Bachelor

March 20, 2015

Cloud WorkBench

A Web-Based Framework for Benchmarking Cloud Services

Joel Scheuner

of Muensterlingen, Switzerland (10-741-494)

supervised by

Prof. Dr. Harald C. Gall

Dr. Philipp Leitner, Jürgen Cito

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Bachelor

Cloud WorkBench

A Web-Based Framework for Benchmarking Cloud Services

Joel Scheuner

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Bachelor

Author: Joel Scheuner, joel.scheuner@uzh.ch

Project period: 04.03.2014 - 14.08.2014

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

This bachelor thesis constitutes the last milestone on the way to my first academic graduation. I would like to thank all the people who accompanied and supported me in the last four years of study and work in industry paving the way to this thesis. My personal thanks go to my parents supporting me in many ways. The decision to choose a complex topic in an area where I had no personal experience in advance, neither theoretically nor technologically, made the past four and a half months challenging, demanding, work-intensive, but also very educational which is what remains in good memory afterwards.

Regarding this thesis, I would like to offer special thanks to Philipp Leitner who assisted me during the whole process and gave valuable advices. Moreover, I want to thank Jürgen Cito for his mainly technologically-related recommendations, Rita Erne for her time spent with me on language review sessions, and Prof. Harald Gall for giving me the opportunity to write this thesis at the Software Evolution & Architecture Lab at the University of Zurich and providing fundings and infrastructure to realize this thesis.

Abstract

Cloud computing has started to play a major role in IT industry and the numerous cloud providers and services they offer are continuously growing. This increasing cloud service diversity, especially observed for infrastructure services, demands systematic benchmarking in order to assess cloud service performance and thus assist cloud users in service selection. However, manually conducting cloud benchmarks is time-consuming and error-prone. Therefore, previous work addressed these problems with automation approaches but has failed to provide a convenient way to automate the process of installing and configuring a benchmark.

In order to solve this benchmark provisioning problem, this thesis introduces a benchmark automation framework called Cloud WorkBench (CWB) where benchmarks are entirely defined by means of code and executed without manual interaction. CWB allows to define configurable benchmarks in a modular manner that are portable across cloud providers and their regions. New benchmarks can be added at runtime and variations thereof are conveniently configurable via a web interface. Integrated periodic scheduling capabilities timely trigger benchmark executions that automatically acquire cloud resources, prepare and run the benchmark, and release previously acquired resources. CWB is used to conduct a case study in the Amazon EC2 cloud to examine a very specific performance characteristic for a combination of different service types. The results reveal limitations of the cheapest service type, show that performance does not necessarily correlate with service pricing, and illustrate that a new service type at the time of conducting this case study is able to reduce variability and enhance performance. CWB has already executed nearly 20000 benchmarks in total and is used in ongoing research.

Zusammenfassung

Cloud Computing spielt zunehmend eine wesentliche Rolle in der IT-Branche und die zahlreichen Cloud Anbieter und deren Dienste wachsen kontinuierlich. Diese steigende Vielfalt an Cloud Diensten, insbesonderes von Infrastruktur-Diensten, erfordert systematisches Benchmarking, um die Leistung der Dienste einzuschätzen und dadurch Cloud Benutzer in der Diensteauswahl zu unterstützen. Benchmarks manuell durchzuführen ist jedoch zeitaufwändig und fehleranfällig. Daher haben bisherige Studien Lösungsansätze mittels Automatisierung vorgeschlagen. Keiner dieser Ansätze bietet jedoch eine komfortable Lösung um den Installations- und Konfigurationsprozess von Benchmarks zu automatisieren.

Um dieses Benchmark-Bereitstellungsproblem zu lösen, führt diese Arbeit ein Automationsframework für Benchmarks, genannt Cloud WorkBench (CWB), ein, das Benchmarks vollständig mittels Code definiert und ohne manuelle Interaktion ausführt. CWB ermöglicht konfigurierbare Benchmarks in modularer Weise zu definieren, die über die Grenzen der Anbieter und deren Regionen hinaus ausführbar sind. Neue Benchmarks können zur Laufzeit hinzugefügt werden und Variationen davon sind komfortabel über eine Web-Oberfläche konfigurierbar. Integrierte periodische Zeitpläne lösen zeitgerecht Benchmark-Ausführungen aus, die automatisch Cloud Ressourcen erwerben, den Benchmark vorbereiten und durchführen, und zuvor erworbene Ressourcen wieder freigeben. CWB wird verwendet, um eine Fallstudie in der Amazon EC2 Cloud durchzuführen und dabei ein sehr spezifisches Leistungsmerkmal für eine Kombination unterschiedlicher Diensttypen zu analysieren. Die Resultate zeigen Einschränkungen des günstigsten Diensttyps auf, zeigen, dass die Leistung nicht zwangsläufig mit dem Dienstpreis korreliert, und illustrieren, dass ein zum Durchführungszeitpunkt der Fallstudie neuer Diensttyp die Variabilität reduzieren und die Leistung verbessern kann. CWB hat insgesamt bereits nahezu 20000 Benchmarks ausgeführt und wird in laufender Forschung eingesetzt.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing [AFG$^+$09, BBG11] introduced a new paradigm that has the potential to fundamentally impact the IT industry. In cloud computing, resources, such as Virtual Machines (VMs), programming environments, or entire application services, are acquired on a pay-per-use basis. Recent literature mentions the revolutionary effect [GVB13, GSR13] of this disruptive [CCVK13, SBC$^+$13, MC13] but still rapidly developing paradigm on the IT industry and the fact that cloud computing is already part of the strategy of major companies in IT industry such as Microsoft [Nad14] indicates its overall importance.

The large number of emerging Infrastructure-as-a-Service (IaaS) providers and new services they offer make selecting an appropriate cloud service a challenge. In the IaaS model *"processing, storage, networks, and other fundamental computing resources"* [MG11] are acquired on a pay-per-use basis and most commonly in the form of VMs. The functional similarities of these services are contrasted by significant variations in non-functional properties. Service performance not only varies between providers, as studies listed in [FJV$^+$12] show, but also for services exhibiting the same specification [GLOT13]. Under these conditions, software engineers obtain the best results for service selection in terms of accuracy and relevance by running the actual (*i.e.*, real world) application in the cloud. However, systematic benchmarking (*i.e.*, performance testing) of actual applications is practically rarely possible without spending extraordinary time and financial expenses. Therefore, representative benchmarks are chosen to estimate the performance of the actual application.

Systematic cloud benchmarking is an elaborate task and demands automation in order to efficiently conduct various benchmarks. Although representative benchmarks are typically much easier to deploy and execute on cloud services than actual applications, testing multiple providers with variable configurations results in a large parameter space to explore, making this kind of benchmarking still labor intensive. Moreover, in fast moving cloud environments, continuous reevaluation is inevitable. Therefore, several research projects [SHG$^+$13, JKC$^+$13, CMS13] aiming at extensible cloud experiment automation were recently introduced to alleviate this problem. They all facilitate systematic cloud benchmarking but none of them provides a satisfactory solution for the benchmark installation and configuration problem. Time-consuming benchmark preparation was identified as recurring problem especially for application benchmarks in [CUWS11] and application deployment in general was mentioned as a key challenge for cloud computing in [MVML13]. Since many benchmarks are not specifically designed for the cloud, they suffer from the same problems [ZL11] as actual applications do. The common approach to address these problems involves manually creating VM images for each benchmark, cloud provider and region. However, this approach does not suffice to conduct systematic benchmarking at large scale.

1.1 Goals and Contributions

The previous section motivates the following research question:

How can a web-based framework support experimenters in defining, scheduling, and executing IaaS cloud benchmarks?

This wide-ranging research question is refined into two sub-questions:

Research Question I:
How can common IaaS cloud benchmarks from literature be defined in a modular and portable manner?

Research Question II:
How can benchmarks from research question I be periodically scheduled and reproducibly executed in cloud environments without manual interaction?

In order to answer these questions this thesis introduces a framework called Cloud WorkBench (CWB) that is designed and implemented to fulfill the requirements implied by the research questions. Furthermore, a case study with a sample benchmark demonstrates the capabilities of CWB.

This thesis makes the following three contributions:

1. It provides a taxonomy of cloud services and IaaS cloud benchmarks with corresponding examples from literature.
2. It introduces CWB, a web-based framework to define, schedule, and execute cloud benchmarks.
3. It presents the results of a case study with a micro-benchmark conducted by using CWB.

1.2 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 provides theoretical and technological background information. Chapter 3 introduces the developed CWB framework. Chapter 4 discusses the results of a sample benchmark that was defined and executed with CWB. Chapter 5 relates this contribution to existing work in the area and finally Chapter 6 concludes this thesis and outlines future work.

Chapter 2

Background

This chapter is a compilation of literature in the areas of cloud computing and cloud benchmarking and gives some technological background for the next chapter describing the CWB framework.

2.1 Definition of Cloud Computing

There has been no commonly accepted definition for cloud computing in literature for several years as mentioned in [VRMCL08, Hil09, AI10, BBG11]. Although still not completely accepted, with the publication of the final National Institute of Standards and Technology (NIST) definition from the U.S. Department of Commerce in 2011, this has become the most cited definition:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.*, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [MG11]."

This definition covers five essential characteristics:

- *On-demand self-service.* Computing resources can be acquired and released at any time in a fully automated manner [MG11, ZL11]. This ability eliminates the up-front commitment to a fixed hardware capacity allowing consumers to scale up and down according to their needs [AFG$^+$09].
- *Rapid elasticity.* Rapid elasticity allows to adapt the capacity of the acquired computing resources within a short time. Customers have the illusion of unlimited computing resources being available almost instantly [AFG$^+$09].
- *Utility-based pricing.* Computing resources are metered and charged based on the actual usage. This pay-per-use or alternatively pay-as-you-go business model enables short-term usage without contractual bindings or up-front investments [BKKL09, ZCB10, AFG$^+$09].
- *Resource pooling.* Cloud service providers massively exploit economies of scale with huge datacenters serving many different customers concurrently [AFG$^+$09,ZCB10]. The complexity of such multi-tenant environments is abstracted and computing resources are offered to customers as services with well defined interfaces [HK11].
- *Broad network access.* The computing resources available over the network are generally accessible over the internet by any type of device [ZCB10]. Cloud computing datacenters

are geographically distributed across the world in order to achieve high availability and quality of service.

The definition provided above describes cloud computing as a model which should point out that cloud computing is no new technology as it is often misunderstood [SASA$^+$11]. Instead, cloud computing builds upon existing technologies such as virtualization [ZCB10] and exploits existing trends such as the ubiquitous network connectivity [Hil09]. Thus, cloud computing is closely related to existing paradigms such as grid or cluster computing as described in [ZCB10, BYV$^+$09].

Another important implication of cloud computing is the increasing focus on horizontal scalability. When horizontally scaling-up, also called scaling-out, additional computing resources are used which contrasts vertical scalability where the computing power of existing resources is increased. The cloud computing characteristics foster the trend from vertical to horizontal scalability. For example, by rapidly acquiring and releasing resources on demand, cloud users can exploit cost associativity since running one instance for 1000 hours costs the same as running 1000 instances for one hour [AFG$^+$09]. Cloud providers achieve resource pooling to this extent by utilizing commodity hardware. Therefore, continuous failures should be treated as usual in cloud environments and handled appropriately.

2.2 Taxonomy of Cloud Services

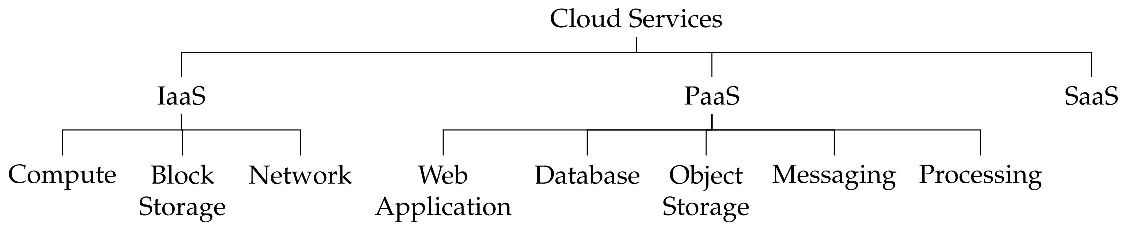
This section classifies cloud services based on cloud computing literature and concrete cloud services review. Basically, cloud services are commonly categorized into the following three layers as shown by Figure 2.1:

1. IaaS constitutes the fundamental layer that offers virtualized low-level compute, storage, and network resources. Its users have usually full control over nearly the entire software stack and fine grained configuration options [KSHD13, AFG$^+$09].

2. Platform-as-a-Service (PaaS) is the middle layer that offers a managed environment for building and deploying applications in the cloud [HK10]. Its users control their applications but have no direct access to the underlying hardware and software infrastructure [MG11].

3. Software-as-a-Service (SaaS) as topmost layer offers fully functional applications or services. Its users can neither control individual applications nor access the underlying infrastructure [MG11].

