Comparative Assessment of Cloud Compute Services using Run-Time Meta-Data

Michael Menzel

Comparative Assessment of Cloud Compute Services using Run-Time Meta-Data

A Framework for Performance Measurements and Virtual Machine Image Introspections

vorgelegt von

Dipl. Wirt.-Inf. Michael Menzel

geb. in Mainz

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Tag der wissenschaftlichen Aussprache: 21.04.2015

Berlin 2015

Credits

As most projects in life, a dissertation stands on the shoulders of giants. Foremost, it is the result of collaborations, ongoing discussions, and inspirations from many acquaintances, companions, and friends. A particularly inspiring giant that supervised the present work is Prof. Stefan Tai whom I like to thank for his patient advises, engagement, and steady support of this project. Furthermore, I am deeply thankful I had the luck to spend time with the most intriguing colleagues[1]. The way all of them engage in discussions and strive for the best is remarkable.

Companionships between like-minded researchers develop unaffected by distance or time zones. I am thankful to have met Rajiv Ranjan and Shrikumar Venugopal at the University of New South Wales. Rajiv accompanied and contributed to some of my publications in the research community. He has been a valuable counterpart in discussions. The arrangements never allowed me to discuss my research topics with Shrikumar as intensively as I would've liked to, but his few hints unfolded as crucial advises.

The support I have received from my friends, especially from Sebastian Roth and Jeanette Homenu-Roth & Tobias Roth, will never be forgotten. Last but even more notably, I am grateful for my parents Sylvia and Wolfgang, and siblings Julia Konstanze and Laura-Sophia, who never doubted I was able to complete this dissertation.

Above all, it is nearly impossible to express how outermost thankful I am for the unending support from one special person. Knowing her for so many years, there exists no word to describe how gratified and lucky I must be to share life with such a patient, understanding, and loving person as is my true love, Anna-Karolina Braun. She continually encouraged me to pursue a PhD and to engage in a project which is as demanding and fulfilling as is this dissertation. Thank you!

Mannheim, April 2015 *Michael Menzel*

[1] In alphabetic order: Alexander Lenk, Bugra Derre, Christian Janiesch, Christian Zirpins, David Bermbach, Erik Wittern, Frank Pallas, Gregory Katsaros, Jens Nimis, Jörn Kuhlenkamp, Markus Klems, Nelly Schuster, Raffael Stein, Robin Fischer, Steffen Müller, Tilmann Kopp, Ulrich Scholten

Abstract

The available amount of meta-data about compute service offerings which proofs reliable, timely, and comparable is unsatisfactory. For example, the meta-data published by compute service providers regarding performance attributes of their offers is typically restricted to hardware figures and, thus, not necessarily sufficient for comparisons or planning tasks, such as a thorough software system capacity planning. A similar problem of meta-data scarcity affects the reuse of Virtual Machine (VM) images available in repositories from compute service providers. The contents of the VM images are not described by any available meta-data, yet.

The present work contributes a framework of compute service assessment and comparison methods to the research community. The methods enables compute cloud consumers to assess and compare compute services regarding diverse characteristics. As the purpose of the methods is to serve consumers, the general scheme is an exploitation of the client-side remote access to VMs in order to gain meta-data at run-time. Therefore, an archetypical run-time assessment automation model is provided. The information extracted at run-time can furthermore be attached to compute services as meta-data through a generic and extensible meta-data model. Furthermore, a Multi-Attribute Decision-Making (MADM)-based scoring method is introduced by the framework which enables consumers to compare compute services regarding multiple characteristics with a single score. Besides, a stopping rule approach is able to enforce cost budgets during sequential compute service assessments. Additionally, in a search for a highest scoring compute service the rule uses priorly available meta-data to skip presumably low scoring services.

The framework is employed in two specific instantiations to assess compute services in regards of performance and VM image contents. In particular, this work features an instantiation of the framework which assesses compute services using performance measurements. Therefore, a method is presented that incorporates a procedure built upon the framework's automation model. The method uses the procedure to measure compute services by injecting benchmarking scripts into VMs via remote interfaces from client-side. Aside from the procedure, the method features a definition language for configuring batches of performance measurements with repetitions and scheduled runs. Resulting performance meta-data can be captured with an extension of the framework's meta-data model. For the comparison of performance meta-data gained with the benchmarking method, the framework's MADM-based method is adapted. The adapted method enables compute cloud consumers to compare compute services according to a score derived from a custom set of performance attributes. In the quest of containing costs from benchmarking various compute services, the framework's stopping rule is configured to consider information about compute service hardware figures available beforehand. In regards of VM image repositories of compute services, the proposed framework is instantiated to generate a procedure to introspect the contents of VM images from the client-side. An extension of the framework's meta-data model is able to capture results from VM image introspections.

The feasibility of the methods is confirmed by software prototypes presented in this work. The prototypes, furthermore, serve as a basis to conduct an evaluation of the proposed methods and models. Several experiments proof the validity of the approaches. Further examinations in the course of an evaluation address aspects such as the promptitude of the assessment approaches and the computational complexity to combine multiple meta-data attributes into a score. Besides, the methods are compared to the state of the art to expose advantages and shortcomings.

Zusammenfassung

Die Menge verfügbarer Meta-Daten über Compute Service-Angebote erweist sich hinsichtlich Zuverlässigkeit, Aktualität und Vergleichbarkeit ungenügend. Zum Beispiel beschränken sich die von Compute Service-Anbietern veröffentlichten Meta-Daten über Leistungsattribute ihrer Angebote typischerweise auf Hardwarekennzahlen und sind daher nicht verwendbar für Vergleiche oder auch Planungsaufgaben wie bspw. eine Kapazitätsplanung für ein Softwaresystem. Ähnlich sind auch Meta-Daten über Abbilder von virtuellen Maschinen (VM) in den Sammlungen der Compute Service-Anbieter spärlich.

Die dargelegte Arbeit stellt der Forschergemeinschaft ein Rahmenwerk mit Bewertungs- und Vergleichsmethoden für Compute Services bereit. Die Methoden erlauben Compute Cloud-Konsumenten Compute Service hinsichtlich verschiedener Charakteristiken zu bewerten und zu vergleichen. Da die Methoden Cloud-Konsumenten dienen, basiert das generelle Vorgehen auf der Nutzung von Client-seitigen Remote-Verbindungen zu virtuellen Maschinen, um Meta-Daten zur Laufzeit zu gewinnen. Hierfür wird ein archetypisches Automatisierungsmodell bereitgestellt. Die zur Laufzeit extrahierten Informationen können Compute Services anhand eines generischen und erweiterbaren Meta-Datenmodells zugeordnet werden. Außerdem wird durch das Rahmenwerk basierend auf Ansätzen der multi-attributiven Entscheidungsfindung (MADM) eine Methode eingeführt, die es Konsumenten ermöglicht Compute Services hinsichtlich mehrerer Charakteristiken anhand eines Scores zu vergleichen. Weiter kann ein Stoppregelansatz Kostenbudgets während sukzessiver Compute Service-Bewertungen durchsetzen. Die Regel schließt bei der Suche nach einem Compute Service mit höchstem Score anhand von vorab verfügbaren Meta-Daten voraussichtlich schlecht abschneidende Services aus.

Das Rahmenwerk wird in zwei spezifischen Instanzen eingesetzt, um Compute Services bezüglich ihrer Leistung und Inhalten von VM-Abbildern zu bewerten. Eine Instanz des Rahmenwerks erlaubt Bewertungen von Compute Services anhand derer Leistung. In diesem Zuge wird eine Methode präsentiert, die eine auf dem Automatisierungsmodell des Rahmenwerks basierenden Prozedur einsetzt, um Compute Services mit Benchmarking-Skripten zu messen, die über Remote-Schnittstellen von der Clientseite ausgeführt werden. Neben der Prozedur bietet die Methode eine Definitionssprache für Stapelverarbeitung sowie die Wiederholung und Terminierung von Leistungsmessungen. Ergebnisse aus Messungen können durch eine Erweiterung des Meta-Datenmodells aus dem Framework festgehalten werden. Zum Vergleich der Ergebnisse, wird die MADM-basierte Scoring-Methode des Frameworks adaptiert. Die angepasste Scoring-Methode erlaubt Compute Cloud-Konsumenten Vergleiche anhand eines Scores aus definierten Leistungsattributen. Zur Limitierung der durch das Benchmarking von verschiedenen Compute Services entstehenden Kosten wird die Stoppregel des Rahmenwerks in solcherart konfiguriert, dass vorab bekannte Hardwarekennzahlen einbezogen werden. In Hinblick auf von Compute Services angebotene VM-Abbilder zeigt eine Instanz des Rahmenwerks eine Prozedur zum Erfassen von VM-Inhalten von der Client-seite. Eine Erweiterung des Meta-Datenmodells des Rahmenwerks erlaubt das Festhalten der Ergebnisse.

Die Umsetzbarkeit der Methoden wird durch Softwareprototypen belegt, die in der dargelegten Arbeit vorgestellt werden. Die Prototypen dienen zudem als Grundlage, um die Methoden und Modell zu evaluieren. Eine Reihe von Experimenten zeigt die Validität der Ansätze. Weitere Untersuchungen adressieren die Promptheit der Bewertungsansätze, sowie die Komplexität der Berechnungen bei der Kombination mehrerer Meta-Datenattribute in einen Score. Darüber hinaus werden die Methoden mit dem Stand der Kunst verglichen, um Vor- und Nachteile herauszustellen.

Contents

1. Introduction 1

1.1. Motivation . . . 1

1.2. Comparative Assessments . . . 4

1.2.1. Performance of Compute Services . . . 4

1.2.2. Virtual Machine Images . . . 8

1.3. Research Questions . . . 10

1.4. Research Methodology . . . 12

1.5. Contributions . . . 15

1.6. Published Material . . . 17

1.7. Thesis Organization . . . 20

I. Foundations 21

2. Background 23

2.1. Cloud Compute Services . . . 23

2.1.1. Definition of Cloud Computing . . . 23

2.1.2. Virtualization . . . 27

2.1.3. Consumption of Cloud Compute Services . . . 28

2.2. Compute Service Meta-Data . . . 32

2.2.1. Performance of Compute Services . . . 32

2.2.2. VM Images . . . 39

2.3. Formal Decision-Making . . . 43

2.3.1. Multi-Criteria Decision-Making (MCDM) . . . 43

2.3.2. The Analytic Hierarchy Process (AHP) . . . 44

2.4. Theory of Optimal Stopping . . . 48

2.4.1. Stopping Problems . . . 48

2.4.2. Stopping Rules . . . 48

3. State of the Art and Related Work 53

3.1. Compute Service Meta-Data Management . . . 53

3.2. Comparative Assessments of Compute Services . . . 55

3.2.1. Assessments of Compute Services . . . 56

3.2.2. Comparisons of Compute Services . . . 57

3.3. Compute Service Performance Measurements and Comparisons 59
3.3.1. Compute Service Performance Benchmarking 59
3.3.2. Automation and Repetition of Performance Benchmarks 61
3.4. Virtual Machine Image Introspections and Comparisons 66
3.4.1. VM Image Meta-Data Attachment 66
3.4.2. Introspection of VM Image Meta-Data 67

II. Conceptual Framework 69

4. Framework for Comparative Assessments of Cloud Compute Services 71
4.1. Introduction & Overview 71
4.1.1. Relations Between Models & Methods 73
4.1.2. Extensibility of the Framework 74
4.2. Meta-Data Model 75
4.3. Run-time Assessment Automation Model 76
4.4. Scoring Method 78
4.4.1. MADM for Value Aggregation 78
4.4.2. Evaluations with the $(MC^2)^2$ 79
4.4.3. Local Scoring 86
4.4.4. Global Scoring 87
4.5. Cost Budgeting Stopping Rule 89
4.5.1. Cost Calculation 89
4.5.2. Compute Service Scores & Ordering 90
4.5.3. Stopping Rules 93
4.6. Software Prototype implementing the Scoring Method 98
4.7. Evaluation of the MADM-based Scoring Method 102
4.7.1. Case Study with the Industry 103
4.7.2. Computational Complexity 105

III. Instantiation 113

5. Assessments of Compute Service Performance at Run-time 115
5.1. Automated Performance Benchmarking of Compute Services 117
5.1.1. Automated Performance Benchmarking Procedure 118
5.1.2. Repetitions and Benchmarking Result Aggregation 120
5.1.3. Data Model for Roadmap Scheduling and Measurement Results 124
5.1.4. Process Model for Using the Automated Benchmarking Procedure 126
5.1.5. Costs, Timeliness & Target Audience 129

