eds.)**, Theory and Applications of Language Equations, Proceedings of the 1st International Workshop, Turku, Finland, 2 July 2007
45. **Mika Hirvensalo, Vesa Halava, Igor Potapov and Jarkko Kari (Eds.)**, Proceedings of the Satellite Workshops of DLT 2007
46. **Anne-Maria Ernvall-Hytönen, Matti Jutila, Juhani Karhumäki and Arto Lepistö (Eds.)**, Proceedings of Conference on Algorithmic Number Theory 2007
47. **Ralph-Johan Back and Ion Petre (Eds.)**, Proceedings of COMPMOD 2008
48. **Elena Troubitsyna (Ed.)**, Proceedings of Doctoral Symposium Held in Conjunction with Formal Methods 2008
49. **Reima Suomi and Sanna Apiainen (Eds.)**, Promoting Health in Urban Living: Proceedings of the Second International Conference on Well-Being in the Information Society (WIS 2008)
50. **Auli Stuominen, Jussi Kantola, Arho Suominen and Sami Hyrynsalmi (Eds.)**, NEXT 2008 – Proceedings of the Fifth International New Exploratory Technologies Conference
51. **Tapio Salakoski, Dietrich Rebholz-Schuhmann and Sampo Pyysalo (Eds.)**, Proceedings of the Third International Symposium on Semantic Mining in Biomedicine (SMBM 2008)
52. **Helena Karsten, Barbro Back, Tapio Salakoski, Sanna Salanterä and Hanna Suominen (Eds.)**, The Proceedings of the First Conference on Text and Data Mining of Clinical Documents (Louhi'08)
53. **Anne-Maria Ernvall-Hytönen and Camilla Hollanti (Eds.)**, Proceedings of the 3rd Nordic EWM Summer School for PhD Students in Mathematics
54. **Terry Rout, Ivan Porres, Risto Nevalainen and Beatrix Barafort (Eds.)**, Software Process Improvement and Capability Determination 9th International Conference, SPICE 2009, Turku, Finland, June 2009 Proceedings
55. **Harri Virolainen, Seppo Sirkemaa and Tero Vartiainen (Eds.)**, Proceedings of 14th International Conference on Telework – ITA 2009

56. **Reima Suomi and Ilkka Ilveskoski (Eds.)**, Navigating the Fragmented Innovation Landscape: Proceedings of the Third International Conference on Well-Being in the Information Society (WIS 2010)
57. **Marina Waldén and Luigia Petre (Eds.)**, Proceedings of the 22nd Nordic Workshop on Programming Theory NWPT'10
58. **Irmeli Laine, Johan Lilius, Tomi Mäntylä, Ion Petre, Outi Tuohi and Ilona Tuominen (Eds.)**, Turku Centre for Computer Science, Annual Report 2012
59. **Heikki Partanen**, Matematiikan johtaminen luonnollisten lukujen teoriasta
60. **Ivan Porres, Tommi Mikkonen and Adnan Ashraf (Eds.)**, Developing Cloud Software: Algorithms, Applications, and Tools

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2952-7
ISSN 1239-1905

Developing Cloud Software

Developing Cloud Software

Developing Cloud Software: Algorithms, Applications, and Tools