Figure 2.1: Taxonomy of Cloud Services



Although the terms IaaS, PaaS, and SaaS have become relatively common, the borders between these layers are still quite blurred. Various attempts to extend this layered model [YBS08, RCL09, dOBM10, KSHD13] seem to increase the confusion around all the different XaaS terms even more. Therefore, for clarity, this thesis will stick to the common IaaS, PaaS, and SaaS terminology.

2.2.1 IaaS Services

The IaaS model exposes the underlying node-based hardware infrastructure [KSHD13] and can be categorized into compute, block storage, and network services.

Compute. Compute services offer CPU and GPU processing time in the form of VMs. Amazon Elastic Compute Cloud (EC2)[1] has made this kind of service popular and subsequently many new services have started to emerge. Currently, beside the market leader Amazon EC2, major services include Microsoft Azure[2] and Google Compute Engine[3] [LTG+14]. Rackspace[4] is especially known for its open source infrastructure based on OpenStack[5] and numerous smaller providers such as CloudSigma[6] are present at the market.

Block Storage. Block storage services offer virtual disks that can be attached to VMs. Although they often come with a preinstalled file system on top, block-level access is granted to cloud users for this kind of storage. Sometimes cloud users can choose between cheaper Hard Disk Drive (HDD) storage and faster Solid State Disk (SSD) storage. Examples of block storage services are Amazon Elastic Block Storage (EBS)[7] and Rackspace Cloud Block Storage[8].

Network. Networking services offer fine grained configuration options covering Virtual Private Network (VPN), IP addresses, subnets, routing tables, and network gateways. An example for this kind of service is Amazon VPC[9].

2.2.2 PaaS Services

The PaaS model abstracts the underlying node-based infrastructure [KSHD13] and provides tools and extensive APIs to support developers in building software. Its services sometimes offer additional features such as automatic scaling, load-balancing and replication [AI10] but always require development effort to create entire applications. These services can be further categorized into platforms for web applications, databases, object storage, messaging, and processing.

Web Application. Platforms for web applications allow developers to focus on the business logic. They are no longer responsible for infrastructure installation and maintenance but may be limited by certain restrictions imposed by the service providers. Commonly, the business logic must be stateless [AFG+09] and file system access is restricted to prevent scalability issues. Examples of web application platforms include Google App Engine[10], AWS Elastic Beanstalk[11], Engine Yard[12], OpenShift[13], Heroku[14], Azure Web Apps[15], and Salesforce1 Platform[16] which was formerly called force.com.

Database. Database platforms offer relational and non-relational NoSQL database services. Developers can focus on the data model and database interactions while the database management is abstracted and only restricted configuration options are exposed.

Offering this convenience for traditional relational database systems is challenging due to the limited horizontal scalability of ACID conform databases in cloud environments. Given that clouds must gracefully handle network failures, according to the CAP theorem [GL02], cloud providers must sacrifice either consistency or availability [BKKL09]. Although recent findings in research alleviate this problem with advanced techniques that deal with network partitioning [Bre12], users should be aware of the trade-off a specific service has made. Examples of relational database services include Azure SQL[17], Amazon RDS[18], and Google Cloud SQL[19]

NoSQL database services exhibit good scale-out capabilities and are sometimes specifically designed for clouds [Mah13, FAK$^+$12]. Examples of such services are Amazon DynamoDB[20], Amazon SimpleDB[21], Azure Table Store[22], Google Cloud Datastore[23], and Google BigQuery[24].

Object storage. Object storage platforms are designed to store large chunks of data. The data is typically organized in a flat namespace within some kind of buckets. In contrast to low-level IaaS block storage, these services are accessed via a web API and handle complex features such as replication transparently. Examples of object storage platforms include Amazon S3[25], Google Cloud Storage[26], Azure Blob Store[27], and Rackspace Cloud Files[28].

Messaging. Messaging services facilitate collaboration among multiple cloud services. For example, Amazon SQS[29] can be used as reliable message bus with high throughput to decouple application components such as web front-ends from long-running background processing logic.

Processing. Processing platforms offer computation capabilities via an abstract computation model that allows to describe processing jobs. In contrast to IaaS, abstract processing jobs do not expose their underlying node-based infrastructure. In Amazon's Elastic MapReduce[30] for example, MapReduce jobs can be submitted and are transparently processed in a distributed manner. Another example with a more generic processing model is PiCloud[31].

2.2.3 SaaS Services

The SaaS model offers entire applications running on cloud infrastructures as a service accessible via a web interface [MG11]. Such services are often well-known from everyday life. Table 2.1 provides some examples of SaaS services categorized by application domain. This list could be arbitrarily extended as there is a vast number of cloud services available via internet.

Table 2.1: SaaS Services

Application Domain	Examples
Collaboration	Dropbox[32], Box[33], Evernote[34], Basecamp[35]
Productivity	Google Apps[36], Microsoft Office 365[37]
CRM	Salesforce1 Sales Cloud[38], Microsoft Dynamics CRM[39]
Social Networks	Facebook[40], Google+[41]
Multimedia	Youtube[42], Flickr[43]
Graphics and Design	Adobe Creative Cloud[44]

2.3 Taxonomy of IaaS Cloud Benchmarks

This section shortly introduces cloud benchmarking and subsequently classifies IaaS cloud benchmarks based on cloud benchmarking literature review.

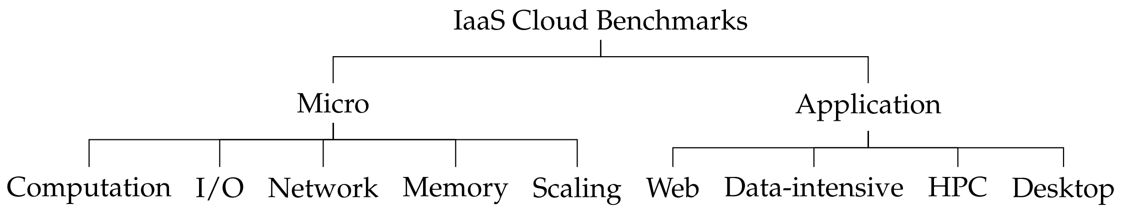


Figure 2.2: Taxonomy of IaaS Cloud Benchmarks

The goal of a cloud benchmark is to assess the performance of a service offered by a cloud provider. Conducting this task is called benchmarking, the service being benchmarked is the System Under Test (SUT) and the benchmark components managing the execution and initiating the workload are called drivers [FAS+13].

Figure 2.2 shows that cloud benchmarks can be categorized into micro- and application benchmarks. Micro-benchmarks *"measure performance of primitive operations supported by"* [Zha01] the cloud service. They typically generate a high artificial workload for a specific computing resource [Gre13]. In contrast, application benchmarks measure entire application stacks whose overall performance is influenced by many aspects. The used workloads typically aim at simulating real-world scenarios. Finally, this section concludes by discussing state of the art cloud benchmarks from contemporary literature.

2.3.1 Micro-Benchmarks

Micro-benchmarks can be particularly useful to examine causes of performance bottlenecks but are unable to estimate actual application performance on its own. They often consist of small pieces of code and typically report the mean of repetitive executions. In the following, micro-benchmarks are further categorized by the cloud operation types they specifically measure, and examples with references to their usage in cloud benchmarking literature are provided.

Computation. General purpose (CPU) and graphical processing (GPU) computation capabilities of cloud VMs are determined by the instance type offered from the cloud provider. Varying VM specifications among cloud providers make the comparison of their computation capabilities difficult. Processor sharing strategies [WN10], applied to increase tenant concurrency for certain instance types, even exacerbate this comparison. Computation micro-benchmarks can reveal fundamental CPU and GPU performance. They typically report the number of low-level operations per time such as Giga Floating Point Operations per Second (GFLOPS) or Giga Operations per Second (GOPS) for integer operations. Some benchmarks have their own scoring metrics and others measure the calculation time for predefined workloads as the examples in Table 2.2 show.

I/O. Cloud providers offer a variety of different block-based storage types. Instance storage, directly available on the VM, can be extended with dedicated block storage that is cloud-internally connected over the network. I/O micro-benchmarks can help selecting the best storage type for a given application by conducting performance analysis of read and write operations. Thereby, two types of I/O benchmarks are differentiated. Disk I/O benchmarks measure the raw performance of the physical disk whereas file system I/O benchmarks are targeted at the overall file system which includes caching, buffering and asynchronous I/O optimization mechanisms. I/O benchmarks are usually configurable with parameters such as operation type (sequential or random) or

Table 2.2: Computation Micro-Benchmarks

Benchmark	Metrics	Sample Usage
LINPACK	GFLOPS for linear equations	[GLOT13]
HPCC/HPL[45] (High-Performance Linpack)	GFLOPS for linear equations	[AM10, NB09, OIY$^+$10, IOY$^+$11, JRM$^+$10]
LAPACK[46] by ATLAS[47]	GFLOPS for vector and matrix operations	[AM10]
HPCC/DGEMM [LBD$^+$06]	GFLOPS of double precision real matrix-matrix multiplication	[OIY$^+$10, IOY$^+$11, JRM$^+$10]
LMBench[48] (Suite of micro-benchmarks for various operation types)	Many *e.g.*, GOPS for integer operations, GFLOPS for floating point operations	[OIY$^+$10, IOY$^+$11, Gre13]
UnixBench[49]	Benchmark score	[OZN$^+$12]
Coremark[50]	Benchmark score	[SHG$^+$13]
Simplex	Completion time in ms	[SASA$^+$11]
SHOC [DMM$^+$10] (GPU)	GFLOPS, GB/s, completion time in seconds	[ETR$^+$13]

block size. Typically, bandwidth and latency are reported as examples from literature in Table 2.3 show.

Table 2.3: I/O Micro-Benchmarks

Benchmark	Metrics	Sample Usage
Bonnie/++[51]	Completion time in seconds, bandwidth in KB/s for sequential I/O	[GLOT13, FJV$^+$12, OIY$^+$10, SDQR10]
FIO[52] Flexible I/O Tester	Bandwidth in KB/s for sequential I/O	[SASA$^+$11]
Filebench[53]	Bandwidth in KB/s, latency in milliseconds	[IHJ11, WJC$^+$10]
Postmark [Kat97]	Completion time in seconds of writing files	[IHJ11, WJC$^+$10]
Dbench[54]	Bandwidth in KB/s	[OZN$^+$12]

Network. Networking performance depends on many parameters that are abstracted by the cloud provider. Thus, cloud users are limited in optimizing the network performance since properties such as the underlying physical structure or the overall workload in the datacenter are unknown. Networking micro-benchmarks can reveal bottlenecks in the large number of connection possibilities in the cloud. Such benchmarks may cover incoming and outgoing connections for different providers, regions, and availability zones within a region. Typical metrics are bandwidth, latency, and reliability as examples from literature in Table 2.4 show.

Table 2.4: Network Micro-Benchmarks

Benchmark	Metrics	Sample Usage
iperf[55]	Bandwidth in MB/s, latency via Round Trip Time (RTT) in milliseconds	[GLOT13, FJV+12, LYKZ10a, HZK+10, BK09]
netperf[56]	Bandwidth in KB/s	[SHG+13]
mpptest[57]	Message passing bandwidth in bits per second, latency in microseconds	[Wal08, GLOT13]
ping	Latency via RTT in milliseconds	[WN10, LYKZ10a]
Badabing [SBDR05]	Reliability via loss frequency and loss rate, latency in milliseconds	[WN10]

Memory. Memory micro-benchmarks in the cloud are basically the same as in non-cloud environments with the exception of clustered scenarios. Typical metrics include bandwidth, latency and number of operations (read/write/update) per time. Table 2.5 shows examples used in literature.

Table 2.5: Memory Micro-Benchmarks

Benchmark	Metrics	Sample Usage
HPCC/STREAM[58]	Bandwidth in MB/s	[SASA+11, GLOT13, OIY+10, IOY+11, TMV+11, JRM+10]
HPCC/RandomAccess[59]	Giga Updates per Second (GUPS)	[OIY+10, IOY+11, JRM+10]
CacheBench[60] [MLT98] (Cache)	Bandwidth in MB/s	[IOY+11, OIY+10]
Redis[61]	Requests per second for SET and GET operations	[OZN+12]

Scaling. Requesting cloud resources takes a certain amount of time before they are ready to use. Scaling micro-benchmarks measure this time differing between providers and services [MH12]. These benchmarks are especially useful in optimizing scale-out applications that acquire resources on demand to cover peak load periods. Studies examining the VM startup time in seconds are [MH12, LYKZ10a, GVB13, SHG+13].

2.3.2 Application Benchmarks

Application benchmarks are used to estimate the performance of actual applications. They reveal performance issues but are unable to identify the root causes. Typical metrics include completion time and throughput for predefined workload scenarios. They can be further distinguished by the type of application they aim to represent.

Web. Currently, web application benchmarks are the most important type of cloud benchmarks. Binnig *et al.* [BKKL09] proposed *"that a new cloud benchmark should be based on a e-commerce scenario"* and the IT industry is especially interested in cost efficiency studies with web application benchmarks. Such benchmarks deploy entire sample web applications and define representative web interaction patterns for different workload scenarios. Reported metrics focus on throughput, system resource usage, and cost efficiency, as the examples from literature in Table 2.6 show.