5.2. Compute Service Performance Scoring 131
5.2.1. Scoring for Performance Meta-Data 131
5.2.2. Aggregation Scheme Requirements 131
5.2.3. Weighted Compute Service Performance Scores 132
5.3. Performance Benchmarking with the Cost Budgeting Stopping Rule . . 135
5.3.1. Cost Calculation 135
5.3.2. Performance Scores as Observation of Compute Services 136
5.3.3. Compute Service Order and Prior Information 136
5.3.4. Configuration of the Stopping Rule 137
5.4. Software Prototype for Automated Compute Service Performance Benchmarking with Stopping Rule 138
5.4.1. LibIntelliCloudBench Library 138
5.4.2. Web Application 140
5.5. Evaluation of the Automated Performance Benchmarking Method . . . 142
5.5.1. Validation of the Automated Performance Benchmarking Approach 142
5.5.2. Comparison to the State of the Art 145
5.5.3. Results of the Promptitude Measurements 146
5.5.4. Statistics for AWS EC2 Compute Services 148
5.6. Evaluation of the Performance Benchmarking Stopping Rule 151
5.6.1. Validation of the Performance Benchmarking Stopping Rule . . 151
5.6.2. Comparison to Standard Stopping Rules 153

6. Assessments of Virtual Machine Image Contents at Run-time 159
6.1. Virtual Machine Image Meta-Data Model 160
6.2. Introspection of Virtual Machine Images 162
6.2.1. Automated VM Image Introspection Procedure 162
6.2.2. Discoverer Component 163
6.2.3. Crawler Component 164
6.3. Search and Comparison of Virtual Machine Images 167
6.4. Software Prototype for Virtual Machine Image Introspection 168
6.4.1. Crawler Script 168
6.4.2. Web Application 169
6.5. Evaluation of the VM Image Introspection Method 170
6.5.1. Validation of the VM Image Introspection Approach 170
6.5.2. Results of the Promptitude Measurements 171
6.5.3. Statistics for AWS EC2 Compute Services 172

IV. Finale 175

7. Conclusions, Discussion & Outlook 177
7.1. Conclusions 177

7.2. Discussion & Outlook 181

Appendix 185

A. Performance Meta-Data on Compute Service Websites 187

B. Implementation Details & Code Examples 189
B.1. Example of AotearoaLib Programming Interface 189
B.2. Excluded Micro-benchmarks in LibIntelliCloudBench 191

C. Original Interview Form for a MADM Software Prototype 193

1. Introduction

This chapter defines the context of the research presented in this work. After a motivation, the problems of assessing compute services in regards of performance and VM images are explained. The chapter continues with the definition of research questions and the followed research methodology. Then, the contributions provided by this work are presented. Finally, the chapter discusses published material and ends with an explanation of how this thesis is organized.

1.1. Motivation

Organizations have widely adopted Information Technology (IT) to support and automate business operations [1]. Some organizations even base whole business models on IT systems, profiting from saving cost over manual work [2–4]. Nevertheless, running an own infrastructure to operate IT systems generates costs for organizations. From the capital expenditures (CAPEX) for real estates, hardware and software to operational expenditures (OPEX) for electricity and human resources, various expenditures aggregate into the total costs of a company's own IT infrastructure [5].

Unlike own IT infrastructures, cloud computing abandons large upfront investments and allows IT infrastructures to be outsourced and consumed as cloud infrastructure services in virtually infinite amounts [6, 7]. There are various cloud infrastructure services to build an IT infrastructure, of which *compute services* are very prominent to serve for computational tasks [8].

When leasing a compute service, a consumer is granted remote access to computer hardware resembled by a Virtual Machine (VM) for the duration of the leasing. With multi-cloud libraries like jClouds [9] multiple compute services can be accessed in a similar manner.

While multi-cloud libraries facilitate consumers with access to various compute services, the differences between services are manifold. Compute services are distinguishable in numerous aspects, such as pricing, contract duration, or service quality [10]. Such characteristics can be expressed as data attributes which are attached or linked to a compute service. Data attached to and describing a compute service is referred to as meta-data.

The notion of *meta-data* is to describe characteristics and properties of an entity. The preceding term *meta* originates from ancient greek and means "add" or "with" which refers to the concepts of "in addition" or "linked to". Meta can, however, also refer to an abstract or higher level view of a concept - also called a meta-level. In regards of meta-data, the prefix defines particular data linked to an entity to describe its - higher level - attributes.

In case of compute services, such high level attributes describe characteristics of the service to make it comparable with competing services. To assess *meta-data* for gaining insights and for comparing compute services, consumers may access provider websites, read contracts, inquire other consumers, or observe a service in action.

There are numerous aspects to consider in an examination and comparison of compute services. In this regard, providers and other sources typically lack in providing consumers with detailed and reliable meta-data concerning the various aspects (c.f. section 2.2 and chapter 3) [10–12]. Therefore, a consumer is obliged to assess additional meta-data about compute services by own means. As this work gives consumers a novel approach for single-handed assessments of compute services, the framework presented in this work is employed to assess compute services in regards of two particular aspects: (1) *performance*, and (2) *VM images*.

To allow a comparison of compute services regarding *performance* meta-data, a consumer is confronted with a non-trivial measurement and evaluation task. Heterogeneous hardware and virtualization software used in compute services introduce variations in performance outcomes [13–16]. In order to acquire reliable data, measurement methods and metrics that can quantify the performance to be expected from a compute service are needed. The act of measuring a compute service's performance, referred to as benchmarking, implies the execution of computational tasks on a compute service VM at run-time. The sheer number of necessary measurement repetitions, due to performance variations in conjunction with observing multiple compute services, calls for an automation of the measurement process.

Even with suitable performance metrics and measurement methods, a comparison of compute services remains multi-dimensional. To enable a simple, one-dimensional comparison of compute services, an aggregated score must be obtained. Hence, a method to combine multiple performance attributes into a one-dimensional, representative total value of a compute service is required. Generally, multi-attribute decision-making is capable of mapping compute services with multiple performance attributes to a single comparable score (on a ratio scale) [17–19]. Yet, an adoption to support compute service consumers with applying these methods for performance attributes of compute services is needed.

Aside from performance meta-data of compute services, the use of compute services involves the assembly of a software stack from a *VM image* which is managed and published by the provider [20–22]. Using VM images available from repositories, consumers can build a custom software stack. While simple VM images ask consumers to start with an operating system only, comprehensive repositories offer a wide range of VM images with whole pre-configured software stacks. Thereby, every consumer can profit from the expertise captured in prepared VM images, or rely on his own experience and build a custom software stack on a simple VM image.

Nonetheless, to capture the diversity in VM image repositories, and to distinguish and understand the contents of VM images, compute services fail to provide sufficient meta-data (see chapter 3) [11, 12]. A plausible approach to assess meta-data about VM image contents is to introspect contents at run-time [23, 24].

Available meta-data about compute services is scarce and not necessarily reliable. Therefore, approaches that empower consumers in their undertaking to assess compute services single-handedly is imminent. This work introduces a framework which gives consumers means to assess and compare compute services based on meta-data gained at run-time.

In particular, an automation model is presented that describes how compute services can be assessed in an automated manner through remote interfaces. Furthermore, the present work includes a method to compare compute services regarding meta-data attributes using a MADM method. In addition, means to approach a mitigation of leasing costs in an assessment of multiple compute service are developed by using a stopping rule.

The generic concepts are employed in two particular instantiations of the framework in order to assess specific characteristics of compute services. Specifically, instantiations to assess compute services in regards of performance and VM image contents are established in this work.

1.2. Comparative Assessments

The first section has argued about the amount of and the absence of reliable meta-data about compute service. Consequently, the necessity of methods to assess and compare compute services in terms of meta-data has been identified. In this section, the intricacies of assessing and comparing compute services are discussed exemplarily for two aspects: (1) performance, and (2) VM images. These two aspects are reflected in the research questions and later in this work treated with instantiations of the proposed assessment framework.

The following subsection will elaborate on the problems occurring in comparative assessments of performance and VM image contents in more detail. First, the intricacies of comparative assessments of compute services in terms of performance are explored. Finally, the tasks to be solved to capture contents of VM images in meta-data are discussed.

1.2.1. Performance of Compute Services

Compute service consumers can choose from a range of offerings available on the market. As a result, compute services compete for consumers with price differentiation and by offering various feature sets [25]. Due to compute services' purpose to serve for computational tasks, compute service consumers typically consider the factors price, available performance attributes, and hardware figures, i.e., processor cores, and memory and hard disk space [10]. Commonly, consumers consider given hardware figures and resulting performance attributes in IT infrastructure capacity planning processes [26, 27].

In the following, a sample of cases that benefit from detailed compute service performance meta-data can be found:

- Capacity planning which incorporates testing and predicting a compute service's performance in terms of system requirements.
- Comparison of compute services according to multiple performance attributes.
- Monitoring of a compute service's performance attributes over time to detect service quality variations.

Nonetheless, providers regularly only describe hardware figures included in compute services, but do not provide comparable performance attributes [10]. Since hardware is heterogeneous between cloud providers and even within data centers, the actual performance of a compute service is not coherent with its hardware figures [13–16]. For example, compute services with 1 CPU core of competing providers may not show an identical processing time for a certain computational task. Hence, methods to assess performance metrics at run-time are necessary to determine a comparable performance value.

Moreover, performance measurements of compute services may show variations over time. A computational task, hence, requires varying processing times when executed at different points in time [28]. Particularly, compute services have been found to show variations as hardware is frequently shared between rented systems [29, 30]. Thus, performance attributes must signify variations or include attributes with variation metrics.

To gain detailed performance meta-data, the field of system performance analysis has introduced performance benchmarking methods to measure performance attributes employing metrics with fixed benchmark scales [26, 31]. Existing work has succeeded in applying standard performance benchmarks to compute services [32–34]. Furthermore, existing approaches incorporate an identification of variations through repetitions [13, 29, 35]. Figure 1.1 depicts a consumer's view and actions involved when measuring performance meta-data of multiple, competing compute services at run-time.

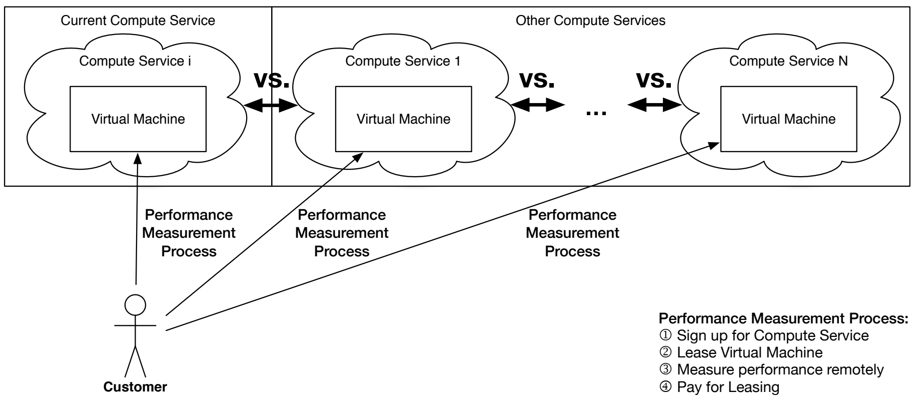


Figure 1.1.: A Consumer Measuring Compute Service Performance Meta-Data

By employing diverse performance benchmarks, a compute service can be measured in multiple dimensions [36, 37]. Nevertheless, determining the best service from comparing a mix of benchmarks is not trivial. Given performance attributes in diverse benchmarking metrics, a best service can easily be determined if it has the highest value for every attribute. Chances are, however, that different services have a highest value for certain attributes. Moreover, benchmarks can be of unequal importance to a consumer.

Thus, multi-dimensional benchmarking results should be represented in an aggregated and weighted score which considers a consumer's preferences. Existing work provides means to compare cloud providers and compute services according to a wide range of weighted attributes – e.g., security, availability or cost – using diverse methods from Multi-Attribute Decision-Making [38–40].

Specifically, methods like the Analytic Hierarchy Process are able to weight multiple attribute values and calculate an aggregated, one-dimensional utility value on a ratio scale [17, 41]. Although methods from Multi-Attribute Decision-Making [19] can weight and map multiple attributes of an alternative to a score, the attributes to consider have to be defined beforehand. Yet, a method to evaluate services according to diverse, weighted performance attributes is absent.

Moreover, employing performance benchmarks requires to lease compute services for the period of a measurement at run-time (see figure 1.1), thereby generating costs [42–44]. When observing multiple compute services, the total costs are the sum of the leasing costs for all services for the duration to conduct measurements.

For services and products in electronic markets, costs to assess information are commonly reduced by platforms of intermediary third-parties, i.e., information providers or brokers [45]. Such third-party information platforms allow to split costs between consumers and persist collected information [46]. However, there could be a set of compute service consumers that requires very specific performance meta-data that may not be available from third-party platforms. For example, a consumer might desire to get hold of specific floating point benchmarking results for a certain set of compute services to evaluate the suitability for a software that involves intensive floating point computations. Hence, a development of tools to assess performance of compute services according to custom benchmarks is imminent.