Table 2.6: Web Application Benchmarks

Benchmark	**Metrics**	**Sample Usage**
CloudStone [SSS+08]	Dollars per user per month	[SSS+08]
RUBiS[62]	CPU usage over time, Service Level Agreement (SLA) violation rate	[GGW10, SSGW11, JSM+12]
RUBBoS[63]	Throughput in requests per second, CPU utilization, response time in seconds	[JMQ+11]
TPC-W[64]	Throughput via requests per second, costs per Web Interactions Processed per Second (WIPS)	[KKL10, CCVK13]
DayTrader[65]	Throughput via transactions per second, CPU usage, disk usage	[UN10, SHG+13]
SPECweb2005[66]	CPU usage, network bandwidth in MB/s	[LW09]
httperf[67] [MJ98]	Throughput via number of responses depending on number of requests	[OZN+12]
Wikipedia workload	Throughput via requests per second	[UN10]

Data-intensive. Data-intensive application benchmarks are getting more important with the hype of "Big Data" [LF13] and the massive amount of data being collected. They help to assess performance and thus estimate completion time for large processing tasks operating on enormous datasets. Processing such amounts of data demands for massive parallelism which is the reason why data-intensive benchmarks usually exhibit cluster topologies. These kind of benchmarks typically underly sequential disk I/O performance bottlenecks [SBV+09] and involve the challenging simulation of enormous datasets. Table 2.7 shows examples of data-intensive application benchmarks.

Table 2.7: Data-intensive Application Benchmarks

Benchmark	**Metrics**	**Sample Usage**
Hadoop HiBench	Throughput via number of tasks completed per minute, job latency in seconds, CPU usage, memory usage, disk usage	[HHD+10, SHG+13]
GridMix	Execution time of MapReduce jobs	[WJC+10]

HPC. High Performance Computing (HPC) application benchmarks are mainly applied in research. They cover performance analysis of complex computation-intensive real-world applications such as physical particle simulations or climate prediction models. Workloads, well-suited for supercomputers, are horizontally distributed across many cloud instances. This typically degrades HPC application performance due to networking bottlenecks [HZK+10]. Table 2.8 shows examples of HPC application benchmarks.

Table 2.8: HPC Application Benchmarks

Benchmark	Metrics	Sample Usage
NAS parallel[68] [BBB+91]	Completion time	[Wal08, AM10, RBD+12]
PARSEC [BKSL08]	Completion time, throughput in tasks per second	[WJC+10, IHJ11]
SPLASH-2 [WOT+95]	No results presented	[LZK+11]
NAMD ApoA1[69] (GPU)	Simulation rate in days per nanoseconds, speedup with increasing number of instances	[ETR+13]
MC-GPU[70] (GPU)	Completion time in seconds, speedup with increasing number of instances	[ETR+13]

Desktop. Desktop application benchmarks are executed on a single VM and may be used to estimate the performance of a VM in general or of a specific programming language environment running on a VM. Table 2.9 shows examples of Desktop application benchmarks.

Table 2.9: Desktop Application Benchmarks

Benchmark	Metrics	Sample Usage
DaCapo[71] [BGH+06]	Completion time	[RMP10]
Linux kernel compilation	Completion time, CPU usage, memory usage	[BK09, ERR10]
SPECjvm2008[72]	Operations per second	[ŠS11]

2.3.3 State of the Art Cloud Benchmarks

Traditional benchmarks such as TCP-W do not suffice to analyze the performance of cloud services. Binnig *et al.* [BKKL09] motivated the need for new benchmarks that are specifically designed for cloud environments. Such a new benchmark should take the characteristics of cloud computing into account. For example, environment constraints limit the control over the SUT, weaken database consistency guarantees, and require fault tolerant benchmark design. Furthermore, cloud features, such as scalability and pay-per-use, make some existing metrics and workloads obsolete and demand for new cost-based metrics and scale-out workloads.

According to Jia *et al.* [JWZ+13], the state of the art benchmark designed for scale-out workloads is currently CloudSuite[73] [FAK+12]. CloudSuite is a collection of benchmarks covering different classes of applications including the web serving CloudStone [SSS+08] benchmark. The CloudStone benchmark aims at representing modern Web 2.0 applications whose interactive workloads differ from static Web 1.0 page serving. Even though CloudStone was designed with Web 2.0 interaction patterns in mind, a study in 2011 [CUWS11] has shown that popular web sites contain significantly more multimedia and interactive content than CloudStone. Therefore, application benchmarks and their workloads must be continuously adapted in order to stay representative, which is especially true for dynamic cloud environments.

2.4 Tools for Cloud Deployment

Tools can help to efficiently manage cloud environments. Deploying software (*e.g.*, benchmarks) in IaaS clouds on demand involves two areas suited to be supported by existing tools. Firstly, the installation and configuration of software (*i.e.*, provisioning) within acquired cloud VMs can be automated and secondly, the VM environment itself (*i.e.*, the lifecyle of cloud VMs) can be managed. In the following, tools that cover provisioning (Chef) and VM environment management (Vagrant) are presented.

2.4.1 Chef

Chef[74] is an industry-leading [SBC+13], open source[75] configuration management tool with a proprietary enterprise extension. It allows to describe the desired state of a node (*i.e.*, physical or virtual machine) using a declarative syntax and automatically converges nodes to this desired state in an idempotent manner. Chef is used by prominent companies such as Facebook to manage cloud infrastructure at large scale [Che13, Dib13].

Chef supports two different modes. In the Chef-client mode a single Chef server manages the configuration of client nodes. Each node fetches its configuration from the server and applies it during the Chef-client run. In the Chef-solo mode the configuration is pushed to a node before executing the Chef-solo run. Chef-solo requires no dedicated Chef server but only supports a subset of the features of the client-server mode.

Chef configurations are modularized into cookbooks. A cookbook is a module of infrastructure code similar to what a gem is in Ruby or a Maven project is in Java. It is versioned and may have dependencies, platform constraints and other metadata. In its core a cookbook consists of at least one recipe that specifies which resources are applied to a node in which order. Recipes are written in the internal Domain Specific Language (DSL) [Cun08] for describing infrastructure state. This DSL is implemented in Ruby and therefore offers the full flexibility of the Ruby programming language. A resource, expressed in this DSL, describes the desired state of a specific configuration item on a node. An example for a resource is a file that should be created on the node based on a predefined template. In order to enhance reusability of cookbooks, Chef provides a sophisticated attribute mechanism.

2.4.2 Vagrant

Vagrant[76] is an open source[77] tool written in Ruby to manage VM environments. Its initial version aimed at supporting the creation of local VirtualBox[78] development environments in a reproducible manner and with minimal manual effort. With version 1.1[79], a plugin architecture has been introduced to support multiple providers. As of July 2014, Vagrant plugins are available for more than 10 cloud providers[80] including the popular public clouds Amazon EC2[81], Google Compute Engine[82], Microsoft Azure (formerly Windows Azure)[83], and Rackspace[84].

Vagrant provides an internal DSL implemented in Ruby for describing VM environments in a configuration file called Vagrantfile. Various configuration options are available depending on the provider. Local providers such as VirtualBox allow detailed VM instance and network specifications whereas cloud providers such as Amazon EC2 have limited options in choosing from predefined instance types and geographic regions. With a single command, a set of VM instances can be automatically allocated and configured according to previously specified options.

Industry-standard provisioners such as Chef or Puppet[85] are supported in order to install and configure software on VMs. The Chef-client provisioner for example connects to a Chef server, fetches the node specific cookbooks and applies them to the node.

The command-line interface offers various commands useful for managing virtual environments. Depending on the actions available for specific providers, VMs can be suspended, halted, and destroyed. Furthermore, remote commands can be executed via Secure Shell (SSH). Thereby, authentication is abstracted and transparently handled by Vagrant via the provided configuration.

Chapter 3

Cloud WorkBench

This chapter describes the CWB framework for defining, scheduling, and executing benchmarks in its structure, interactions, implementation, and deployment. After giving an overview over the overall system architecture, it details how a benchmark is defined, what happens in which order when a benchmark is executed, what states a benchmark execution can take, and what type of benchmark results are reported. Finally, some implementation details reveal how the employed technology contributes to CWB.

3.1 Overall System Architecture

Defining and executing a benchmark with CWB involves interactions among five components as illustrated by Figure 3.1. The *workstation* is the web client device of the experimenter used to define benchmarks via the provisioning service and the CWB web interface which subsequently allows to schedule and manage executions of benchmarks. The *CWB server* is the main component consisting of a three-tiered web application. It provides the web interface, implements the business logic in collaboration with external dependencies, and stores its data in a relational database. The *provider API* of a IaaS cloud provider is used to acquire and release cloud resources that are most importantly cloud VMs (*i.e.*, VM instances in the cloud). The *cloud VM* is usually the SUT, may contain the benchmark driver, and is supported by a custom CWB client utility library. The *provisioning service* manages VM configurations and provides a query API that enables configuring benchmarks exhibiting multi-VM topologies.

The CWB business logic builds upon external dependencies for providing benchmark scheduling capabilities and managing the lifecyle of cloud VMs. The latter responsibility is fulfilled by the extensible VM environment manager. In its core, the VM environment manger offers generic, provider-independent VM lifecycle functionality such as provisioning or remote command execution. Cloud provider plugins extend the core by implementing concrete IaaS cloud provider APIs to acquire and release cloud VMs and optionally manage additional provider-specific resources.

The CWB server and the cloud VMs interact with each other in order to control the execution of a benchmark. On one side, the CWB server prepares the benchmark by orchestrating VM provisioning which in turn includes interaction with the provisioning service. Additionally, the CWB server is responsible for starting the benchmark and initiating postprocessing of benchmark results on the cloud VM. On the other side, the cloud VM uses the CWB client library to notify state updates and submit metrics (*i.e.*, observed benchmark results) back to the CWB server.

The architecture of CWB is based on the following principles:

- CWB is a framework and therefore designed for extensibility. It focuses on providing the infrastructure for defining, scheduling, and executing benchmarks. For this purpose, it de-

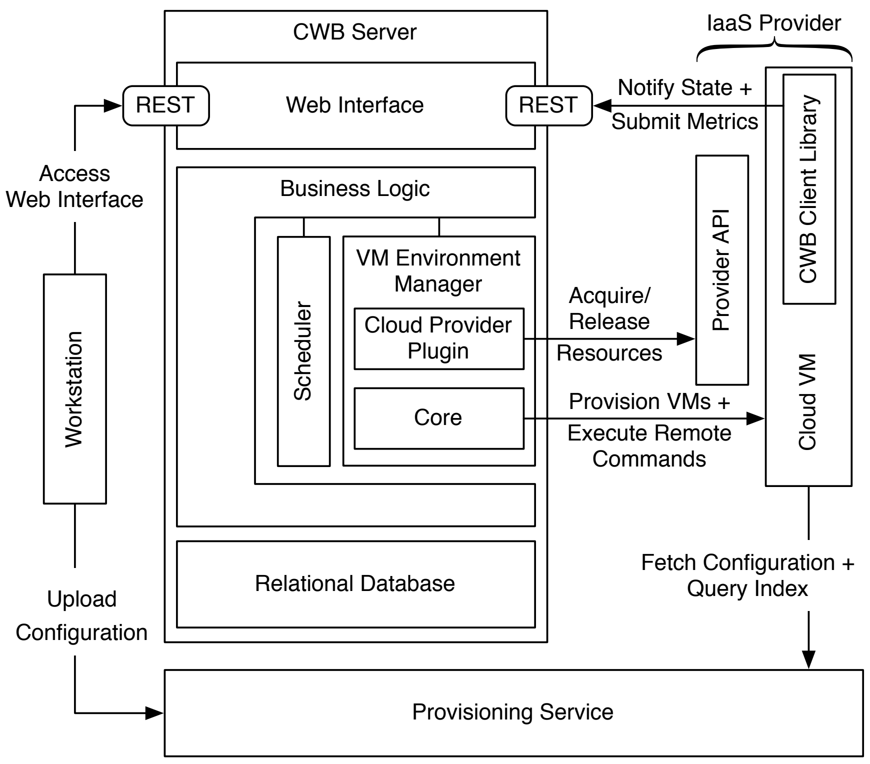


Figure 3.1: Architecture Overview

fines an interface and offers utilities to define a new benchmark with minimal effort. Experimenters are intended to create their own benchmarks as CWB has no predefined benchmarks with the exception of one sample benchmark.

- CWB does not reinvent the wheel and therefore builds upon existing tools. External software is abstracted by CWB wherever possible and exposed to the experimenter where justified by major enhancements in functionality.

- CWB is intended to offer its capabilities via a web interface that requires no local installation and can be accessed from any device.

3.2 Anatomy of a Benchmark

New benchmarks can be defined entirely through provisioning configurations and the CWB web interface. Provisioning configurations handle benchmark installation and multi-VM topology setup. For proper interaction with CWB, such user-defined configurations must adhere to the CWB framework interface. The CWB web interface then allows to create different variations of a benchmark, declare benchmark metrics, and schedule benchmark executions. These responsibilities are detailed in the following paragraphs.

CWB motivates experimenters to build their benchmarks with provisioning configurations. Thereby, the benchmark specific installation process is described by means of code making this process reproducible, modularizable, flexible, and testable by using software engineering techniques. Common components among benchmarks can be easily shared and provisioning configurations from a large provisioning service community can be reused to efficiently describe the benchmark installation. Multi-VM topologies require dynamic configuration since newly acquired cloud VMs receive a priori unknown IP addresses. To overcome this issue, cloud VMs can query the provisioning service's indices to retrieve data such as IP addresses about other VMs.

CWB defines an interface to handle interactions with user-defined benchmarks. Each benchmark must implement a hook (*i.e.*, a piece of code) to start its run and should use the provided CWB client utility library to notify state updates (*e.g.*, when the benchmark run is completed) and submit metrics to the CWB server. The CWB client library consists of a simple Ruby class that issues HTTP requests to the Representational State Transfer (REST) API of the CWB server. It is transparently installed and configured via provisioning configurations on each cloud VM to facilitate interaction with the CWB server.

Variations of a benchmark can be fully specified via the CWB web interface once the initial benchmark setup is completed, that is the provisioning configurations for benchmark installation are uploaded to the provisioning service. New variations are created either based on a generic template or by cloning an existing benchmark entity. Such a benchmark variation is mainly described by a concise VM environment DSL that specifies the cloud resources required in order to execute the benchmark and configures provisioning that handles benchmark installation. Creating a new benchmark entity further demands choosing a unique name, specifying a benchmark running timeout and selecting a supported cloud provider. Additionally, metrics that a benchmark will produce should be defined by name, type, and unit before benchmark executions are triggered manually via the web interface or automatically by a benchmark schedule. Such a schedule is expressed in the Cron syntax and associated with a benchmark entity.

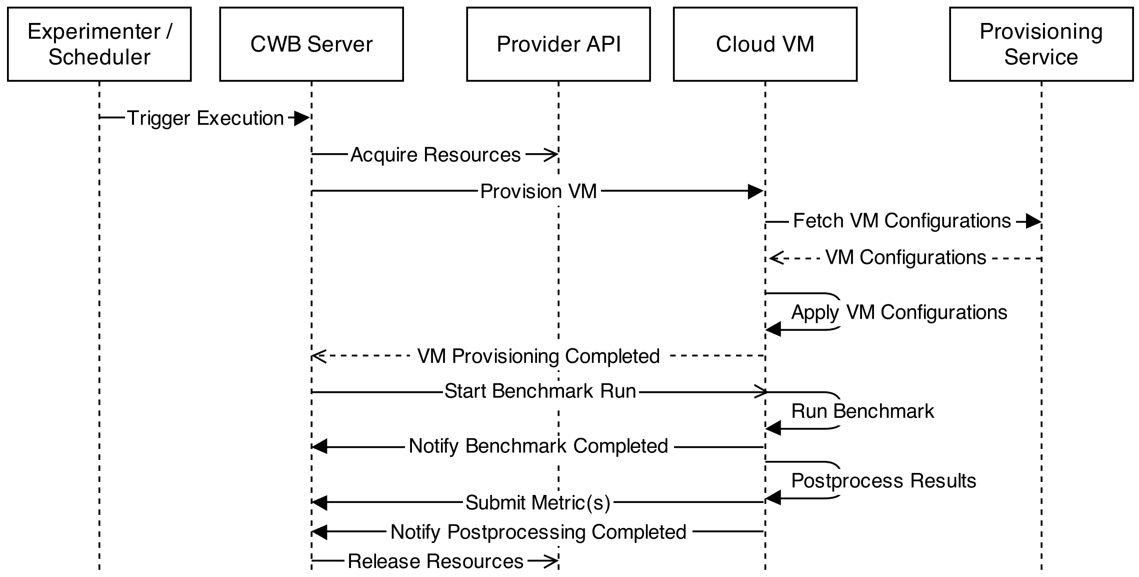


Figure 3.2: Interactions of a Benchmark Execution

3.3 Benchmark Execution

Figure 3.2 illustrates the interactions when a new benchmark execution is triggered from the experimenter or the scheduler. Subsequently, a plugin of the VM environment management tool asynchronously acquires cloud resources that are most importantly cloud VMs but may also comprise cloud specific features such as dedicated block storage[86] or dynamically mapped IP addresses[87]. As soon as the VM environment management tool successfully managed to establish a remote shell connection to the cloud VM, it starts orchestrating the VM provisioning via the remote shell connection. Thereby, each cloud VM fetches its role dependent configurations from the provisioning service and applies them. At this point, the benchmark is entirely prepared for running and asynchronously started via a remote shell command. This command invokes a programming language independent hook script that new benchmarks have to provide. Once the benchmark workload is completed, the hook script should notify this state update to the CWB server via the CWB client library. The benchmark results are then postprocessed, which typically involves textual result extraction, and submitted to the CWB server as individual metrics or as a collection of metrics via a Comma-Separated Values (CSV) file. After completed work, the cloud VM notifies the state update to the CWB server in order to trigger all resources being released.

3.4 Benchmark State Model

The event-based state model for benchmark executions is designed to present enough information to the experimenter so that he is able to track the flow of an execution and understand its current state. Events are stored in the database and subsequently used to calculate the state of the execution. This approach has the advantage of providing multiple perspectives, that is an event- and state-based view, from the same data. It further allows to associate additional data with events such as a timestamp of occurrence, an error code, and a textual message which can be used to store error logs. Concrete events and their corresponding states for benchmark executions were identified based on the data useful to logically track an execution and the data technically available or possible to retrieve in some reliable way. The following paragraphs present the state model illustrated in Figure 3.3.

An execution is *created* either manually via the web interface or automatically via a schedule. It is then *WAITING FOR START PREPARING* until the CWB server has processing capabilities available to start preparation. Immediately before starting preparation, the event *started preparing* is fired and subsequently, during preparation, the execution is in the *PREPARING* state. Unexpected and therefore unhandled exceptions during preparation cause the execution to enter the *FAILED ON PREPARING* state and release the acquired resources after a configurable timeout has been elapsed. This timeout gives the experimenter the opportunity to activate the interactive development mode, fix any provisioning errors, and reprovision the cloud VMs again. Interactive development mode introduces additional events and states that are not covered here as they are only relevant during development of a benchmark. After an execution has *finished preparing* it is *WAITING FOR START RUNNING* until the CWB server has free processing capabilities available. The benchmark run is then started on the cloud VM via a remote shell command resulting in a *started running* event in case of success and *failed on running* event and state on failure. Failures on start running and failures described in the following are treated the same way as failures on preparing, that is the acquired resources are released after a timeout has been elapsed.

A successfully started benchmark is *RUNNING* until a cloud VM notifies its completion or the specified running timeout from the benchmark definition has been elapsed. In the latter case the execution is being treated as *FAILED ON RUNNING* since it failed to complete within the expected time duration. The *FAILED ON RUNNING* state can also be reached if a cloud VM detects and notifies a failure. After *finished running*, a cloud VM may either immediately continue with postprocessing or enter the *WAITING FOR START POSTPROCESSING* state until the CWB server has processing capabilities available to trigger start postprocessing. This indirection is aimed to support multi-VM benchmarks where the responsibilities for recognizing benchmark completion and postprocessing are taken by distinctive cloud VMs. Subsequently, postprocessing follows the pattern of asynchronously executed remote commands (*e.g.*, running the benchmark) and releasing resources follows the pattern of locally executed commands (*e.g.*, preparing the benchmark). Releasing resources differs in exception handling because failures must be resolved manually by the experimenter since, beside retrying multiple times, there is no appropriate exception strategy applicable to automatically solve this problem. Executions without any failures remain in the *FINISHED* state after having *finished releasing resources*. Executions that exhibit at least one failure show their first failure state so that the experimenter can easily recognize at what step an execution has failed.

3.5 Types of Benchmark Results

The observed results of a benchmark execution, so called metrics, are represented differently based on the type of their definition. Metric definitions follow the established classification of nominal, ordinal, interval, and ratio scale from [Ste46]. Nominal scale metrics are stored as String data types whereas metrics of the other scale types are represented as floating point data types. This distinction enables efficient sorting at database level whereas presenting a uniform interface to the rest of the application by abstracting the implementation detail.

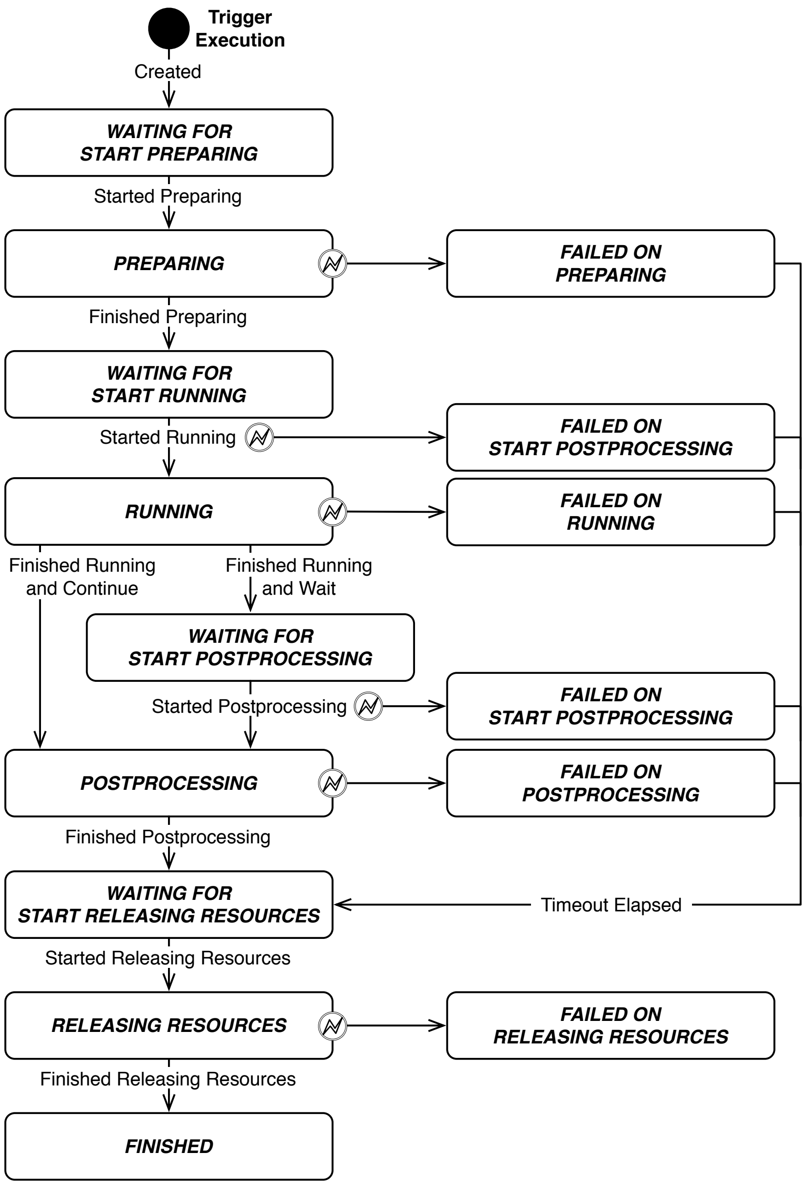

Trigger Execution
Created
WAITING FOR START PREPARING
Started Preparing
PREPARING
FAILED ON PREPARING
Finished Preparing
WAITING FOR START RUNNING
Started Running
FAILED ON START POSTPROCESSING
RUNNING
FAILED ON RUNNING
Finished Running and Continue
Finished Running and Wait
WAITING FOR START POSTPROCESSING
Started Postprocessing
FAILED ON START POSTPROCESSING
POSTPROCESSING
FAILED ON POSTPROCESSING
Finished Postprocessing
WAITING FOR START RELEASING RESOURCES
Timeout Elapsed
Started Releasing Resources
RELEASING RESOURCES
FAILED ON RELEASING RESOURCES
Finished Releasing Resources
FINISHED

Figure 3.3: State Model of a Benchmark Execution

3.6 Implementation and Deployment

The CWB web application is implemented with the Ruby on Rails[88] framework. Its business logic builds upon Cron as scheduler and Vagrant as VM environment management tool. Chef contributes industry-leading provisioning capabilities to CWB and offers essential features to conduct multi-VM benchmarks. Finally, the whole system is optimized for deployment in the cloud.

CWB is available as an open source project on Github at:

`https://github.com/sealuzh/cloud-workbench`

This repository comprises the CWB server, several example benchmarks as Chef cookbooks, and infrastructure code to automatically provision CWB. Detailed instructions are covered in the corresponding documentation.

3.6.1 Web Application

The web interface takes advantage of the popular Bootstrap[89] front-end framework and is visually enhanced with an open source[90] template and custom styling. It provides basic CRUD operations for the application entities where meaningful, context dependent tabular listing of entities, CSV export functionality, some basic search and filter operations, and live log refresh via Ajax. Figure 3.4 shows its ability to adapt to different types of devices by dynamically rearranging the user interface elements appropriately.