Alternatively, costs can be reduced by only assessing information for a small, relevant subset of compute services [47–49]. While smaller sets of compute services mean less costs, an intricate trade-off must be made to avoid sacrificing too many measurement results and, thereby, fidelity by the virtue of ignored services and saved costs.

Stopping rules developed in the field of stopping problems are capable of making trade-offs between the continuation and ending of a process according to a stopping rule. To apply a stopping rule approach, a specific rule must be found that expresses the trade-off to be made after each step of a process. For performance measurements, a stopping rule is needed which evaluates the trade-off between result fidelity and costs of continuing the measurements. Existing meta-data about compute services, such as hardware figures, can be an indicator of which performance can be expected in pending measurements.

In conclusion, due to a lack of meta-data, making compute services comparable according to performance attributes requires run-time measurements. For measuring multiple performance attributes of compute services, available standard benchmarks can be applied. Besides, approaches exist which help to observe variations through repeated measurements. However, means to aggregate measurement results into a single score while considering a consumer's preferences respecting benchmarks must be developed. Thereby, consumers are facilitated with means to compare services according to multiple attributes. In addition, in a performance benchmarking process, consumers are able to trade expenditures for the lease of compute services with the fidelity of the results by adjusting the number of compute services considered. Since the trade-off is not trivial, consumers need support in determining when to stop driving further measurements.

1.2.2. Virtual Machine Images

Aside from the performance attributes of a compute service, also the array of VM images available for a compute service are of interest to a consumer. Compute service consumers are served with *Virtual Machines* resembling computers with hardware resources, i.e., processors, memory, hard disks, and network interfaces. By using *virtualization* technology, multiple VMs can be run on computer hardware using a Virtual Machine Monitor (VMM), also referred to as a *hypervisor* [6, 50]. A VM is persisted as a *VM image* that contains a copy of the (virtual) hard disk and, optionally, a hardware configuration, e.g., including the number of CPU cores and amount of memory demanded by the VM. Using a VM image, multiple VMs can be instantiated with specific hardware configurations and access to copies of the virtual hard disk, hence, with the same data and software available.

The amount of data and pre-configured software on a VM image's hard disk varies depending on what a VM is needed for. For example, a VM image may contain only an operating system if VMs from the image shall be very customizable or only need the operating system as a basis. In contrast, a VM image that contains an operating system and a pre-configured software stack gives a basis to deploy software with few efforts for installation and configuration procedures. The latter, more comprehensive VM image category is referred to as *virtual appliance*.

In compute services, VM images are offered and maintained in repositories. VM images in a repository comply with a certain format and are not necessarily compatible with more than a single compute service. Depending on the repository policy pursued by a compute service provider, a repository is either open to the public or is accessible to consumers only. Furthermore, actors adding and maintaining VM images in the repository can be any consumer or the provider only.

The contents of VM images and, in particular, of cloud appliances created from other consumers is typically not included in existing meta-data, nor can it be assessed from the VM image as a file alone. Instead, a consumer needs to instantiate and inspect a VM image at run-time to discover its contents, such as contained data and software like the operating system or installed system libraries. Potentially, the use of VM introspection allows for automated access to a VM image's content at run-time [23, 24]. In addition, the field of Operating System Configuration Management (OSCM) is promising in terms of its capabilities to detect installed software packages and libraries automatically when access is granted to a running VM [51, 52].

In the following, a list of example use cases is provided that illustrates how a compute service consumer benefits from VM image meta-data:

- Search for a VM image with certain software and libraries from the repository while benefiting from a provider's or a community's experience.

- Highlighting differences between similar VM images and versions of one VM image.
- Documentation of a VM image to rebuild an image with a similar software configuration at a different compute service.

Nevertheless, repositories of VM images can comprise large numbers entries[1] rendering a manual introspection of all images unfeasible due to time and cost constraints. Besides, virtual appliances may contain a plethora of system libraries and software packages and, thus, demand a long time for an introspection. Therefore, an automation of the introspection process is necessary to free consumers from manual introspection efforts.

In conclusion, to gain insights about available VM images and, in particular, virtual appliances, contents could be introspected at run-time. To document the available contents, the field of configuration management offers promising approaches to detect and document installed software packages. However, depending on the amount of software packages in a VM image a consumer faces tremendous manual efforts in an introspection. Also, the sheer number of available VM images indicates that consumers may want to repeat the introspection process for various images. Altogether, the expected effort of VM image introspection for various compute services and repositories demands an automation of the process.

[1]Amazon EC2 comprises over 24.000 images in the US-EAST-1 region on 7th January 2014

1.3. Research Questions

This thesis researches the problem of assessing compute services to gain meta-data, particularly with relation to performance attributes and VM image contents. The problem leads to the principal question that asks for methods or procedures to gain and interpret meta-data on performance attributes and VM image contents as a consumer.

The principal research question to be addressed by this work is:

> *How can cloud compute services be assessed*
> *at run-time regarding meta-data?*

Since answering the principal research question is not trivial, the question is subdivided into multiple more specific research questions RQ1-RQ3. Implicitly, this also allows for a detailed evaluation of the principal research question through its sub-questions.

The research questions posed in this work shall be divided according to the two meta-data aspects of compute services: performance and VM image contents. In a first step, the following question addressing performance meta-data shall be researched:

> *RQ1. How can multiple compute services be made comparable regarding meta-data?*

For comparing compute services regarding performance meta-data as requested by RQ1, approaches to measure performance are required. To gain a single-dimensional metric that reflects the performance value of a compute service in regards of a consumer's preferences, MADM seems a suitable basis to calculate a weighted performance value. The following sub-questions shall be answered in this research work:

> *RQ1a. How can compute services be assessed at run-time in an automated manner?*
> *RQ1b. Can MADM support the aggregation of multiple meta-data attributes into a subjective value for a compute service?*
> *RQ1c. How is the computational complexity of using MADM to compare compute services regarding meta-data?*

Given that compute services can be measured and compared regarding meta-data, approaches to reduce costs during assessments shall be researched. Run-time assessments need access to a VM, the system under test [31], and entail leasing costs from compute services. This research focuses on a budgeting of the costs occurring from run-time assessments, leading to the following research question:

RQ2. How can costs for run-time assessments of multiple compute services be budgeted?

Apart from research questions of general nature regarding run-time assessments of meta-data, this work explores two particular characteristics of compute services, i.e. performance and VM image contents. Both characteristics provide a task to instantiate and evaluate the framework introduced by this work. The instantiations of the framework furthermore represent units to examine the research questions in experiments and practical scenarios.

As one particular compute service characteristic, the present work researches the run-time assessment of compute service performance. Within the scope of compute service performance, results of measurements should be captured as meta-data. Besides, an automation of the performance measurement process needs to be explored. Therefore, the following research question shall guide the exploration of assessments of compute services in regards of performance:

RQ3. How can performance be assessed at run-time from multiple compute services in an automated manner?

The present work also explores the assessment of VM image meta-data. The possibility to document contents of a VM image as meta-data is of distinguished concern. The feasibility of automation regarding an assessment of VM images at run-time are to be researched in this work. Consequently, the following research question shall be explored:

RQ4. How can Virtual Machine image contents be assessed at run-time from multiple compute services in an automated manner?

1.4. Research Methodology

The research conducted in this work follows a high-level process of three major steps: observation, development, and evaluation. Initially, in an observation phase, experiments with cloud compute services and a study of literature have led to the motivation of this work. After a clear understanding of the problem has been developed, research questions and a research methodology could be defined. With this in mind, models and methods were developed to address the questions. The development of models and methods was carried out in iterations of conceptual development, implementation, and test phases. Ultimately, final versions of the models and methods were validated and evaluated to test the results against the research questions. Figure 1.2 depicts the process followed in the research methodology that evolved over the course of this work.

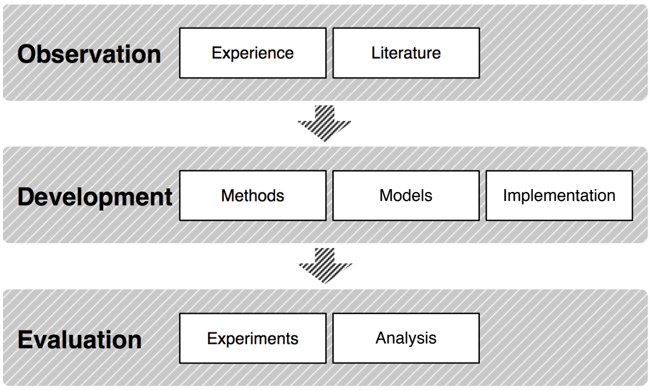


Figure 1.2.: Overview of Research Process

To test the validity and quality of the approaches proposed in this work, an analytical evaluation and experiments with the help of instantiations as software prototypes are conducted. For the validation and evaluation of the proposed methods and their instantiations, research strategies from the field of software engineering research are applied [53]. In addition, the quality of proposed methods is evaluated in analytical examinations.

In software engineering research, a specific set of research strategies exists [53]. Each strategy is defined by three components: (1) research question type, (2) research result type, and (3) validation type. A research question asks either for a method (for development or analysis), an analysis, a generalization, or a feasibility test. The research result corresponding to a question represents a procedure, a model, notation or tool, specific solution, judgement, or a report. And the validation may be manifested with an analysis, through experiences, in examples, or in evaluations.

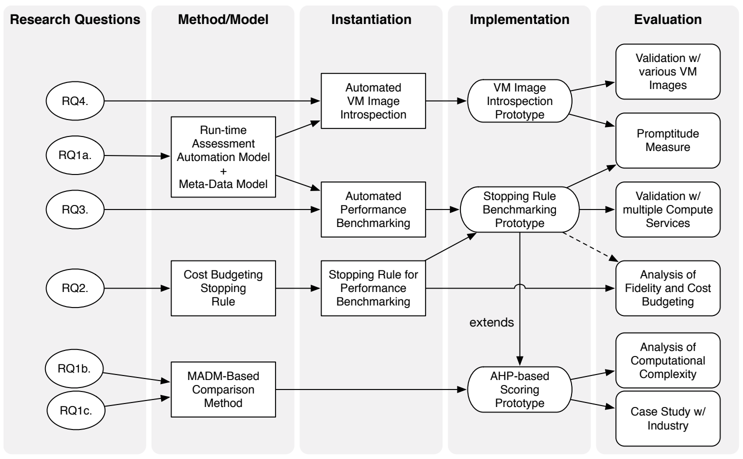

Research Questions
Method/Model
Instantiation
Implementation
Evaluation
RQ4.
RQ1a.
RQ3.
RQ2.
RQ1b.
RQ1c.
Run-time Assessment Automation Model + Meta-Data Model
Cost Budgeting Stopping Rule
MADM-Based Comparison Method
Automated VM Image Introspection
Automated Performance Benchmarking
Stopping Rule for Performance Benchmarking
VM Image Introspection Prototype
Stopping Rule Benchmarking Prototype
extends
AHP-based Scoring Prototype
Validation w/ various VM Images
Promptitude Measure
Validation w/ multiple Compute Services
Analysis of Fidelity and Cost Budgeting
Analysis of Computational Complexity
Case Study w/ Industry

Figure 1.3.: Overview of Research Paths

This work follows diverse paths to validate the presented methods in the light of the research questions. Figure 1.3 illustrates an overview of the research paths taken to answer all research questions. On each path, a research question leads to a validation or evaluation of a proposed method.

In terms of software engineering research strategies, the research questions RQ1 and RQ2 point at a method for the analysis of compute service performance meta-data. Besides, RQ3 and RQ4 can be interpreted to demand a method of analysis concerning with the documentation of performance and VM image meta-data. All of the presented conceptual methods are implemented directly or indirectly as software prototypes and validated in examples and partly in case studies, some of which were conducted with the industry (c.f. figure 1.3). While the *MADM-based comparison method* is implemented directly as a software prototype, the *run-time assessment automation model*, the *meta-data model*, and *cost budgeting stopping rule* are indirectly implemented as software prototypes for concrete instantiations of the methods. The instantiations for the *run-time assessment model* are using the model as a blueprint to assess compute services in terms of performance and VM image contents. The software prototypes *stopping rule benchmarking prototype* and *VM introspection prototype* are an implementations of the instantiations *automated performance benchmarking* and *automated VM image introspection* respectively.

Figure 1.3 also highlights relations between the prototypes. While the implementation of the subjective evaluation method in an *AHP-based scoring prototype* is evaluated in a case study, both other implementations are validated by example. Moreover, implementations are used to explore practical qualities in the application of the methods. In particular, the actual computational complexity and promptitude of the methods is tested with the implementations in empirical experiments. Experiments are conducted with all prototypes, namely the *stopping rule benchmarking prototype*, the *AHP-based scoring prototype*, and the *VM introspection prototype* (cf. figure 1.3).

1.5. Contributions

This work pursues the goal of giving means to assess cloud compute services at run-time to gain meta-data. In this regard, several contributions are made that support such assessments, also under a cost aspect, and enable a comparison of compute services. The contributions are two-fold in their kind: conceptual work and implemented prototypes. The following subsections first present the conceptual contributions and then the contributed software prototypes.

Contributed Conceptual Methods

In the following, a list of conceptual contributions to conduct assessments at run-time and compare compute services:

Meta-data model As part of the framework a meta-data model is introduced which is capable of attaching meta-data to compute services in a central, queriable database. The meta-data model is furthermore extended in two distinct instantiations to capture performance attributes of compute services and contents of VM images available in the repositories of a compute service. The extended meta-data model for performance attributes captures measured benchmarking results and variations from multiple iterations. Also, compute services in diverse hardware configurations can be reflected in the model. The extended meta-data model to capture VM image contents provides a generalization of information available from diverse software package managers that manage software configurations on various operating systems.

Automation of compute service assessments The conceptual method presented in this work comprises an archetypical automation model to assess compute services in an automated manner. In particular, performance benchmarking driven with a procedure built upon the automation model is able to attain results repeatedly over time and in diverse frequencies. Besides, a second instantiation of the model allows consumers to introspect VM images of a compute service provider in an automated manner. During an introspection, meta-data about an image's contents, specifically about installed software and libraries, is extracted.

Calculation of a comparable compute service score This work introduces a method to aggregate meta-data attributes into a one-dimensional, weighted score that represents a compute service. An AHP-based evaluation normalizes and aggregates multiple meta-data attributes into a single value on an absolute scale. In an aggregation, the diverse considered attributes are weighted according to a consumer's preferences.

Stopping rule with cost budget Using the stopping rule, a consumer is enabled to set a cost budget on the leasing costs for assessments of compute services. The stopping rule calculates after each assessment the probability to still find a higher scoring compute service according to a AHP-based score.

Contributed Software Prototypes

The second kind of contributions contained in this work are software prototypes. All prototypes are instantiations of conceptional methods presented in this work. The following list summarizes the software prototypes contributed by this work:

Compute service comparison web application Implementation of a web application that facilitates compute service consumers with weighted, aggregated scores for compute services. The prototype supports diverse criteria catalogs as templates, particularly with performance attributes, but also catalogs with a wide range of criteria to compare compute services according to aspects like security, prices, support quality, etc.

Compute service benchmarking web application Implementation of a system to drive performance benchmarks on compute services. The system allows users to measure multiple compute services with various benchmarks and, optionally, employ a stopping rule to trade measurement costs with fidelity of benchmarking results. Benchmarking runs can be scheduled and, thereby, a repetition of measurements is possible.

Virtual Machine image crawler Implementation of a software prototype to describe and capture meta-data of Amazon machine images with Ubuntu Linux operating systems. In particular, the system focuses on documenting the software configurations of images and allows consumers to access a database of VM image meta-data via a web application.

1.6. Published Material

There is preliminary work that has been published in the research community through peer-reviewed papers at conferences and in journals. Following, a list of publications which incorporate related concepts referenced in this document:

[1] M. Menzel, M. Schönherr, J. Nimis, and S. Tai. “$(MC^2)^2$: A Generic Decision-Making Framework and its Application to Cloud Computing”. In: *Proceedings of the International Conference on Cloud Computing and Virtualization (CCV 2010)*. GSTF. Singapore: GSTF, 2010.

[2] M. Menzel, M. Schönherr, and S. Tai. “$(MC^2)^2$: Criteria, Requirements and a Software Prototype for Cloud Infrastructure Decisions”. In: *Software: Practice and Experience* (2011).

[3] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann. “What Are You Paying for? Performance Benchmarking for Infrastructure-as-a-Service Offerings”. In: *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*. IEEE. Washington, D.C., USA: IEEE, 2011, pp. 484–491.

[4] S. Haak and M. Menzel. “Autonomic Benchmarking for Cloud Infrastructures: An Economic Optimization Model”. In: *Proceedings of the 1st ACM/IEEE workshop on Autonomic computing in economics*. ACM. Karlsruhe, Germany: ACM, 2011, pp. 27–32.

[5] M. Menzel, M. Klems, H. A. Le, and S. Tai. “A Configuration Crawler for Virtual Appliances in Compute Clouds”. In: *Proceedings of the 1st International Conference on Cloud Engineering (IC2E)*. IEEE. San Francisco, USA: IEEE, 2013, pp. 201–209.

[6] M. Menzel and R. Ranjan. “CloudGenius: Decision Support for Web Server Cloud Migration”. In: *Proceedings of the 21st International Conference on World Wide Web*. ACM. Lyon, France: ACM, 2012, pp. 979–988.

[7] M. Menzel, R. Ranjan, L. Wang, S. U. Khan, and J. Chen. “CloudGenius: A Hybrid Decision Support Method for Automating the Migration of Web Application Clusters to Public Clouds”. In: *Computers, IEEE Transactions on* (2014).

Figure 1.4 sets all published material into context with the aspects of performance and VM image meta-data of the framework presented in part II. Parts of the publications build the foundation for compute service evaluations and decision-making. The publications [1] and [2] present a framework to build multi-criteria decision making methods. The $(MC^2)^2$ framework helps to define a scoring function applicable to compute services. With the scoring function, an aggregated, weighted value can be calculated given measured performance attributes of a compute service. The framework and scoring functions have already shown their applicability in strategic cloud infrastructure decisions [2].

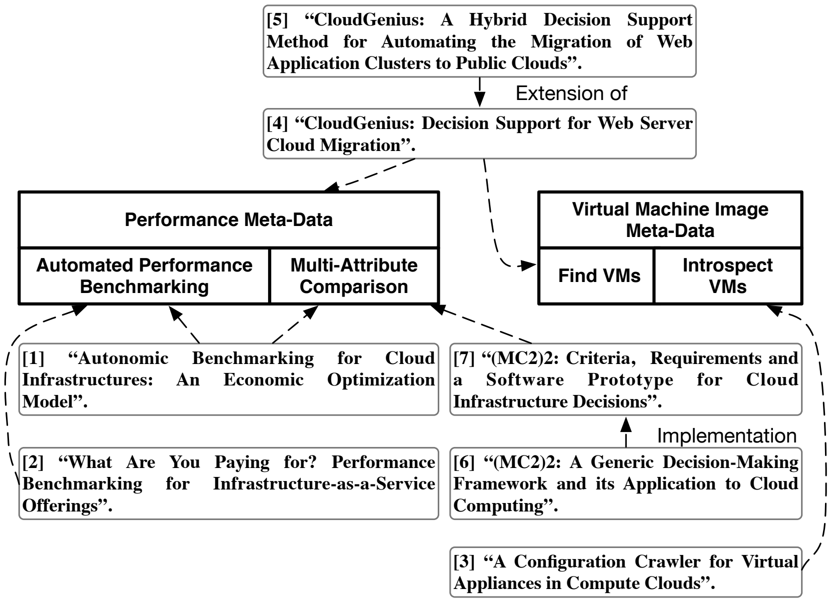

[5] "CloudGenius: A Hybrid Decision Support Method for Automating the Migration of Web Application Clusters to Public Clouds".
Extension of
[4] "CloudGenius: Decision Support for Web Server Cloud Migration".
Performance Meta-Data
Automated Performance Benchmarking
Multi-Attribute Comparison
Virtual Machine Image Meta-Data
Find VMs
Introspect VMs
[1] "Autonomic Benchmarking for Cloud Infrastructures: An Economic Optimization Model".
[7] "(MC2)2: Criteria, Requirements and a Software Prototype for Cloud Infrastructure Decisions".
Implementation
[2] "What Are You Paying for? Performance Benchmarking for Infrastructure-as-a-Service Offerings".
[6] "(MC2)2: A Generic Decision-Making Framework and its Application to Cloud Computing".
[3] "A Configuration Crawler for Virtual Appliances in Compute Clouds".

Figure 1.4.: Published Material Overview

Moreover, there is preliminary work regarding performance benchmarking of compute services. [3] analyses which standard benchmarks apply for compute services and establishes a novel measurement method. Unlike existing methods, benchmarking is conducted over a time span in diverse frequencies. Using the method, experiments have shown that the performance of a compute service can vary. A second finding of the experiments is that performance benchmarks of compute services are costly.

To counter costs, [4] proposes a stopping rule that aims at budgeting the costs and minimize the number of observed services when benchmarking their performance. The goal is to find a service with a best performance score within a given cost budget for benchmarking.

Apart from performance measurements, [5] describes a method to crawl software configurations of VM images. The method instantiates VM images of compute services, determines the operating system and collects the list of software and system libraries available.

[6] introduces CloudGenius, a cloud migration support framework which makes use of the previously mentioned assessment and scoring methods in the case of web server migrations. CloudGenius supports a selection of a compute service and VM image as target platform in a migration. The framework comprises the notion of scoring functions, a process model, a data model, and links to meta-data assessment methods. For scorings

of compute services, CloudGenius benefits from the $(MC^2)^2$ framework published in [1]. Besides, the work refers to [3, 5] for suitable methods to measure performance and introspect VM images.

The CloudGenius framework has been extended to give decision support in the realm of web application migrations to the public cloud [7]. The extended version incorporates a genetic algorithm implemented with the MapReduce programming model to evaluate large numbers of web server cluster setups on compute services using the $(MC^2)^2$ framework. The work also discusses the use of $(MC^2)^2$ evaluation functions for scoring purposes and fitness functions.

1.7. Thesis Organization

The rest of the thesis is organized as follows: foundations in part I facilitate the reader with definitions and background, and shed light on related work and the state of the art.

In part II the conceptual framework and methods contributed by this work are presented. Chapter 4 gives an overview of the framework and how methods relate to another. Furthermore, methods to assess compute services in an automated manner, capture metadata attributes and link the to compute services, compare compute services using a scoring, and a stopping rule are presented and explained in detail. An implementation of the scoring method is presented and provides the basis for an evaluation with the industry and in experiments in respect of computational complexity.

Part III presents instantiations of the framework's models and methods to assess particular characteristics of compute services. First, chapter 5 instantiates parts of the framework to benchmark compute services regarding performance in an automated manner. The instantiation also adapts the scoring method for compute service comparisons according to performance attributes. Furthermore, with the use of the framework's stopping rule a cost budget can be enforced in compute service benchmarking tasks. A software prototype asserts the feasibility of automated compute service performance benchmarking and the application of the stopping rule. Besides, several experiments are conducted to evaluate the compute service performance benchmarking method.

In chapter 6 the framework's models and methods are instantiated to assess compute services in regards of VM image contents. A method to introspect VM images regarding contents, such as software libraries, in an automated manner is presented. An implementation of the method shows the feasibility and is employed in experiments to evaluate the method.

Finally, part IV complete this document with a conclusion, discussion and outlook.

Part I.

Foundations

2. Background

Chapter 2 introduces the reader to basic concepts relevant throughout the present work. First, cloud computing and, in particular, cloud infrastructure services including compute services are introduced. Following, meta-data of compute services is explored. Then, methods of formal decision-making and comparisons are presented. Finally, the theory of optimal stopping is explained in detail.

2.1. Cloud Compute Services

Cloud computing provides new models to offer and consume hardware infrastructure and software. The following sections define the notion of cloud computing with a focus on compute services, a particular cloud infrastructure service. In this respect, virtualization technology is explained as it provides a basis to cloud compute services and affects the behavior of compute services. The final sections specifically shed light on the consumption and constituent parts of compute services.

2.1.1. Definition of Cloud Computing

A definition of cloud computing has been approached in many publications and by diverse authors, in research, industry, and standardization organizations [6, 7, 54, 55]. Amongst the multitude of definitions, two particular instances cover the essential factors this work is based on. The National Institute of Technology (NIST) has published a definition of cloud computing that spans multiple dimensions:

> 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. This cloud model is composed of five essential characteristics, three service models, and four deployment models. [7]

The definition of the NIST is coherent with the definition of Baun et al. [6] which channels important aspects and influences this work as well. In contrast to NIST, Baun et al. expects resources utilized in cloud computing to be based on *virtualization* (see section 2.1.2). Although using virtualization as a basis is not strictly common for all services [56], most infrastructure services are employing virtualization technology [57, 58].

> By using virtualized computing and storage resources and modern Web technologies, cloud computing provides scalable, network-centric, abstracted IT infrastructures, platforms, and applications as on-demand services. These services are billed on a usage basis. [6]

The NIST definition explicitly refers to a list of characteristics and models that describe the aspects of cloud computing in more detail. The definition of Baun et al. observes similar cloud characteristics and explains details in addition to the definition. In the light of both definitions, the subsequent subsections describe characteristics and models related to cloud computing.