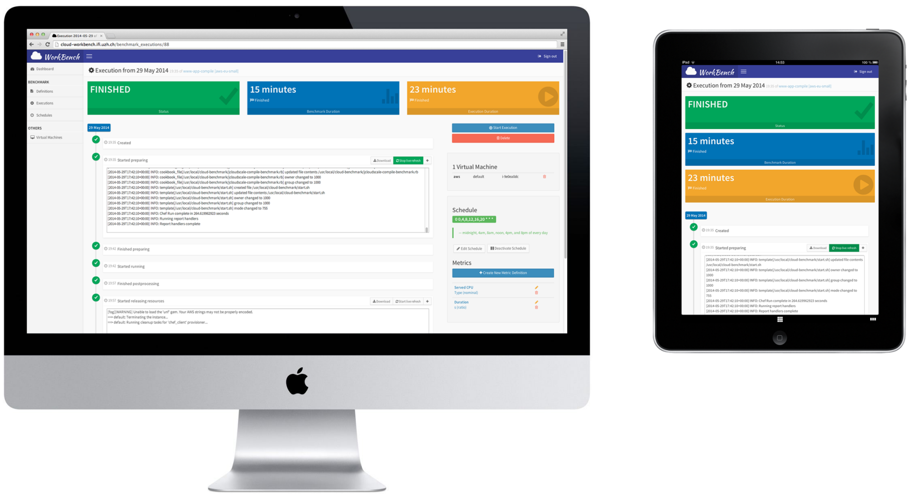


Figure 3.4: Responsive Web Interface

The business logic, implemented in the Ruby[91] programming language, mainly focuses on automating the benchmark execution. For this purpose, it operates on Rails application entities that are described with a relational data model. This model basically separates data that persists across multiple benchmark executions (*e.g.*, metric type) from execution related data (*e.g.*, observed benchmark result). Additionally, the business logic encapsulates the external dependencies Vagrant and Cron whose usage is detailed in the following paragraphs. In doing so, interactions with Vagrant involve long-running operations such as benchmark preparation that are unsuitable to be processed within the typical request/response model for web applications. Asynchronous background job processing is necessary in order to keep the web interface responsive and handle automatically scheduled jobs. Therefore, this kind of jobs is separated from the web server processes and executed in a dedicated pool of worker processes.

VM Environment Manager. Vagrant was chosen as a VM environment management tool since it provides a lot of functionality that CWB demands and otherwise would have to be implemented from scratch. It abstracts cloud provider APIs, provisioning orchestration, and the execution of remote shell commands. Although losing some flexibility, for example regarding the VM state information available, the Ruby DSL of Vagrant exposes the relevant configuration options in a declarative manner. Vagrant is successfully used in the Test Kitchen[92] test harness tool in a similar way as in CWB and thus follows the principle of building onto existing software as described in Section 3.1.

Scheduler. Cron, a UNIX system utility program for running periodically scheduled jobs, is used as scheduler to timely trigger benchmark executions. The periodic schedule is defined via the Cron expression syntax that allows to specify time intervals for minutely-, hourly-, daily- (in context of month and week), and monthly-based executions. The experimenter can directly enter a Cron expression in the web interface as a schedule. All active schedules are reflected into the operating system Cron utility which subsequently triggers new benchmark executions at the specified times.

3.6.2 Provisioning Service

Choosing Opscode Chef with a dedicated Chef server as provisioning service solves the problem of installing the CWB client library and more importantly provides a flexible way to install and configure benchmark components in a reusable manner by exploiting Chef attributes. Experimenters can reuse software components (*e.g.*, databases) in terms of cookbooks from a large Chef community and easily share benchmark infrastructure code with others. They manage cookbooks from their workstation once some initial authentication setup has been done.

Furthermore, Chef integrates particularly well with Vagrant. The attribute passing mechanism from Vagrant to Chef allows to build configurable and thus reusable benchmark cookbooks. Since both, Chef and Vagrant, use an internal Ruby DSL, they not only ensure language consistency across the project but also offer the capabilities of a fully featured programming language that is exploited with the use of variables and utility functions. Finally, Chef was preferred over the competitor Puppet mainly because of its querying capabilities and the knowledge that the open source version has been proven to work well as first-hand personal experience and information from [Dib13] has shown. The querying capabilities of the Chef-client mode are essential in order to support multi-VM benchmarks that require dynamic IP address configuration.

3.6.3 Deployment in the Cloud

CWB aims to be easily installable, configurable, and deployable on UNIX based operating systems in cloud environments. The installation of CWB commonly involves two separate machines including one for the CWB server and another for the Chef server. Installing the CWB web application for a production environment requires setting up a relational database, a Ruby application server, and a web proxy server. In addition to such a typical Ruby on Rails application setup, the CWB server must manage background worker processes, install Chef and Vagrant including its plugins, and configure all this software consistently. Deploying the web application code into the CWB server raises the configuration complexity even further. Since a manual installation would be too time-consuming and error-prone, the installation and deployment process of CWB is automated following the principles of Infrastructure-as-Code [Hüt12]. Automation of the installation and configuration process is implemented with Vagrant and Chef-solo and deployment is automated using Capistrano[93]. Finally, with the exception of some initial authentication and IP address configuration, CWB can be fully installed, configured, and deployed for operational usage on a new VM instance in the cloud with just a few shell commands.

Chapter 4

Case Study

This chapter describes experiments with a disk I/O micro-benchmark in the Amazon EC2 cloud conducted by using CWB. The goal of this study is to assess and compare the raw sequential write bandwidth of three general purpose instance types in combination with two dedicated block storage types. Specifically, it aims to answer the following questions regarding raw sequential write performance:

1. *When do larger instance types perform better than smaller instance types?*
2. *When should larger instance types be preferred over the better block storage type?*
3. *How do instance types and block storage types influence performance variability?*

4.1 Method

The data for this study was collected between June 20th and 23th in 2014 distributed over the day. Experiments were repeated for each setting 8 to 12 times depending on the observed variability.

Cloud Resources. All experiments were conducted in the Amazon EC2 region Ireland (eu-west-1) using the official Ubuntu 14.04 VM image from Canonical[94] publicly available as Amazon EC2 AMI with the identifier ami-896c96fe. Table 4.1a summarizes the specification of the used instance types with their current (June 23, 2014) prices[95]. Additionally, 20 GB of Amazon EBS was provisioned for each instance. Table 4.1b shows the two types of EBS storage used in the experiments. The general purpose storage type (gp2) was just announced 3 days before conducting the first experiments. Thus, this is the first study analyzing this newly available storage type.

Benchmark. The Flexible I/O Tester (FIO)[96] benchmark was chosen for several reasons. Its usage in literature indicates suitably for testing cloud infrastructures. Furthermore, it is under active development and offers many configuration options which is useful to adapt the benchmark to cloud environments and demonstrate the configuration capabilities of CWB. This study used the version 2.1.10 of the FIO benchmark compiled from source with gcc 4.8.2. Sequential write is performed with workloads of 1 Gibibyte (GiB) ($\sim$1074 MB) and 4 GiB ($\sim$4295 MB) using the default block size of 4 KiB (4096 bytes). Direct I/O mode is used in order to assess the raw write performance ignoring caches. Additionally, the refill buffers mode is enabled in order to prevent SSD compression effects.

Table 4.1: Experiment Resources

(a) Amazon EC2 Instances

Instance Type	Networking Performance	Price per Hour
t1.micro	Very Low	0.020$
m1.small	Low	0.047$
m3.medium	Moderate	0.077$

(b) Amazon EBS Types

EBS Type	Pricing
Magnetic Volumes	0.055$ per GB per month (rounded hourly) 0.055$ per 1 million I/O requests
General Purpose (SSD)	0.11$ per GB per month (rounded hourly)

Cloud WorkBench. A Chef cookbook was created that describes the FIO benchmark in a configurable manner. It automatically installs the benchmark and makes its parameters configurable via the CWB web interface by leveraging the Chef attribute model. This cookbook also generates Ruby code to start the execution, postprocess the results, submit the observed metrics, and notify state updates to the CWB server. Thereby, two metrics are defined and reported. Firstly, the CPU model name and secondly, a log of the bandwidth with the resolution of 500 milliseconds.

4.2 Results and Discussion

In the following, each case study question from above is answered by presenting and discussing the obtained results. Subsequently, further observations are noticed and the obtained results are compared with existing results from literature.

1. *When do larger instance types perform better than smaller instance types?*

Raw sequential write performance increases by about factor 4 for both EBS types when upgrading from the smallest instance type *t1.micro* to the next larger instance type *m1.small* or the even larger instance type *m3.medium*. Figure 4.1 illustrates this performance increase but also reveals that larger instance types (*m3.medium*) do not necessarily perform better than smaller instance types (*m1.small*).

The large difference between the smallest instance type *t1.micro* and the larger instance types *m1.small* and *m3.medium* may be explained with resource sharing and limited networking capabilities of the *t1.micro* instance type. As the only instance type from Amazon EC2, *t1.micro* shares its single CPU with another tenant. Therefore, it only gets half of the CPU cycles at maximum [WN10] which may affect the disk I/O performance [Gre13]. Networking performance influences the disk I/O performance since block storage is connected to the VM instance over the network. The networking performance specification provided by Amazon is very vague as shown by Table 4.1a. Thus, a potentially significantly slower networking performance of *t1.micro* may degrade its disk I/O performance. In order to assess these assumptions, further studies may correlate disk I/O performance with CPU and networking performance.

2. *When should larger instance types be preferred over the better block storage type?*

Larger instance types should be preferred over the better block storage type when using a *t1.micro* instance type. This is shown by Figure 4.1 since the absolute performance gain is much higher when upgrading from *t1.micro* to *m1.small* (+2750 KB/s) than when upgrading from the standard to the general purpose storage type (+250 KB/s). The combination of *m1.small* with standard EBS will cost more than *t1.micro* with general purpose EBS for block storage sizes below 350

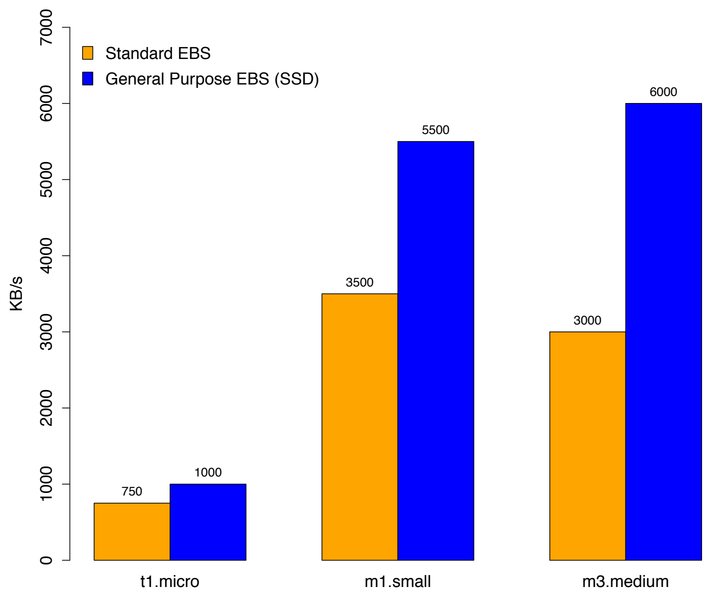


Figure 4.1: Sequential Write Bandwidth by Instance and Storage Type

GBs. However, without considering I/O operation expenses, the cost/performance ratio is always better for the *m1.small* and standard EBS combination. Disk I/O intensive applications with more than one million I/O requests per hour can shift this ratio in favor of the other combination (*t1.micro* with general purpose EBS).

Contrary, the better block storage type should be preferred over larger instance types when using a *m1.small* instance type. General purpose EBS can improve the performance of a *m1.small* instance type while the performance remains approximately the same when upgrading to the larger *m3.medium* instance type. Additionally, the cost/performance ratio is always better with the *m1.small* and general purpose EBS combination. Considering the expenses for I/O requests will even increase this advantage.

These results indicate that standard EBS is limited to approximately 3500 KB/s whereas the performance of the *t1.micro* instance type is restricted for other reasons. Similarly, general purpose EBS reaches approximately 6000 KB/s with the two larger instance types whereas its performance is restricted to about 1000 KB/s by the *t1.micro* instance type.

3. *How do instance types and block storage types influence performance variability?*

Although, in general, the disk I/O performance varies remarkably, two patterns were recognized when comparing the variability across and within distinct benchmark executions for different types of VM instances and EBS. Firstly, standard EBS exhibits larger variability than general purpose EBS for all instance types across and within distinct benchmark executions as shown by Table 4.2. Secondly, the *t1.micro* instance type exhibits a much larger variability within but not across distinct benchmark executions compared to the larger instance types. Table 4.2 shows this unusually high performance variability of 20 to 50 % for both EBS types within single benchmark executions for the *t1.micro* instance type.

The fact that this extraordinary high variability is equalized across distinct benchmark executions supports the assumption that CPU scheduling negatively influences the performance since the CPU scheduling effect is only recognizable in the high resolution performance analysis conducted within single benchmark executions.

Table 4.2: Sequential Write Bandwidth Variability (1 GiB)

	t1.micro	**m1.small**	**m3.medium**
Standard EBS	20% (20-50%)	20% (10-20%)	30% (15-60%)
General Purpose EBS (SSD)	10% (20-40%)	10% (5-15%)	10% (5-10%)

The variability is given as standard deviation in percentage of the mean across and within (in brackets) distinct benchmark executions.