Characteristics of Cloud Computing

The NIST definition of cloud computing incorporates five characteristics that describe the concept of cloud computing in more detail. The definition of the characteristics focuses on a description for the role of a consumer who actively interacts with cloud services and is interested in meta-data of cloud services. Consequently, the explanations of the characteristics in this work will adapt the role of consumers. In contrast, a customer who purchases cloud services is only involved in the business transactions to attain access to the services of a provider.

A consumer can provision resources from a cloud provider unilaterally, when needed and without further interaction (*on-demand self-service*). Furthermore, through *broad network access* capabilities become available over the network (with standard mechanisms) by using thin and thick client platforms. Cloud computing channels a multi-tenant model for pooled computing resources which leads to a concealing of the resources location (*resource pooling*). Besides, in cloud computing capabilities scale with demand, often automatically (*rapid elasticity*). For consumers, the elastic behavior memes an availability of unlimited resources. The usage of resources is monitored, controlled and can be reported. Hence, a metering of consumer's utilization of capabilities is available (*measured service*).

The list of advantages and disadvantages induced by cloud computing's characteristics is respectable. An implication caused by the fact that notion of cloud computing embraces measured services is a pay-per-use billing model realized by cloud providers [6].

In turn, a pay-per-use billing model allows companies to avoid high capital expenditures (CAPEX) that were occurring in a purchase of on-premise IT infrastructure. Thereby, CAPEX are transmuted into operational expenditures (OPEX) which increase prallel to the company's growth. In addition, with economies of scale, the cloud promises to save costs and to be more reliable than typical on-premise infrastructures [55].

Service Models of Cloud Computing

The capabilities served by cloud providers following the cloud computing model can be divided into three service models. Commonly, offered cloud services can be divided into the three models: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [8]. While SaaS offers software and PaaS execution platforms as services, IaaS offers hardware infrastructures as service. Lenk et al. [8] developed a more precise representation of service models depicted as a cloud stack in figure 2.1 which includes a classification of a range of cloud services. The cloud stack representation also sorts the three models into layers since services can be based on layers beneath.

The cloud stack comprises the IaaS layer that utilizes physical hardware to provide infrastructure services, e.g., compute services, storage services and network services. Furthermore, the PaaS layer is built ontop the IaaS layer, and the SaaS layer is constituted from any of the aforementioned layers. Although, the stack implicates that services are built ontop of each other, this is not necessarily the case in every PaaS and SaaS implementation [56, 59, 60].

Deployment Models of Cloud Computing

There are multiple deployment models of cloud computing distinguishable by whom operates and consumes services. A private cloud is consumed by a single organization which may also be the owner and operator. Often, a third-party operates the private cloud which may also be located off-premise. A billing can be absent in private clouds while services are still measured and other characteristics remain valid. A federation of multiple private clouds is referred to as a community cloud.

The public cloud deployment model offers cloud services to the public. Commonly, consumers are billed by a company that operates and owns the cloud infrastructure. A mix of the private and hybrid cloud deployment model is referred to as hybrid cloud. A hybrid model results in a composition of public and private (or community) cloud services, allowing consumers to use public and private cloud services with the same technologies.

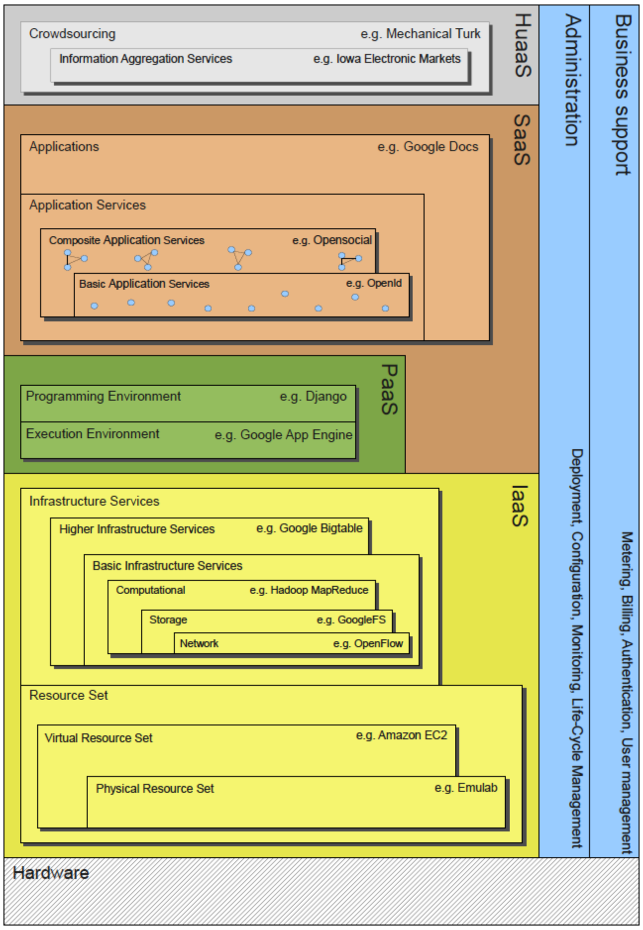


Figure 2.1.: Service Models and Layers in the Cloud Stack [8]

2.1.2. Virtualization

Virtualization technology is the basis of cloud infrastructure services and its notions of dividing hardware into virtual machines is a fundamental background for the present work. Particularly, the environment and effects created by virtualization software are considerable factors in the assessment of compute services, which is a cloud infrastructure service.

The notion of virtualization is to build an abstraction layer ontop of physical hardware. Through virtualization a creation of a virtual resemblance of a system, hardware or software, becomes possible. For cloud computing, and most of all for cloud infrastructure services, hardware virtualization is a fundamental prerequisite. A virtualized hardware resembles a set of hardware, most commonly a whole physical machine with its processors, memory, hard disks and network devices. Virtualized physical machines are refered to as Virtual Machines (VMs). A Virtual Machine Monitor (VMM) or hypervisor manages multiple VMs on a host system. VMs share resources of the host system and can be started or stopped by the hypervisor. The basis for a VM is a Virtual Machine (VM) image. By instantiating a VM image a new VM can be created or started. Images include a resource configuration that describes which hardware is required to instantiate a VM. Besides, a VM image contains initial data on the virtual hard disks and virtual memory.

How a hypervisor is installed on a host system depends on the hypervisor implementation. A hypervisor might be installed with it's own operating system on the bare hardware or on a pre-installed operating system. Multiple vendors offer hypervisor products, e.g., VMware vSphere [61], Citrix Xen Hypervisor [50], Red Hat Entperise Virtualization [62]. Figure 2.2 depicts the structure of a Xen hypervisor example setup.

As shown in figure 2.2, with virtualization software hardware is shared between many tenants. The use of the same hardware resources by multiple tenants leads to a potentially more economical utilization of the physical system. Traditionally, a single tenant uses a hardware resources constantly to a small extent or only occasionally to a full extent depending on the workload patterns. Then, the hardware is under-utilized, at least at times. By sharing the hardware resources, the costs to operate a physical system can be split between the tenants.

Nevertheless, virtualization comes with several side-effects which potentially affect the performance of VM provisioned by a compute service [30]. A more detailed discussion of such effects is conducted in section 2.2.1 by exploring the consumer-view of compute service performance.

Cloud computing and in particular cloud infrastructure services became possible when all requirements were fulfilled in order to attain a full automation of software deployments in large scales. Virtualization technology closed the gap that hindered hardware resources to be managed via remote interfaces, instantly and fully automated. Remote

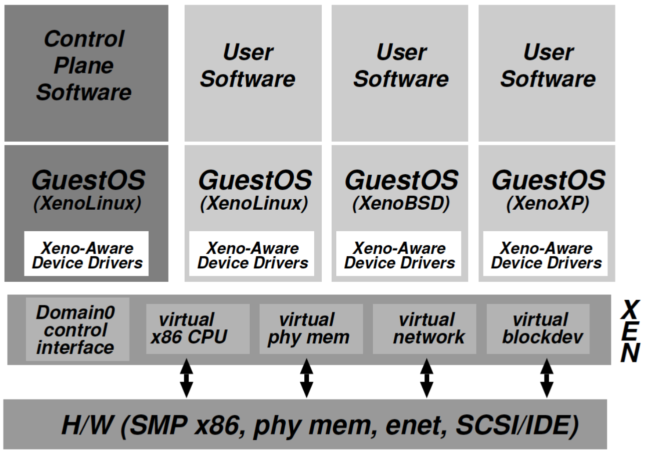


Figure 2.2.: Example Structure of a Xen Hypervisor [50]

interfaces of hypervisors allow to orchestrate large amounts of virtual machines in large data centers. The wrapping of these management calls in web service interfaces, offered to consumers and billed per use, completes the concept of cloud compute services and of most cloud infrastructure service.

2.1.3. Consumption of Cloud Compute Services

On the IaaS level, compute services provide virtual computer hardware with processors, memory and hard disks operated in a data center and make them available remotely to circle of consumers, depending on the deployment model. For realization of a compute service, typically, virtualization technologies are employed to divide physical machines into Virtual Machine segments and to homogenize varying physical hardware [55, 63]. Prominent compute services are Amazon Web Services Elastic Compute Cloud [57], Rackspace Public Cloud [58], and GoGrid Cloud Servers [64].

Consumers of a compute service can instantiate VM images that package at least a basic operating system or can even encapsulate a whole software stack. Virtual Machine (VM) images available for a compute service differ in the included operating system and software packages (see section 2.1.3). The typical process of deployment, assumed an account with a compute service provider is available, follows a three-step process:

1. Choose a VM image
2. Choose a compute service hardware configuration
3. Create one or multiple VM instances from the VM image

Steps (1) and (2) do not have a particular order and step 3 can appear in arbitrary frequency, e.g., triggered by an automatic scaling logic, depending in which context the compute service is used. As a result of the process a consumer receives access to the VM instances via remote interfaces like Secure Shell (SSH) remote terminals.

Virtual Machine Images in Public Compute Services

VM images are an essential part of compute services as every instantiation of a VM always involves an assignment of an image. To know about VM image contents is fundamental for consumers to successfully use compute services. VM images differ in contained operating system and software installed. Depending on the software contained a VM image, a consumer must install missing software or remove unnecessary items.

To understand which contents of VM images can and should be assessed, it is necessary to observe the structure and availability of VM images in compute services. Generally, the provisioning of VM images, that contain software and libraries, varies between compute clouds. Repositories differ regarding the dimensions of openness and VM image complexity. VM image complexity ranges from simple images comprising only an operating system to complex full software stack images, called virtual or cloud appliances. A cloud appliance is a VM image that has been optimally configured for a particular purpose, e.g., a database server appliance or an application server appliance.

Specific images with different configurations and software versions can lead to immensely large repositories. However, images from one repository of a certain provider cannot necessarily be instantiated at a different provider. Although many providers base their compute service on the same virtualization technologies, interoperability is not yet granted.

The management of a repository lies typically with the cloud provider, but contributions of new images can either be made by the cloud provider only or an open community. Figure 2.4 depicts an enhanced gantt chart of VM image repository contents and a classification of three compute cloud providers, namely GoGrid, Rackspace, and Amazon. In general, three types of repositories are distinguishable: (1) centralized packaging and management of bare operating system images (e.g., Rackspace [58]), (2) centralized packaging of a wide range of VM images, from bare operating system to complex software stacks (e.g., GoGrid [64]), and (3) decentralized packaging that involves users and communities (e.g., Amazon EC2 [57]).

VM image repositories of providers not only contain freely available images. Commercial software vendors, including IBM, SAP, and Oracle, package VM images and either release them for free or with commercial licenses. For example, Amazon's Application Programming Interface (API) comprises fee and license payment services for VMs

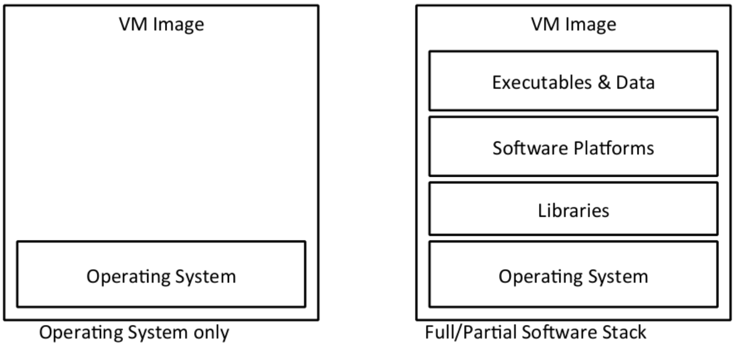


Figure 2.3.: Software on VM Images and Appliances

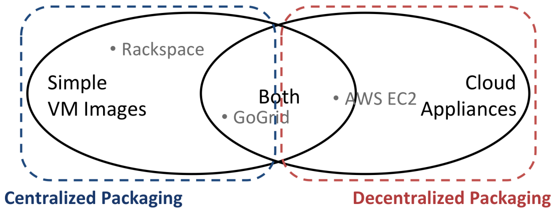


Figure 2.4.: Overview of VM Image Repository Contents

which are charged by hour or paid on a subscription basis. In parallel, commercial VM images of Amazon's repository are published on Amazon's AWS Marketplace [65].

Apart from cloud provider's API access to VM images, community websites publish lists of images. The images are often created by the community and available for multiple public cloud services. For example, the Cloud Market [66] is a VM image aggregator that holds and maintains a database of 85,365[1] VM images compatible with Amazon's EC2 compute service. Similarly, bitnami [67] provides a wide range of prepackaged software on VM images as an intermediary seller.

Compute Service Configurations

Compute service offerings differ in various attributes. Besides, service providers commonly offer multiple compute services differing in attributes of the Virtual Machine and service levels. A compute service is offered in a configuration with a certain amount of hardware resources, e.g., CPU cores, amount of memory, and hard disk space, for a price per time unit – typically full hours [68–71].

[1]checked 8th Sept. 2013

The pricing for compute service configurations differ. Configurations made available to consumers are predetermined in offered service variants by providers. In contrast, there are also providers which allow consumers to configure the hardware resources of a compute service. Depending on the pricing model, one time fees and other additional costs may appear.

Compute services of one provider are distinguishable by hardware configuration, coined as compute service type, since only the amount of hardware resources changes. Providers, including Amazon [72] and Rackspace [58], offer compute services in predefined configurations distinguishable by multiple dimensions such as processor cores, amount of memory, and hard drive size. In contrast, some compute services, e.g., 1&1 Dynamic Cloud Servers [73], facilitate consumers with a custom hardware configuration along those multiple dimensions. Accuracy and variations choosable within the dimensions vary.

Detailed information about available predefined hardware configurations is commonly available together with price tables on the websites of providers. For configurable compute services, the amount of hardware resources can be defined upon instantiation of a VM. Depending on the compute service, a VM image may restrict the configuration of the hardware resources. For example, Amazon EC2 offers VM images that only work with 64-bit and 32-bit processors.

Compute Service Access

How compute services are offered is similar for most compute service providers. Compute services can typically be accessed via web frontends and web service interfaces, using a web browser or client software handed out by the provider respectively. Using the interfaces, a consumer can start, configure and stop VM instances or save an instance as a VM image, depending on the capabilities of the service.

Furthermore, since VMs are potentially accessible from anybody via the internet, consumers typically need credentials to access a running VM. For example, when a provider uses the SSH protocol for remote connections to VMs a username and password or a SSH certificate is required. Alternatively, some providers employ a Virtual Private Network (VPN) for security and establish remote connections to VMs via remote console software. Figure 2.5 gives an overview of the diverse access channels for using the service or accessing VMs.

The jClouds java library [9] aggregates cloud service APIs of multiple providers into a uniform interface for consumers. Additionally, interface specifications like OCCI [74, 75] and CIMI [76] aim at fostering a unified access to compute services.

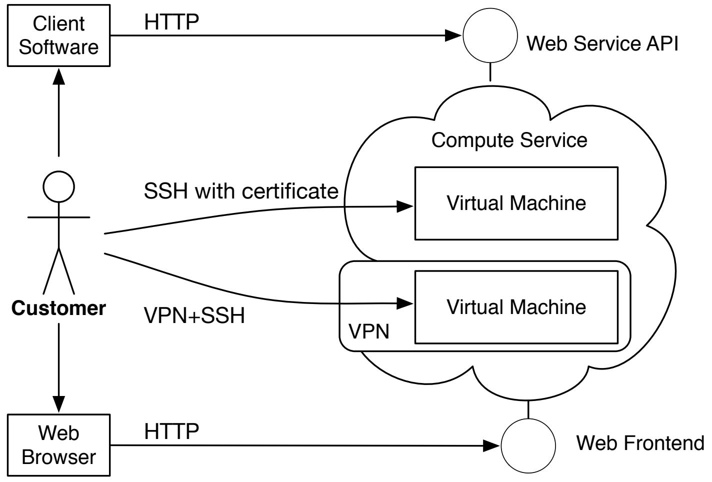


Figure 2.5.: Exemplary Access Channels to a Compute Service

2.2. Compute Service Meta-Data

Meta-data is essential to distinguish and compare compute services of one or multiple providers. This section explores the extent of meta-data available for compute services from providers and third-parties, particularly in respect of performance and VM image contents. Besides, generally available methods to assess compute services in regards of performance and software contents are presented.

2.2.1. Performance of Compute Services

Performance is a property of compute services that is of particular interest to consumers. The time to complete computational tasks depends on the performance utilizable from a service. Available meta-data indicates what performance can be expected of a compute service. To gain more details about a computer system's performance, benchmarking is an approach for further analyses. Since the performance may vary over time, repetitions give means to capture variations with metrics, such as statistical variance or standard deviation.

In the following, first the extent of available meta-data about compute services' performance characteristics is examined. Then, methods to assess a computer systems performance are presented and the general applicability to compute services is discussed.

Available Compute Service Performance Meta-Data

For compute services, available meta-data is scarce in a sense of making profound performance predictions. The information provided in public sources, such as from the APIs of compute service providers and on their websites, only gives an overview of the hardware dimensions. A summary of hardware and performance meta-data attributes published by AWS *EC2*, *Rackspace* Cloud Servers, and *GoGrid* Cloud Servers is given in table 2.1. The screenshots in figures A.1, A.2, A.3, and A.4 in chapter A of the appendix give an impression of how performance meta-data is published on provider websites.

Attribute	EC2	Rackspace	GoGrid
Amount CPU Cores	✓	✓	✓
CPU Performance Indicator	✓	×	×
CPU Architecture	✓	×	×
CPU Hardware	✓	×	×
Amount Memory	✓	✓	✓
Memory Performance Indicator	×	×	×
Disk Volumes	✓	✓	✓
Disk Performance Indicator	×	×	×
Network Bandwidth	×	✓	✓

Table 2.1.: Meta-data Attributes available on Provider Websites

Publicly available meta-data of compute services, whether hardware configuration or performance, is generally rather scarce [10]. Commonly, compute services indicate three hardware specifics: (1) number of processors (or cores) and their architecture, (2) amount of memory, (3) amount of hard disk space. While (2) and (3) are commonly described in bits or bytes metric, (1) is defined by a number and frequently with an architecture identifier, such as 32bit or 64bit. Figure 2.6 illustrates the structure of hardware configuration meta-data typically available for compute services.

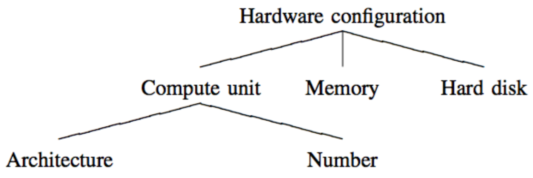


Figure 2.6.: Structure of Compute Service Hardware Configuration Meta-Data [10]

A further specification of the performance characteristics of the physical hardware (regarding (1)-(3)) is absent at the examined providers or, if available, diverse and rather vague. For example, the read or write performance of the memory is not published, and hard disk performance is clustered in non-SSD and SSD hard drives. Regarding processor performance, some providers state the clock speed precisely, others use processor cores or compute cycles to specify the performance. In parallel, some providers introduce own metrics to describe the processor performance. For instance, Amazon defines the metric Elastic Compute Unit (ECU) and Media Temple introduces a Grid Performance Unit (GPU).

Performance Analysis of Compute Services

The analysis of a system's performance is essential to better understand its quality. In the use, design and administration of a system, the quantification of performance attributes of a system according to metrics can be a fundamental factor. Considering performance attributes helps to select a system according to its performance, determine or modify its design or detect discrepancies in its behavior. The general goal is to maximize the performance of a system in a performance-cost trade-off.

To conduct a performance analysis, several performance evaluation strategies can be followed [31, 77]. Figure 2.7 depicts an overview of four classes of different strategies. Every strategy has its benefits and disadvantages. While using a model of the system is typically cheap to build (emulation and simulation), the real system provides results that reflect the actual performance behavior (in-situ and benchmarking). Also, exploring a system's performance behavior under real workloads is most accurate (in-situ and emulation) while modeled workloads give a general evaluation of a system (benchmarking and simulation) [31].

Compute service consumers are facing a specific situation in regards of performance analysis. A need to measure the actual performance of a compute service is due to system design tasks, but also in order to single-handedly monitor the quality provided by a compute service. However, in his task to measure a compute service's performance behavior, a consumer is restricted to remote access to the system. A restricted access to the target system creates requirements regarding the form of the performance measurement.

The following subsections examine how a performance analysis can be conducted by compute service consumers. In addition, the peculiarities of virtualized systems are discussed in regards of compute service benchmarking. Also, the nature of metrics used to represent benchmarks is explained. Finally, repetitions of performance benchmarks in the light of non-deterministic systems are discussed and an overview of existing performance benchmarking software is provided.

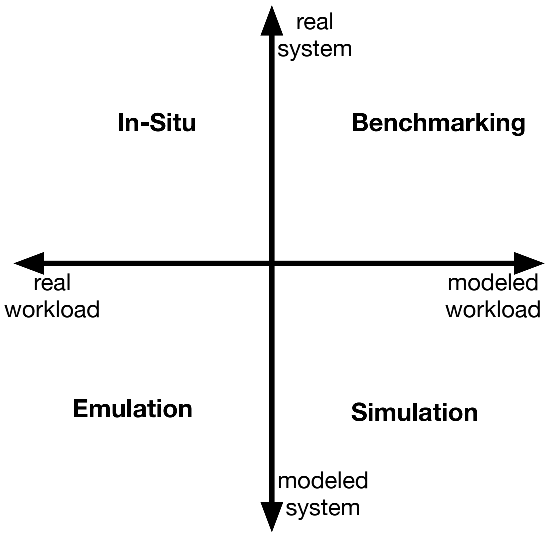


Figure 2.7.: Classification of Performance Evaluations [77]

Performance Benchmarking by Compute Service Consumers From a consumer-side, the access to a compute service in a performance analysis is restricted to the API offered by the provider and remote access to VMs. The physical hardware and virtualization layer are inaccessible to the consumer. Consequently, a performance analysis deals with a black box system and must focus on strategies applicable in the given situation.

An in-situ evaluation is possible by developing a real system running ontop a VM of a compute service and monitoring its behavior in production or testing stage. The results, however, remain very specific to the system. In contrast, performance benchmarking can be applied to compute services without necessarily developing a system. Existing benchmarking software provides systems that generate modeled workload on a VM of a compute service. The results provide general insights into a system's performance behavior and can be compared between systems. However, benchmarking results miss the accuracy and specificity of an in-situ evaluation with a production system.

Moreover, software systems are often operated in a complex environment. Depending on the environment, a system is confronted with diverse tasks to be dealt with, such as cronjobs, network requests, or operating system maintenance. Therefore, a system can show a non-deterministic performance level. A benchmark that is repeated on a system at different points in time can show a variation in the results [28].

Particulary in virtualized systems, a non-deterministic performance degradation for processor, memory, and I/O of VMs can be observed [30, 78]. Despite a hardware support for virtualization software, processor performance degrades up to 5%, memory up to 97%, and for I/O up to 28% according to results by Huber et al. [30] using the Passmark benchmark.

Even though virtualization software aims at an isolation between VMs, the influence of shared hardware usage is still recognizable [79]. Repeated performance benchmarking of compute services has shown that specifically processing power varies between different VMs of the same service and over time [80]. With AWS EC2's m1.small compute service the available average processor performance is halved in one-third of the available VMs due to two distinct processor types. Therefore, a repetition of performance benchmarks is necessary to gain a clearer overview of the actual performance and variations to be expected by a service.

This work seizes the insights from performance measurements exposing the behavior of virtualized compute services and includes benchmarking repetitions. In this regard, it is necessary to define metrics and aggregate benchmarking results from multiple runs. The following paragraphs address these aspects.

Metrics in Performance Benchmarking A metric is the unit in which a benchmark result can be expressed. The metric defines in which context a value must be interpreted and indicates that values of an identical metric feature a comparability. The features available in a comparison depend on the scale of a metric. While a *nominal scale* does not enable to compare values, an *ordinal scale* allows for rankings. Besides, *interval scales* and *ratio scales* facilitate comparisons in numeric terms. Interval scales support to compare values and identify a difference. With ratio scales, values can even be compared in relation expressed as a ratio.