Variability within executions is mostly ignored in literature and only the average value of single executions are collected. Although this makes sense in general, analyzing single executions in detail can help to better understand the nature of disk I/O performance. Figure 4.2 compares the sequential write performance over time for single executions of the instances types *t1.micro* and *m1.small* in combination with standard and general purpose EBS. It illustrates strong oscillation for both instance and storage types. This common behavior appears in combination with sudden performance drops that may turn out even stronger and endure even longer than illustrated by the curve *m1.small* with standard EBS especially around minute 15. In addition, this curve exemplifies the unpredictable performance behavior of standard EBS exhibiting arbitrary ups and downs. On the contrary, the bandwidth for general purpose EBS typically oscillates around the mean but still periodically drops in performance.

Further Observations. The observed CPU model name metrics have shown variability within the same instance type. Repeatedly acquiring the same instance types resulted in different CPU types being served for some instance types. This known phenomena from literature happened particularly often for the *t1.micro* instance type but also occasionally occurred for other instance types. The insufficient number of samples available made it impossible to correlate hardware specification with the measured performance as it was done for CPU performance in [LML$^+$11]. Ou *et al.* [OZN$^+$12] analyzed this kind of hardware heterogeneity in detail and presented strategies to exploit this phenomena. Such "placement gaming" strategies were further extended in [FJV$^+$12].

Comparision with Existing Results. Salah *et al.* [SASA$^+$11] conducted a comparable experiment with the same benchmark three years ago. Although the used CPU type matches with contemporary *t1.micro* instances, the overall instance type specification is rather comparable with contemporary small, medium, or even large instance types. In this sense, the results of this case study indicate sequential write performance improvements compared to the reported 2540 KB/s from 2011.

4.3 Threats to Validity

The small number of 8 to 12 samples per setting exhibiting high variability limits the statistical expressiveness of the results. Block storage volumes were not erased completely before running the benchmark as suggested by Amazon[97] in order to optimize performance. By analyzing raw sequential write performance, this study focused on a very specific performance aspect that may help to estimate bulk I/O-intensive application performance but has limited representativeness

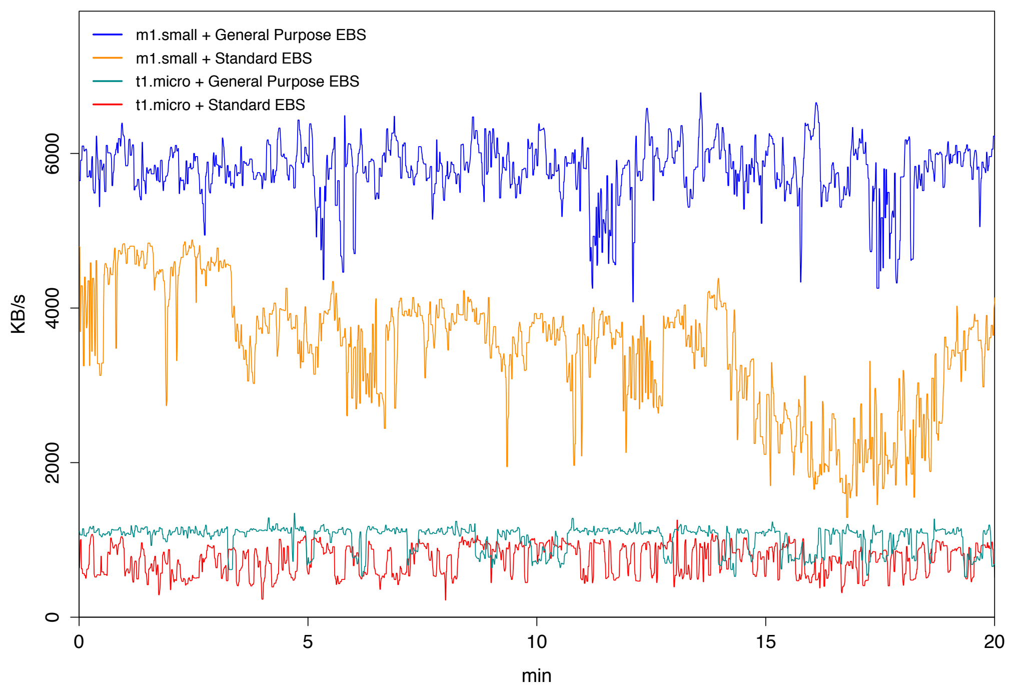


Figure 4.2: Sequential Write Bandwidth of Single Executions over Time

on its own. In particular, many reasons (*e.g.*, file system caching) could cause a mismatch between raw disk I/O and application I/O performance [Gre13]. Consequently, the results are less relevant for web applications whose majority of I/O request are typically served by much faster file system optimized I/O [Gre13]. The results should also be interpreted as a "snapshot-view" of a dynamic cloud computing environment. As a common issue in cloud benchmarking, continuously changing cloud infrastructures threaten reproducibility which apparently became true with Amazon's announcement of new instance types^{98} shortly after the experiments were completed. Individual sample experiments revealed a potential performance increase for micro and small instance types with standard EBS.

Chapter 5

Related Work

The need for supporting experimenters that conduct benchmarks in cloud environments have been recognized in literature and recently emerging studies focusing on automation frameworks indicate active research. Studies whose goals most closely match those of CWB include CloudBench [SHG^{+}13], Expertus [JSM^{+}12, JKC^{+}13], and Cloud Crawler [CMS13]. Despite pursuing similar objectives, different approaches were chosen.

CloudBench[99] [SHG^{+}13] proposes a mainly imperative approach [CMS13] for defining benchmarks at different levels of abstraction. New benchmarks are defined via a high-level experiment plan, one or multiple medium-level application or workload templates, and multiple low-level hook shell scripts. An experiment plan describes the behavior of a benchmark over time and is able to model joining and leaving VMs during an execution. Templates configure roles, hook shell scripts, and load distributions. Hook shell scripts are used to setup and orchestrate the benchmark execution. Furthermore, CloudBench offers its capabilities via a command line interface, a service API, and a graphical web interface. It also integrates a distributed and scalable metric collection system. The authors claim that CloudBench can *"represent and benchmark almost every observable interaction of a cloud"* [SHG^{+}13]. Comprehensive literature review has shown that CloudBench is currently the most sophisticated and extensive approach. Especially its capabilities to execute complex and dynamic scale-out benchmarks, as demanded in [BKKL09,CST^{+}10,FAK^{+}12], makes CloudBench unique.

Expertus, introduced in [JSM^{+}12] and extended in [JKC^{+}13], proposes a code generation based approach for defining benchmarks. New benchmarks are defined via XSLT[100] templates that generate shell scripts for the individual cloud VMs. Expertus further provides benchmark and workload configurability via XML, large-scale metric collection and a web interface with interactive visualization and statistical analysis capabilities.

Cloud crawler [CMS13] proposes a declarative approach for defining benchmarks. New benchmarks are defined via its own external DSL using a YAML[101] based syntax. Additionally, each benchmark must extend the Crawler execution engine by implementing a Java interface.

The following work matches similar goals than CWB. However, all of these approaches exhibit at least one major difference. They cannot be easily extended with additional benchmarks, require human interaction during benchmark execution, or are limited to certain types of benchmarks.

Smart CloudBench [CCVK13] provides a web interface to choose, configure, execute, and analyze predefined application benchmarks. Although mentioning its extensibility capabilities, experimenters are limited to the predefined benchmarks as the authors do not describe how to define new benchmarks. CloudHarmony[102], described in [GLOT13], further abstracts the idea pursued in Smart CloudBench and offers on demand execution of predefined benchmarks as a paid service . The large amount (more than 200 benchmarks with different configurations available as of July 2014) of predefined micro- and application benchmarks available alleviates the disadvantage of not being able to define new benchmarks. CloudCmp [LYKZ10b, LYKZ10a] is an open source[103] tool that systematically compares different cloud providers based on predefined representative workloads. Adding new benchmarks would require implementing a Java class that satisfies an undocumented interface.

Cloud-Gauge [ERR10] provides a web interface to manage benchmark executions, integrates real-time VM monitoring, but is restricted to private clouds. OLTP-Bench [DPCCM14, CDPCM12] is an open source[104] framework that automates benchmarking databases in the cloud with configurable workloads but does not integrate acquiring and releasing VM resources. C-Meter [YIEO09] focuses on describing and submitting workloads to experimenter managed cloud resources. It is used and extended with integrated VM resource management and metric analysis in [Ant12]. The CloudStone [SSS+08] web application benchmark is bundled with non-integrated automation scripts for preparation and tear down actions compatible with the Amazon EC2 cloud.

5.1 Comparison with Cloud WorkBench

None of the previously described approaches is known to offer provisioning capabilities to the same extent and with the same modularity as CWB does. There is also no solution known that integrates periodic scheduling functionality into a web-based framework. Furthermore, CWB together with CloudBench are the only frameworks designed for benchmark extensibility at runtime.

CloudBench supports portable benchmarks across cloud providers via its own provisioning solution using shell scripts. However, this approach has several limitations that are mainly originated in the nature of the shell scripting language. Its missing language features make sharing code among benchmarks difficult. The resulting duplication especially affects the maintainability of benchmark dependencies and is even aggravated by conditional, operating system or cloud provider specific code. CWB alleviates these problems by leveraging industry-leading provisioning technology with Chef. This also facilitates benchmark definition and development since there are tools available to debug and test infrastructure code. Finally, the integrated Ruby programming language is much more expressive and flexible than pure shell scripts.

The code generation approach from Expertus suffers from the same problems caused by shell scripts as CloudBench. Additionally, the flexibility and extensibility gained through XML and XSLT introduces complexity that makes debugging and describing new benchmarks difficult. CWB should be able to define benchmarks and its configuration much more concise by exploiting the convention over configuration design paradigm when reusing existing benchmark components or software packages available from the Chef community.

With Cloud Crawler, experimenters must manually provide VM images with preinstalled and preconfigured benchmarks. This approach does not scale for systematic benchmarking of multiple cloud providers having multiple regions. CWB solves this problem with Chef and furthermore provides an easier way to define new benchmarks than Cloud Crawler. The internal Ruby DSL of the Vagrantfile achieves similar expressiveness as the external YAML-based DSL of Cloud Crawler but offers more flexibility. Defining a new benchmark with Cloud Crawler is also too cumbersome and cannot be realized at runtime as it requires implementing a Java-based interface.

Chapter 6

Conclusion

This thesis presented a web-based framework called Cloud WorkBench (CWB) that supports experimenters in conducting IaaS cloud benchmarks. CWB was designed and implemented to automate the benchmarking lifecycle from the definition to the execution of benchmarks. Its extensibility allows to add additional benchmarks at runtime and support new cloud providers with minimal effort. Currently, CWB is tested with two cloud providers (Amazon EC2 and Google Compute Engine), has already executed nearly 20000 benchmarks completely autonomous, and is successfully used in an ongoing large-scale cloud evaluation.

In the following, the research questions from the first chapter are revisited:

Research Question I:
How can common IaaS cloud benchmarks from literature be defined in a modular and portable manner?

Cloud benchmarks are entirely defined by means of code making them configurable for reuse in a modular manner and portable across cloud providers and their regions. Hereby, benchmarks can easily share components among each other and even build upon existing software packages from a large community of the provisioning service. The CWB web interface contributes to modularity even further by allowing to easily create and configure variations of a benchmark. Overall, this generic approach is capable to model a wide diversity of micro- and application benchmarks exhibiting single- and multi-VM topologies. However, it has no integrated support for scale-out benchmarks that dynamically acquire and release cloud resources during their executions.

Research Question II:
How can benchmarks from research question I be periodically scheduled and reproducibly executed in cloud environments without manual interaction?

Standard system utilities and existing tools were combined to build a fully automated benchmark execution environment with periodic scheduling capabilities that is manageable via a web interface. The integrated provisioning service solves the benchmark installation problem by completely automating benchmark provisioning in a configurable manner. Furthermore, utilities are provided to facilitate metric collection and error handling strategies deal with most failures automatically. Thus, automation eliminates any error-prone human interactions threatening reproducibility.

In order to illustrate the capabilities of CWB, a case study with a disk I/O micro-benchmark was conducted wherein the raw sequential write performance of different types of VM instances and block storage in the Amazon EC2 cloud were compared. Results revealed the limited capabilities of the smallest instance type but has also shown that larger instance types do not necessarily perform better than smaller instance types. In that case, a newly announced type of block storage service could be recommended to cost-efficiently improve performance and reduce performance variability. Finally, variability within single benchmark executions and of the hardware being served were discussed and a comparison with existing results from literature indicated performance improvements of contemporary services.

Future Work. We plan to extend and improve CWB in a project with multiple master students. Specifically, our plans include:

- adding and testing additional cloud providers.
- automating the collection of common metrics such as VM startup time or CPU model name.
- integrating statistical analysis and visualization capabilities.
- facilitating benchmark definition.

The ultimate goal of CWB is to support the entire benchmarking lifecycle, from benchmark definition to the statistical analysis and visualization of the observed metrics, via a single web interface.

Appendix A

Endnotes

All the following link were verified on July 13, 2014.