Commonly, benchmarks are expressed in metrics correlating with *elapsed time* or *number of completed cycles*. Depending on the benchmark metric, a lower or higher value is more favorable to a system engineer and consumer. Generally, three metric types can be distinguished for benchmarks [31].

- *Higher is Better (HiB).* The higher the value expressed in such a benchmark metric, the better is the perceived value. E.g., computation cycles, throughput.
- *Lower is Better (LiB).* The lower the value expressed in such a benchmark metric, the better is the perceived value. E.g., computation time, latency.
- *Nominal is Better (NiB).* The closer the value expressed in such a benchmark metric resides to a defined optimum, the better is the perceived value. The defined optimum can be an arbitrary value between the maximum and minimum of a scale. E.g., utilization.

Aside from the performance metric, also the variations over multiple measurements must be considered. While in deterministic systems variations may be neglectable or absent, in non-deterministic systems, as in virtualized systems, variations are present. It is necessary to also assess variations, such as standard deviation or coefficient of variation, which are typically expressed in LiB metrics.

Repetitions in Performance Benchmarking The performance of hardware reflected in a benchmark is typically subject to variations over time [28, 31]. Other software running on a system and the external environment can have an influence on the performance achieved in a benchmarking. Systems showing non-deterministic variations cannot be captured in a single benchmark run.

Therefore, benchmarking needs to be repeated multiple times to gain meaningful statistics. An aggregation of multiple benchmarking runs requires the use of an aggregation function to acquire a summarized, single result for comparisons.

From a pessimistic perspective, the minimum function can be an adequate performance result aggregation. Conversely, with an optimistic view, the maximum function provides an eligible performance result aggregation. Moreover, mathematics and statistics offer more aggregation functions like mean, mode or median which aim at reflecting a central tendency. In addition, percentiles help to incorporate the variability of benchmarking results. Any of the aggregation functions can be applied to set of measured benchmarking results to determine a summarized value.

The following sub-sections present exemplary mean and median aggregation functions from mathematics and statistics.

Median The median is defined for a sample $\{x_1, x_2, ..., x_n\}$ as follows:

$$\overline{x} = \begin{cases} x_{\frac{n+1}{2}} & n \text{ odd} \\ \frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}) & n \text{ even} \end{cases}$$

Arithemtic Mean The arithmetic mean is defined for a sample $\{x_1, x_2, ..., x_n\}$ as follows:

$$\overline{x} = 1/n \sum_{i=1}^{n} x_i$$

Geometric Mean The geometric mean is defined for a sample $\{x_1, x_2, ..., x_n\}$ as follows:

$$\overline{x} = (\prod_{i=1}^{n} x_i)^{1/n}$$

Harmonic Mean The harmonic mean is defined for a sample $\{x_1, x_2, ..., x_n\}$ as follows:

$$\overline{x} = \frac{n}{1/x_1 + 1/x_2 + ... + 1/x_n}$$

Performance Benchmarking Software This work capitalizes on the abundance of benchmarks and related benchmarking software publicly available to automate compute service performance measurements. First of all, the structure of available benchmarking software and several common software packages shall be explored in the following. Finally, the Phoronix benchmarking software is presented that serves as a basis for later software prototypes in this work.

To measure performance aspects of a system in practice, a plethora of benchmarking software is publicly available and contains implementations of all sorts of benchmarks. Benchmarking software is typically packaged for execution on a wide range of systems. Therefore, benchmarking software exists in multiple versions and implementations for diverse hardware and operating system variants. Apart from that, software for benchmarks specific to certain system types exist as well, e.g., benchmarks for Unix systems only. Then, implementations are only provided for a certain system type.

Several benchmarking software have become more common as tools for measurements and comparisons than others [36, 37]. A list of examples of popular benchmarking software is given in table 2.2.

Name	Metric
Whetstone [81, 82]	Mio. Whetstone Instructions Per Sec. (MWIPS)
LINPACK [83]	Mio. Floating Point Oper.s Per Sec. (MFLOPS)
Dhrystone [84]	Program Iteration Completions Per Sec.
CPU2006 [85, 86]	Run-time Seconds

Table 2.2.: Sample of Common Performance Benchmarks

A large number of benchmarking software executes only a single benchmark task, a micro-benchmark, in a measurement run to gain a benchmark score. In contrast, benchmarking software that comprises a set of benchmark tasks is referred to as a benchmarking suite. Benchmarking suites commonly cover varying performance aspects and benchmark tasks are executed sequentially [87–90]. The sum of the executed benchmark tasks, sometimes weighted, equals the final benchmark score a system achieved. A benchmarking suite provides more diverse metrics to compare systems and allows for simpler execution of benchmarks. Since executing benchmarking software packages would need multiple steps, a benchmarking suite bundles diverse benchmarking software packages in a manner that allows a single execution step.

There is existing software that enables bundling of benchmarking software into suites [91, 92]. For example, Phoronix [91] offers a large set of existing benchmarking software [2] that can be composited to suites and executed on any computer with a PHP interpreter.

To use Phoronix, a bundle of PHP scripts need to be installed on the target system to be benchmarked. Using the scripts, a set of benchmarks can be bundled into a suite. Aside from custom assembled suits, predefined suits are available to execute multiple benchmarks sequentially on the target system. Any benchmarking software needed for benchmarking is automatically downloaded from Phoronix's repositories during the process. Results are exported as XML files.

In a global parameter, Phoronix can be advised to repeat each benchmark a specific number of times or until a maximum standard deviation is underpassed. The Phoronix software aggregates results from multiple runs in an average (arithmetic mean), maximum and minimum score.

2.2.2. VM Images

Compute services offer sets of VM images to consumers in order to simplify the building process of software to be run on their service. In general, VM images differentiate according to a range of attributes, from time of creation and access permissions to contained software and required disk volume.

In the following, first the extent of available meta-data about VM images is explored. Finally, the field of software management and its relation to VM images is described.

[2]available on open-benchmarking.com, accessed 2014-01-15.

Available VM Image Meta-Data

Compute service providers usually offer mechanisms to annotate VM images with meta-data and make this information accessible via APIs or on their websites. Information, however, is restricted to few meta-data that describes the status of an image. Table 2.3 shows the different meta-data attributes examined in the APIs of the compute services AWS *EC2*, *Rackspace* Cloud Servers and *GoGrid* Cloud Servers.

Attribute	EC2	Rackspace	GoGrid
ID	✓	✓	✓
Name	✓	✓	✓
Timestamp Created	×	✓	✓
Timestamp Updated	×	✓	✓
Status Accessibility (Public, Non-Public)	✓	×	✓
Status Availablity (Active, Saving, Deleting)	✓	✓	✓
Status Progress (Modification Progress)	×	✓	×
Operating System	×	×	✓
Price	✓	×	✓
Geo Location	✓	×	×
File Location	✓	×	✓
Owner	✓	×	✓
Architecture	✓	×	×
Category/Type	×	×	✓
Description Text	✓	×	✓
Hypervisor	✓	×	×
Disk Volumes	✓	×	×

Table 2.3.: Meta-data Attributes accessible via Compute Service APIs

An open-source software called OpenStack [93] that provides the basis to operate infrastructures in the private cloud deployment model offers extended support of meta-data for VM images. In addition to the meta-data available at compute service providers, OpenStack allows to associate key-value pairs to any VM image [94]. The meta-data is stored at the VM image service and can be accessed via web service interfaces [95].

Software Management

As VM image contain data and software, a primary concern is to configure software based on the initial state provided by an image. Initially, existing software must be detected within a VM of an image to establish a basis for further configurations. The state of software provided by a VM image can either be repeatedly detected when needed or persistently stored as meta-data of the image. Based on the current state of an image, an extended software configuration can be built and stored as a new VM image if needed. The process of software configuration may be conducted manually or with methods and tools which help to automate the configuration.

In this regard, the field of configuration management [51, 52] establishes and researches a wide range of concepts, methods, and tools to manage and detect software configurations.

To install software packages on a computer system, diverse methods and tools exist. Installation bundlers (e.g., InstallShield [96], DPKG [97]) envelop software packages in a self-contained bundle that can install and uninstall the software package. Besides, package management systems (short: package managers) that access maintained repositories with software packages are able to install and uninstall packages [98]. Furthermore, package managers can resolve dependencies between packages and automatically install required dependent packages. Prominent package manager implementations are APT based on DPKG [97, 99], YaST [100], and RPM [101].

In conjunction with installation and package manager tools, Operating System Configuration Management (OSCM) software builds whole software stacks orchestrating software installations on computers to be remotely managed by the software [102, 103]. Depending on the implementation, scripts or declarations describe what actions need to be executed on an operating system to build the stack [104–106]. Package managers and installation bundles help with remotely managing computers by installing software packages constituting a software stack.

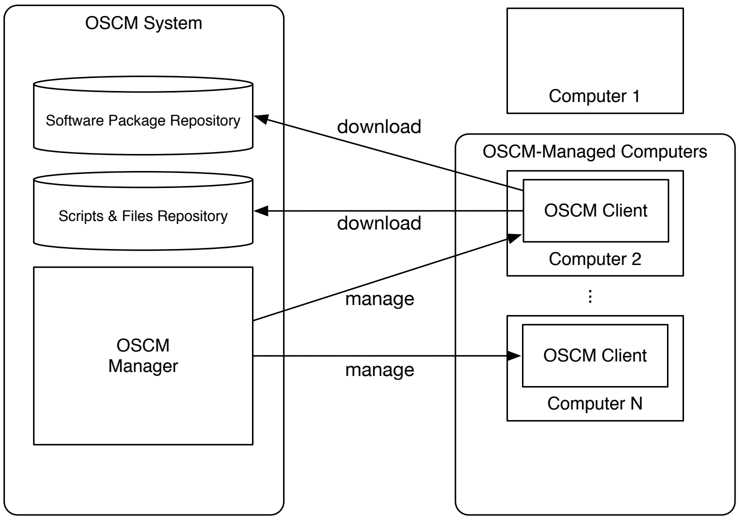


Figure 2.8.: Typical OSCM System Architecture

Figure 2.8 depicts the typical architecture of OSCM systems. A manager component takes over the part of orchestrating the configuration tasks applied to the managed computers. Initially, on every managed computer, a client software needs to be installed. The installation process is typically carried out via remote protocols (e.g., SSH, telnet) by the manager component. By employing repositories, the manager component can let clients remotely download and install software, execute scripts and programs, or insert and replace files with system parameters or data.

In an initial phase of a software stack build, already installed software needs to be detected to avoid conflicts and skip unnecessary steps. Therefore, existing OSCM implementations include a detection component which can document existing software packages and operating system attributes by accessing system tools and package managers. The snippet in listing 2.1 provides an example of a system libraries detection result documented with Ohai on an Ubuntu Linux system. The Ohai tooling is part of the Chef Configuration management system, a OSCM implementation from Opscode [107].

```
...
"python2.7": {
  "version": "2.7.1-5ubuntu2",
  "description": "An interactive high-level object-
                  oriented language (version 2.7) "
},
"python2.7-minimal": {
  "version": "2.7.1-5ubuntu2",
  "description": "A minimal subset of the Python
                  language (version 2.7) "
},
"readline-common": {
  "version": "6.2-0ubuntu1",
  "description": "GNU readline and history libraries,
                  common files "
},
"rsync": {
  "version": "3.0.7-2ubuntu3",
  "description": "fast remote file copy program
                  (like rcp) "
},
...
```

Listing 2.1: Opscode Chef Ohai Output from a Ubuntu 11.04 Natty System

2.3. Formal Decision-Making

Decisions have to be made on various levels and situations, from political or strategic business decisions to what to wear on a particular day. One way to settle for a choice is to stress mathematics and numbers. The field of formal decision-making provides methods that allow to evaluate options with mathematical tools. Methods of this field are described as MCDM methods. All methods in this field allow to define preferences and constraints regarding multiple criteria and find a pareto-optimal solution.

One strength of a subset of MCDM methods is scoring, which is the ability to express the quality of decision options in a value. With a scoring, options can be compared on a scale and differences can be denominated in percentages. In this work, scoring plays an important role to allow compute service consumers to compare services. Therefore, this section is primarily occupied with MCDM methods from a scoring perspective.

The following subsections give an overview of the field of Multi-Criteria Decision-Making and present one method from that field. The presented method, the Analytic Hierarchy Process, will be part of the framework contributed by this work. Consequently, an overview and classification of existing methods is given. The comparison of methods substantiates the motivation to settle for AHP as the method of choice in the framework presented in chapter 4.

2.3.1. Multi-Criteria Decision-Making (MCDM)

The field of Multi-Criteria Decision-Making (MCDM) engages with the wish to find an optimal solution to a decision problem regarding multiple criteria. Methods of that field help to map the quality of alternative solutions to a value, expose pareto-optimal solutions and determine preferred or non-dominated solutions.