[1]`https://aws.amazon.com/ec2/`

[2]`http://azure.microsoft.com/`

[3]`https://cloud.google.com/products/compute-engine/`

[4]`https://www.rackspace.com/cloud/`

[5]`https://www.openstack.org/`

[6]`https://www.cloudsigma.com/`

[7]`https://aws.amazon.com/ebs/`

[8]`https://www.rackspace.com/cloud/block-storage/`

[9]`https://aws.amazon.com/vpc/`

[10]`https://cloud.google.com/products/app-engine/`

[11]`https://aws.amazon.com/elasticbeanstalk/`

[12]`https://www.engineyard.com/`

[13]`https://www.openshift.com/`

[14]`https://www.heroku.com`

[15]`http://azure.microsoft.com/services/web-sites/`

[16]`https://www.salesforce.com/salesforce1/`

[17]`https://azure.microsoft.com/services/sql-database/`

[18]`https://aws.amazon.com/rds/`

[19]`https://developers.google.com/cloud-sql/`

[20]`https://aws.amazon.com/dynamodb/`

[21]`https://aws.amazon.com/simpledb/`

[22]`http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-tables/#what-is`

[23]`https://developers.google.com/datastore/`

[24]`https://cloud.google.com/products/bigquery/`

[25]`https://aws.amazon.com/s3/`

[26]`https://cloud.google.com/products/cloud-storage/`

[27]https://azure.microsoft.com/en-us/documentation/articles/storage-nodejs-how-to-use-blob-storage/#what-is

[28]https://www.rackspace.com/cloud/files/

[29]https://aws.amazon.com/sqs/

[30]https://aws.amazon.com/elasticmapreduce/

[31]http://www.picloud.com

[32]https://www.dropbox.com

[33]https://www.box.com

[34]https://evernote.com

[35]https://basecamp.com/

[36]http://www.google.com/enterprise/apps/business/

[37]http://office.microsoft.com/en-001/business/office-365-online-business-software-programs-FX102997619.aspx

[38]http://www.salesforce.com/

[39]http://www.microsoft.com/en-us/dynamics/crm.aspx

[40]https://www.facebook.com/

[41]http://www.google.com/+/learnmore/

[42]https://www.youtube.com/

[43]https://www.flickr.com/

[44]https://creative.adobe.com/

[45]http://www.netlib.org/benchmark/hpl/

[46]http://www.netlib.org/lapack/

[47]http://math-atlas.sourceforge.net/

[48]http://bitmover.com/lmbench/

[49]http://freecode.com/projects/unixbench

[50]http://www.eembc.org/coremark/about.php

[51]Bonnie: https://code.google.com/p/bonnie-64/,
Bonnie++: http://www.coker.com.au/bonnie++/

[52]http://git.kernel.org/cgit/linux/kernel/git/axboe/fio.git

[53]http://sourceforge.net/projects/filebench/

[54]https://www.samba.org/ftp/tridge/dbench/

[55]iperf2: http://iperf.sourceforge.net/,
iperf3: https://github.com/esnet/iperf

[56]http://www.netperf.org/netperf/

[57]http://www.mcs.anl.gov/research/projects/mpi/mpptest/

[58]http://www.cs.virginia.edu/stream/

[59]http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/

[60]http://icl.cs.utk.edu/projects/llcbench/cachebench.html

[61]http://redis.io/

[62]http://rubis.ow2.org/

[63]http://jmob.ow2.org/rubbos.html

[64]http://www.tpc.org/tpcw/

[65]https://geronimo.apache.org/GMOxDOC20/daytrader.html

[66]http://spec.org/web2005/

[67]Original: http://www.hpl.hp.com/research/linux/httperf/,
Current: https://code.google.com/p/httperf/

[68]http://www.nas.nasa.gov/publications/npb.html

[69]http://www.ks.uiuc.edu/Research/namd/

[70]https://code.google.com/p/mcgpu/

[71]http://www.dacapobench.org/

[72]http://www.spec.org/jvm2008/

[73]http://parsa.epfl.ch/cloudsuite/overview.html

[74]http://www.getchef.com/chef/

[75]https://github.com/opscode/chef

[76]http://www.vagrantup.com/

[77]https://github.com/mitchellh/vagrant

[78]https://www.virtualbox.org/

[79]Vagrant changelog: https://github.com/mitchellh/vagrant/blob/master/CHANGELOG.md

[80]https://github.com/mitchellh/vagrant/wiki/Available-Vagrant-Plugins#providers

[81]vagrant-aws: https://github.com/mitchellh/vagrant-aws

[82]vagrant-google: https://github.com/mitchellh/vagrant-google

[83]vagrant-azure: https://github.com/MSOpenTech/Vagrant-Azure

[84]vagrant-rackspace: https://github.com/mitchellh/vagrant-rackspace

[85]http://puppetlabs.com/

[86]Amazon EBS storage: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html

[87]OpenStack floating IP: http://docs.openstack.org/user-guide/content/floating_ip_allocate.html

[88]http://rubyonrails.org/

[89]http://getbootstrap.com/

[90]AdminLTE template: https://github.com/almasaeed2010/AdminLTE

[91]https://www.ruby-lang.org/

[92]http://kitchen.ci/

[93]http://capistranorb.com/

[94]https://cloud-images.ubuntu.com/locator/ec2/

[95]https://aws.amazon.com/ec2/pricing/

[96]http://git.kernel.org/cgit/linux/kernel/git/axboe/fio.git

[97]http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/benchmark_piops.html

[98]http://aws.amazon.com/about-aws/whats-new/2014/07/01/introducing-t2-the-new-low-cost-general-purpose-instance-type-for-amazon-ec2/

[99]https://github.com/ibmcb/cbtool

[100]http://www.w3.org/standards/xml/transformation

[101]http://www.yaml.org/

[102]https://cloudharmony.com/benchmarks

[103]https://github.com/angl/cloudcmp

[104]https://github.com/oltpbenchmark/oltpbench

Appendix B

Abbreviations

ACID Atomicity, Consistency, Isolation, and Durability

Ajax Asynchronous JavaScript and XML

CAP Consistency, Availability, and Partition Tolerance

CPU Central Processing Unit

CRUD Create, Read, Update, and Delete

CSV Comma-Separated Values

CWB Cloud WorkBench

DSL Domain Specific Language

EC2 Elastic Compute Cloud

EBS Elastic Block Storage

FIO Flexible I/O Tester

GFLOPS Giga Floating Point Operations per Second

GiB Gibibyte

GOPS Giga Operations per Second

GPU Graphical Processing Unit

GUPS Giga Updates per Second

HDD Hard Disk Drive

HPC High Performance Computing

IaaS Infrastructure-as-a-Service

NIST National Institute of Standards and Technology

PaaS Platform-as-a-Service

REST Representational State Transfer

RTT Round Trip Time

SaaS Software-as-a-Service

SLA Service Level Agreement

SSD Solid State Disk

SSH Secure Shell

SUT System Under Test

VM Virtual Machine

VPN Virtual Private Network

WIPS Web Interactions Processed per Second

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformation

Bibliography

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`.

[AI10] Syed A. Ahson and Mohammad Ilyas. *Cloud Computing and Software Services: Theory and Techniques*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010. URL: `http://www.crcpress.com/product/isbn/9781439803158`.

[AM10] Sayaka Akioka and Yoichi Muraoka. Hpc benchmarks on amazon ec2. In *24th IEEE International Conference on Advanced Information Networking and Applications (AINA) Workshops*, pages 1029–1034, April 2010. `doi:10.1109/WAINA.2010.166`.

[Ant12] Athanasios Antoniou. Performance evaluation of cloud infrastructure using complex workloads. Master's thesis, Delft University of Technology, 2012. URL: `http://repository.tudelft.nl/view/ir/uuid%3Ad8eda846-7e93-4340-834a-de3e4aa93f8b/`.

[BBB+91] David H. Bailey, Eric Barszcz, John T. Barton, David S. Browning, Russell L. Carter, Leonardo Dagum, Rod A. Fatoohi, Paul O. Frederickson, Thomas A. Lasinski, Rob S. Schreiber, et al. The nas parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991. `doi:10.1177/109434209100500306`.

[BBG11] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud computing: Principles and Paradigms*, volume 87. John Wiley & Sons, March 2011. URL: `http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470887990.html`.

[BGH+06] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. The dacapo benchmarks: Java benchmarking development and analysis. In *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06)*, pages 169–190, 2006. `doi:10.1145/1167473.1167488`.

[BK09] Christian Baun and Marcel Kunze. Building a private cloud with eucalyptus. In *5th IEEE International Conference on E-Science Workshops*, pages 33–38, December 2009. `doi:10.1109/ESCIW.2009.5408006`.

[BKKL09] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. How is the weather tomorrow?: Towards a benchmark for the cloud. In *Proceedings of the Second International Workshop on Testing Database Systems (DBTest '09)*, pages 9:1–9:6. ETH Zurich, 2009. doi:10.1145/1594156.1594168.

[BKSL08] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08)*, pages 72–81, 2008. doi:10.1145/1454115.1454128.

[Bre12] Eric Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, February 2012. doi:10.1109/MC.2012.37.

[BYV$^+$09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009. doi:10.1016/j.future.2008.12.001.

[CCVK13] Mohan Baruwal Chhetri, Sergei Chichin, Quoc Bao Vo, and Ryszard Kowalczyk. Smart cloudbench – automated performance benchmarking of the cloud. In *Sixth IEEE International Conference on Cloud Computing (CLOUD)*, pages 414–421, June 2013. doi:10.1109/CLOUD.2013.7.

[CDPCM12] Carlo Curino, Djellel E. Difallah, Andrew Pavlo, and Philippe Cudré-Mauroux. Benchmarking oltp/web databases in the cloud: The oltp-bench framework. In *Proceedings of the Fourth International Workshop on Cloud Data Management (CloudDB '12)*, pages 17–20, 2012. doi:10.1145/2390021.2390025.

[Che13] Opscode Chef. Facebook likes opscode and private chef [online]. February 2013. Chef announcing Facebook reference. URL: http://www.getchef.com/press-releases/facebook-likes-opscode-and-private-chef/ [cited 2014-06-03].

[CMS13] Matheus Cunha, Nabor Mendonça, and Américo Sampaio. A declarative environment for automatic performance evaluation in iaas clouds. In *Sixth IEEE International Conference on Cloud Computing (CLOUD)*, pages 285–292, June 2013. doi:10.1109/CLOUD.2013.12.

[CST$^+$10] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*, pages 143–154, 2010. doi:10.1145/1807128.1807152.

[Cun08] H. Conrad Cunningham. A little language for surveys: Constructing an internal dsl in ruby. In *Proceedings of the 46th Annual Southeast Regional Conference on XX (ACM-SE 46)*, pages 282–287, 2008. doi:10.1145/1593105.1593181.

[CUWS11] Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood, and Prashant Shenoy. Benchlab: An open testbed for realistic benchmarking of web applications. In *Proceedings of the 2nd USENIX Conference on Web Application Development (WebApps'11)*, pages 4–4. USENIX Association, 2011. URL: http://dl.acm.org/citation.cfm?id=2002168.2002172.

[Dib13] Phil Dibowitz. Scaling systems configuration at facebook [online]. April 2013. Keynote at ChefConf 2013 in San Francisco. URL: `http://youtu.be/SYZ2GzYAw_Q` [cited 2014-06-06].

[DMM+10] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU '10)*, pages 63–74, 2010. `doi:10.1145/1735688.1735702`.

[dOBM10] Daniel de Oliveira, Fernanda Araujo Baião, and Marta Mattoso. Towards a taxonomy for cloud computing from an e-science perspective. In *Cloud Computing*, Computer Communications and Networks, pages 47–62. Springer, 2010. `doi:10.1007/978-1-84996-241-4_3`.

[DPCCM14] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment*, 7(4), 2014. URL: `http://www.vldb.org/pvldb/vol7/p277-difallah.pdf`.

[ERR10] Mohamed A. El-Refaey and Mohamed Abu Rizkaa. Cloudgauge: A dynamic cloud and virtualization benchmarking suite. In *19th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, pages 66–75, June 2010. `doi:10.1109/WETICE.2010.17`.

[ETR+13] Roberto R. Expósito, Guillermo L. Taboada, Sabela Ramos, Juan Touriño, and Ramón Doallo. General-purpose computation on gpus for high performance cloud computing. *Concurrency and Computation: Practice and Experience*, 25(12):1628–1642, 2013. `doi:10.1002/cpe.2845`.

[FAK+12] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *SIGARCH Computer Architecture News*, 40(1):37–48, March 2012. `doi:10.1145/2189750.2150982`.

[FAS+13] Enno Folkerts, Alexander Alexandrov, Kai Sachs, Alexandru Iosup, Volker Markl, and Cafer Tosun. Benchmarking in the cloud: What it should, can, and cannot be. In *Selected Topics in Performance Evaluation and Benchmarking*, volume 7755 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2013. `doi:10.1007/978-3-642-36727-4_12`.

[FJV+12] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D. Bowers, and Michael M. Swift. More for your money: Exploiting performance heterogeneity in public clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*, pages 20:1–20:14, 2012. `doi:10.1145/2391229.2391249`.

[GGW10] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *International Conference on Network and Service Management (CNSM)*, pages 9–16, October 2010. `doi:10.1109/CNSM.2010.5691343`.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002. `doi:10.1145/564585.564601`.

[GLOT13] Lee Gillam, Bin Li, John O'Loughlin, and Anuz Tomar. Fair benchmarking for cloud computing systems. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):6, 2013. doi:10.1186/2192-113X-2-6.

[Gre13] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Prentice Hall, 2013. URL: http://books.google.ch/books?id=pTYkAQAAQBAJ.

[GSR13] Prashant Gupta, A. Seetharaman, and John Rudolph Raj. The usage and adoption of cloud computing by small and medium businesses. *International Journal of Information Management*, 33(5):861 – 874, 2013. doi:10.1016/j.ijinfomgt.2013.07.001.

[GVB13] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4):1012 – 1023, 2013. doi:10.1016/j.future.2012.06.006.

[HHD+10] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *26th IEEE International Conference on Data Engineering Workshops (ICDEW)*, pages 41–51, March 2010. doi:10.1109/ICDEW.2010.5452747.

[Hil09] David Hilley. Cloud computing: A taxonomy of platform and infrastructure-level offerings. Technical Report GIT-CERCS-09-13, Georgia Institute of Technology, 2009. URL: http://www.cercs.gatech.edu/tech-reports/tr2009/git-cercs-09-13.pdf.

[HK10] Christina N. Höfer and Georgios Karagiannis. Taxonomy of cloud computing services. In *IEEE Globecom Workshops*, pages 1345–1350. IEEE, December 2010. doi:10.1109/GLOCOMW.2010.5700157.

[HK11] Christina N. Höfer and Georgios Karagiannis. Cloud computing services: taxonomy and comparison. *Journal of Internet Services and Applications*, 2(2):81–94, 2011. doi:10.1007/s13174-011-0027-x.

[Hüt12] Michael Hüttermann. *DevOps for Developers*. Apress, 2012. doi:10.1007/978-1-4302-4570-4_9.

[HZK+10] Qiming He, Shujia Zhou, Ben Kobler, Dan Duffy, and Tom McGlynn. Case study for running hpc applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, pages 395–401, 2010. doi:10.1145/1851476.1851535.

[IHJ11] Shadi Ibrahim, Bingsheng He, and Hai Jin. Towards pay-as-you-consume cloud computing. In *IEEE International Conference on Services Computing (SCC)*, pages 370–377, July 2011. doi:10.1109/SCC.2011.38.

[IOY+11] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, June 2011. doi:10.1109/TPDS.2011.66.

[JKC+13] Deepal Jayasinghe, Josh Kimball, Siddharth Choudhary, Tao Zhu, and Calton Pu. An automated approach to create, store, and analyze large-scale experimental data in clouds. In *14th IEEE International Conference on Information Reuse and Integration (IRI)*, pages 357–364, August 2013. doi:10.1109/IRI.2013.6642493.

[JMQ+11] Deepal Jayasinghe, Simon Malkowski, Wang Qingyang, Jack Li, Pengcheng Xiong, and Calton Pu. Variations in performance and scalability when migrating n-tier applications to different clouds. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 73–80, July 2011. doi:10.1109/CLOUD.2011.43.

[JRM+10] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Second IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 159–168, November 2010. doi:10.1109/CloudCom.2010.69.

[JSM+12] Deepal Jayasinghe, Galen Swint, Simon Malkowski, Jack Li, Qingyang Wang, Junhee Park, and Calton Pu. Expertus: A generator approach to automate performance testing in iaas clouds. In *5th IEEE International Conference on Cloud Computing (CLOUD)*, pages 115–122, June 2012. doi:10.1109/CLOUD.2012.98.

[JWZ+13] Zhen Jia, Lei Wang, Jianfeng Zhan, Lixin Zhang, and Chunjie Luo. Characterizing data analysis workloads in data centers. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 66–76, September 2013. doi:10.1109/IISWC.2013.6704671.

[Kat97] Jeffrey Katcher. Postmark: A new file system benchmark. Technical Report TR3022, Network Appliance, Inc., 1997. URL: http://wangmir.com/content/down/Katcher97-postmark-netapp-tr3022.pdf.

[KKL10] Donald Kossmann, Tim Kraska, and Simon Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*, pages 579–590, 2010. doi:10.1145/1807167.1807231.

[KSHD13] Steffen Kächele, Christian Spann, Franz J. Hauck, and Jörg Domaschka. Beyond iaas and paas: An extended cloud taxonomy for computation, storage and networking. In *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pages 75–82, December 2013. doi:10.1109/UCC.2013.28.

[LBD+06] Piotr R. Luszczek, David H. Bailey, Jack J. Dongarra, Jeremy Kepner, Robert F. Lucas, Rolf Rabenseifner, and Daisuke Takahashi. The hpc challenge (hpcc) benchmark suite. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '06)*, 2006. doi:10.1145/1188455.1188677.

[LF13] Hung LeHong and Jackie Fenn. Hype cycle for emerging technologies. *Gartner*, August 2013. URL: http://www.gartner.com/newsroom/id/2575515.

[LML+11] Alexander Lenk, Michael Menzel, Johannes Lipsky, Stefan Tai, and Philipp Offermann. What are you paying for? performance benchmarking for infrastructure-as-a-service offerings. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 484–491, July 2011. doi:10.1109/CLOUD.2011.80.

[LTG+14] Lydia Leong, Douglas Toombs, Bob Gill, Gregor Petri, and Tiny Haynes. Magic quadrant for cloud infrastructure as a service. *Gartner*, May 2014. URL: http://www.gartner.com/technology/reprints.do?id=1-1UKQQA6&ct=140528&st=sb.

[LW09] Huan Liu and Sewook Wee. Web server farm in the cloud: Performance evaluation and dynamic architecture. In *Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 2009. doi:10.1007/978-3-642-10665-1_34.

[LYKZ10a] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*, pages 1–14, 2010. doi:10.1145/1879141.1879143.

[LYKZ10b] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: Shopping for a cloud made easy. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*, 2010. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=136451.

[LZK$^+$11] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. Cloudprophet: Towards application performance prediction in cloud. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*, pages 426–427, 2011. doi:10.1145/2018436.2018502.

[Mah13] Zaigham Mahmood. *Cloud Computing: Methods and Practical Approaches*. Computer Communications and Networks. Springer, 2013.

[MC13] Lorraine Morgan and Kieran Conboy. Key factors impacting cloud computing adoption. *Computer*, 46(10):97–99, October 2013. doi:10.1109/MC.2013.362.

[MG11] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), September 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[MH12] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *5th IEEE International Conference on Cloud Computing (CLOUD)*, pages 423–430, Juni 2012. doi:10.1109/CLOUD.2012.103.

[MJ98] David Mosberger and Tai Jin. Httperf - a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998. doi:10.1145/306225.306235.

[MLT98] Philips J. Mucci, Kevin London, and John Thurman. The cachebench report. *CEWES MSRC/PET*, 19(TR/98-25), March 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.6186&rep=rep1&type=pdf.

[MVML13] Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente. Key challenges in cloud computing: Enabling the future internet of services. *IEEE Internet Computing*, 17(4):18–25, July 2013. doi:10.1109/MIC.2012.69.

[Nad14] Satya Nadella. Mobile first, cloud first [online]. March 2014. Press Briefing of Microsoft CEO. URL: http://www.microsoft.com/en-us/news/speeches/2014/03-27nadella.aspx [cited 2014-07-07].

[NB09] Jeffrey Napper and Paolo Bientinesi. Can cloud computing reach the top500? In *Proceedings of the Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop (UCHPC-MAW '09)*, pages 17–20, 2009. doi:10.1145/1531666.1531671.

[OIY+10] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of ec2 cloud computing services for scientific computing. In *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, pages 115–131. Springer, 2010. `doi:10.1007/978-3-642-12636-9_9`.

[OZN+12] Zhonghong Ou, Hao Zhuang, Jukka K. Nurminen, Antti Ylä-Jääski, and Pan Hui. Exploiting hardware heterogeneity within the same instance type of amazon ec2. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'12)*, 2012. URL: `http://dl.acm.org/citation.cfm?id=2342763.2342767`.

[RBD+12] Eduardo Roloff, Francis Birck, Matthias Diener, Alexandre Carissimi, and Philippe O. A. Navaux. Evaluating high performance computing on the windows azure platform. In *5th IEEE International Conference on Cloud Computing (CLOUD)*, pages 803–810, June 2012. `doi:10.1109/CLOUD.2012.47`.

[RCL09] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *5th International Joint Conference on INC, IMS and IDC (NCM '09)*, pages 44–51, August 2009. `doi:10.1109/NCM.2009.218`.

[RMP10] Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker: Efficient wide-area live virtual machine migration using distributed content-based addressing. Technical Report RR-7198, INRIA, February 2010. URL: `http://hal.inria.fr/inria-00454727`.

[SASA+11] Khaled Salah, M. Al-Saba, M. Akhdhor, O. Shaaban, and M.I. Buhari. Performance evaluation of popular cloud iaas providers. In *6th International Conference on Internet Technology and Secured Transactions (ICITST)*, pages 345–349, December 2011.

[SBC+13] James Staten, Dave Bartoletti, Andras Cser, John Kindervag, Rachel A. Dines, Lauren E. Nelson, Liz Herbert, Christopher Voce, and Heather Belanger. Predictions for 2014: Cloud computing. *Forrester Research, Inc.*, 12 2013.

[SBDR05] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. Improving accuracy in end-to-end packet loss measurement. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '05)*, pages 157–168, 2005. `doi:10.1145/1080091.1080111`.

[SBV+09] Alexander S. Szalay, Gordon Bell, Jan Vandenberg, Alainna Wonders, Randal Burns, Dan Fay, Jim Heasley, Tony Hey, Maria Nieto-SantiSteban, Ani Thakar, Catharine van Ingen, and Richard Wilton. Graywulf: Scalable clustered architecture for data intensive computing. In *42nd Hawaii International Conference on System Sciences (HICSS '09)*, pages 1–10, January 2009. `doi:10.1109/HICSS.2009.234`.

[SDQR10] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1):460–471, September 2010. `doi:10.14778/1920841.1920902`.

[SHG+13] M. Silva, M.R. Hines, D. Gallo, Qi Liu, Kyung Dong Ryu, and D. Da Silva. Cloudbench: Experiment automation for cloud environments. In *IEEE International Conference on Cloud Engineering (IC2E)*, pages 302–311, March 2013. `doi:10.1109/IC2E.2013.33`.

[ŠS11] Vladimir Šor and Satish Narayana Srirama. A statistical approach for identifying memory leaks in cloud applications. In *First International Conference on Cloud Computing and Services Science (CLOSER 2011)*, pages 623–628, 2011.

[SSGW11] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, pages 5:1–5:14, 2011. `doi:10.1145/2038916.2038921`.

[SSS$^+$08] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, Armando Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0, 2008. URL: `http://cyberaide.googlecode.com/svn/trunk/misc/cloud-papers/cca08/33.pdf`.

[Ste46] Stanley Smith Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, June 1946.

[TMV$^+$11] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. The impact of memory subsystem resource sharing on datacenter applications. In *38th Annual International Symposium on Computer Architecture (ISCA)*, pages 283–294, June 2011.

[UN10] Yohai Ueda and Toshio Nakatani. Performance variations of two open-source cloud platforms. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10, December 2010. `doi:10.1109/IISWC.2010.5650280`.

[VRMCL08] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, January 2008. `doi:10.1145/1496091.1496100`.

[Wal08] Edward Walker. Benchmarking amazon ec2 for high-performance scientific computing. *Usenix Login*, 33(5):18–23, October 2008.

[WJC$^+$10] Hongyi Wang, Qingfeng Jing, Rishan Chen, Bingsheng He, Zhengping Qian, and Lidong Zhou. Distributed systems meet economics: Pricing in the cloud. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10)*, 2010.

[WN10] Guohui Wang and T. S. Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proceedings IEEE INFOCOM*, pages 1–9, March 2010. `doi:10.1109/INFCOM.2010.5461931`.

[WOT$^+$95] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA '95)*, pages 24–36, 1995. `doi:10.1145/223982.223990`.

[YBS08] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop (GCE '08)*, pages 1–10, November 2008. `doi:10.1109/GCE.2008.4738443`.

[YIEO09] Nezih Yigitbasi, Alexandru Iosup, Dick Epema, and Simon Ostermann. C-meter: A framework for performance analysis of computing clouds. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 472–477, 2009. `doi:10.1109/CCGRID.2009.40`.

[ZCB10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010. doi:10.1007/s13174-010-0007-6.

[Zha01] Xiaolan Zhang. *Application-specific benchmarking*. PhD thesis, Harvard University Cambridge, Massachusetts, 2001. URL: http://www.eecs.harvard.edu/~syrah/application-spec-benchmarking/publications/thesis.pdf.

[ZL11] Gong Zhang and Ling Liu. Why do migrations fail and what can we do about it? In *Proceedings of the 25th International Conference on Large Installation System Administration (LISA'11)*, pages 25–25, 2011. URL: https://www.usenix.org/legacy/events/lisa11/tech/full_papers/Zhang.pdf.