Multi-Attribute Decision-Making vs. Multi-Objective Decision-Making

MCDM can be separated into two groups of decision problems: (1) Multi-Attribute Decision-Making (MADM), and (2) Multi-Objective Decision-Making. While (1) is concerned with predetermined, discrete solutions, (2) faces competing objective functions and constraints that determine viable solutions. In MADM, an evaluation of large sets of discrete solutions, also called search spaces, regarding a decision-maker's preferences can imply high computational efforts. An approach to tackling large search spaces is translating a multi-attribute decision into a multi-objective decision to find a solution and pick a close, discrete solution.

Other than multi-attribute-oriented approaches, in multi-objective decision-making, methods find an optimal solution by searching an infinite solution space along multiple dimensions. A search considers constraints set to a dimension or relations between dimensions. An optimal solution is either a viable global pareto maximum or minimum. A pareto solution is not a guaranteed optimal discrete solution, since even discrete solutions close to a global (pareto) optimum must not necessarily be a globally optimal discrete solution. In multi-objective decision-making the plurality of methods originate from various decision problems and algorithms.

Multi-Attribute Decision-Making Methods

Multi-Criteria Decision-Making (MCDM) methods can be categorized by the point in time when a decision-maker specifies his preferences, i.e., a priori, interactively, or a posteriori [108]. In Multi-Attribute Decision-Making preferences are always required a priori [19].

Furthermore, the applicability of a MADM method can be determined by the available information. Figure 2.9 depicts a taxonomy for MADM methods including a range of exemplary methods. The amount of information and knowledge about relations between data, e.g., by an ordinal scale, increases the precision of a method. At the same time, a method must be able to deal with many information and involve knowledge about scales in its evaluation. Multiple methods are able to cope with information about multiple attributes and map values on a cardinal scales (see Figure 2.9). The Analytic Hierarchy Process is the only method that also allows hierarchical relations between attributes. Also, it is able to normalize attribute values and result values on a [0, 1] ratio scale.

2.3.2. The Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) is a method to make decision based on multiple criteria [109, 110]. The Analytic Network Process (ANP) is a generalization of the simpler AHP and extends criteria hierarchies to complex criteria networks. Both criteria hierarchies and criteria networks intend to model the real world situation of criteria influences more realistic than single dimension value functions. However, the ANP requires very complex criteria network definitions and value computations. In most cases, a simple network definition reflecting the special case of a hierarchy is sufficient.

Criteria weights can be determined by pairwise comparisons what is a fundamental concept in the AHP. Also, pairwise comparisons help to normalize attribute values of alternatives to a $[0, 1]$ ratio scale. Using a super matrix filled with weights of a criteria hierarchy and the results of all pairwise attribute comparisons the AHP can compute a value on a ratio scale for each alternative. The structure of the super matrix and the

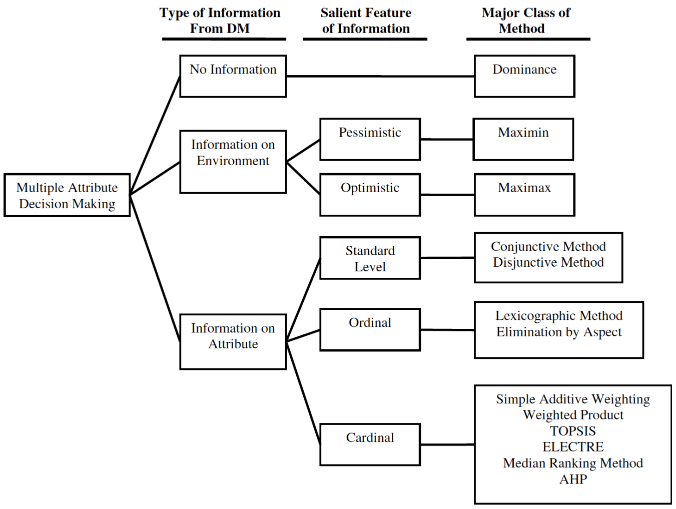


Figure 2.9.: Taxonomy of Multi-Attribute Decision-Making Methods [19]

numeric operation to gain the scores is explained in the following subsections. Also, the sections introduce extended concepts of the AHP by elaborating on indices and the concepts of synthesizing modes in rank reversals.

Criteria Hierarchies

A criteria hierarchy models a complex structure of alternatives, criteria, and goals. In the AHP criteria hierarchies allow to group criteria. The number of levels in a hierarchy is not limited. On the top level one or more goals head the hierarchy. Goals can be diverging or yield towards the same direction. The AHP proposes four merits for goals: benefits, opportunities, costs, risks (BOCR) [111]. Criteria grouped under goals may have metrics that point in a positive or negative direction. For positive metrics a higher value is better, for negative metrics a lower value is better. Figure 2.10 shows an example of a goal hierarchy with the two competing goals *features* (positive) and *costs* (negative).

A hierarchy structure and its multiple levels influence how weights of criteria are determined. The gravity of a criterion relies on the weights of related goals in upper hierarchy levels. The calculation of the final value incorporates all weights of the hierarchy.

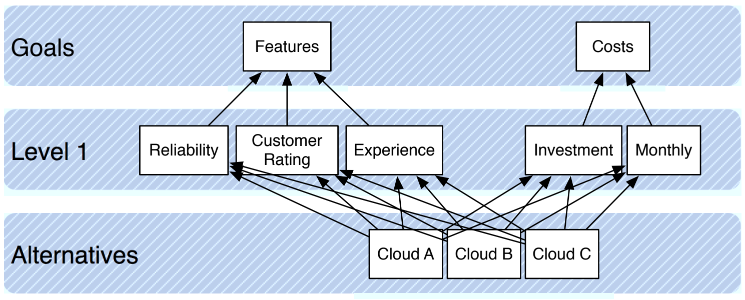


Figure 2.10.: Example of a AHP Hierarchy

$$
\begin{array}{c|c}
 & \begin{matrix} A_1 & \dots & A_n \end{matrix} \\
\begin{matrix} A_1 \\ \vdots \\ A_n \end{matrix} & \begin{bmatrix} \frac{w_1}{w_1} & \dots & \frac{w_1}{w_n} \\ \vdots & \ddots & \vdots \\ \frac{w_n}{w_1} & \dots & \frac{w_n}{w_n} \end{bmatrix}
\end{array}
$$

Figure 2.11.: Pairwise Comparison Matrix [112]

Pairwise Comparisons

The AHP uses pairwise comparisons of elements within a hierarchy level to insert criteria values into $[0, 1]$ ratio scales and to calculate weights. A pairwise comparison can be done for n alternatives A_i by creating a comparison matrix as shown in Figure 2.11. Alternatives must be compared only regarding criteria in the lowest level of the hierarchy. Criteria weights are derived from comparisons for each group on every level.

A pairwise comparison of alternative A_i in row i and alternative A_j in column j regarding a criterion results in a comparison value $\omega_{i,j}$ that expresses the ratio ω_i/ω_j of A_i's value in contrast to A_j. After all pairwise comparisons have been computed, on the matrix's diagonal all values will be 1 and under the diagonal are the inverses of the values above the diagonal. The normalized comparison result vector ω can be obtained by calculating the eigenvector with $A\omega = n\omega$.

To calculate local weights on one level in the hierarchy the comparison matrix is applied similarly. The matrix holds columns and rows for the affected criteria c_1 to c_n. Again, in each cell the two weights are compared with their values with ω_i/ω_j. The normalized local weights result from the eigenvector with $A\omega = n\omega$.

Calculation of Rankings and Scores

In the AHP pairwise comparison and normalization of weights and values is the prerequisite to map alternatives to a final value per goal. The ranking and scores of all alternatives regarding a goal are calculated in two steps. First, global weights ω_g must be calculated for all criteria on the lowest level (leaf criteria). A global weight ω_g is computed by multiplying local weights along the path of the hierarchy, from goal to a lowest level criterion. Global weights will still remain normalized but reflect the actual weight of a leaf criterion. A final value can be calculated by multiplying the global weights and the normalized values of leaf criteria and summing up the resulting values. For n leaf criteria, this step can be formally expressed as $\sum_{i=1}^{n} \omega_i v_{i,A_j}$ for alternative A_j. The ranking of all alternatives results from ordering by score.

Using Indices

Indices are an extended variant of applying the AHP to gain a score for alternatives. They allow the combination of scores from parallel AHP results. This is in particular of interest when considering multiple goals. AHP suggests indices based on the four merits benefits, opportunities, costs, and risks (BOCR) [111]. This also addresses the need to differentiate between positive and negative criteria. The four merits imply four parallel AHP computations with criteria hierarchies. Then all four scores are put into a relation by a function. A simple index for BOCR suggested by Saaty is $idx_1 = \frac{B+O}{C+R}$. A normalization of indices is optional but will map the alternative index values to a ratio scale.

Normalization and Idealization

In case an additional alternative or criterion is added into a decision ex post, rank reversal can occur when using the typical normalization mode. To avoid this behavior, the AHP also offers an idealization mode. While the usually used normalization mode makes pairwise comparisons which result in vectors with a sum of 1, the idealization mode discards this notion and sets the value for the best element to 1. Other alternatives are assigned a value relative to the value of the best element. The best element is called the ideal element.

In idealization mode, rank reversal is avoided when the original ideal element keeps its original value 1. The values of new elements that are rated better than the original ideal element can achieve a value >1.

2.4. Theory of Optimal Stopping

As the assessment of many compute services is costly, a mechanism to reduce the number of compute services is beneficial. The notion of stopping in a sequential traversal of the compute service set according to rules is an approach to achieve a cost reduction and budgeting followed in this work. The theory of optimal stopping has already developed rules that determine a stop under consideration of predefined criteria. Therefore, the concepts and existing stopping rules from this field shall be examined in the following.

The theory of optimal stopping is concerned with problems incorporating a solving algorithm with an infinite or finite number of steps [48, 113]. The intricate question to be solved is finding a stopping rule that determines to stop at a certain time and maximizes utility (or minimizes costs).

2.4.1. Stopping Problems

A stopping problem is described by a sequence of independent, random variables $X_1, X_2, \ldots, X_n$ whose joint distribution $F(\cdot)$ is assumed known [113, 114]. Then, a sequence of utility functions

$$u_0, u_1(x_1), u_2(x_1, x_2), \ldots, u_\infty(x_1, x_2, \ldots)$$

map a valued utility to the observations

$$X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$$

which are one-dimensional. After each observation, one can either decide to stop or keep on gathering information.

A stopping problem is classified as a finite horizon problem with a finite number n of observable random variables. Finite horizon problems are a special case of infinite horizon problems and, thus, allow for different stopping rules. A stop is always guaranteed in a finite horizon problem since only a finite number of observations can be made.

2.4.2. Stopping Rules

A stopping rule (1) always leads to a stop and (2) returns an optimal or acceptable solution in respect of the utility (or costs). The goal of a stopping rule is to lead to a stop to maximize utility, given that every further observation of a random variable X_{n+1} may discover a better solution but induces costs.

The following subsections present two famous examples in which a decision-maker faces problems similar to benchmarking a set of compute services. Similarly to benchmarking, any formerly discovered information and option can be recalled. The given examples help to understand Stopping Theory-approaches and the definition of appropriate stopping rules in the context of an information search – as adapted for benchmarking resp. meta-data assessment in this work. Finally, several standard stopping rules are presented which target stopping problems where the distribution function of the random variables $X_1, X_2, \ldots, X_n$ is unknown.

Secretary Search Example

The search of a secretary is a classic example for stopping rules [115, 116]. Let X_i be the i-th applicant and $X_1, \ldots, X_n$ be the random variables for n applicants. After every X_n the employer has to decide for X_n or continue with X_{n+1}.

For a finite horizon of n jobs, a viable solution to the stopping problem is then to reject k applicants and accept a later applicant with a value higher than any applicant before. The number k can be calculated as follows [116]

$$k = n/e$$

This leads to a stopping rule where approximately 37% of the jobs should be skipped and then the first job with a value higher than all observed jobs should be accepted.

Job Search & House Selling Example

When a prospective employee is searching and applying for a new job, job offers can be ranked without ties and the search process implies costs for every additional offer [117]. Job offers can be evaluated by the job seeker according to his preferences and are received on a daily basis. Let X_i be the job offer received on day i and $X_1, \ldots, X_n$ be the random variables for n days. The costs to receive an offer per day are $c > 0$, and after every X_n the job seeker can accept X_n while X_{n+1} may result in a higher offer, not known in advance.

The utility of n offers is:

$$\begin{aligned} u_0 &= 0 \\ u_n(x_1, \ldots, x_n) &= x_n - n \cdot c \quad \text{for } n = 1, 2, \ldots \\ u_\infty(x_1, x_2, \ldots) &= -\infty \end{aligned}$$

Similarly, in a house selling situation, a best offer has to be found while costs for the search rise daily for additional living costs and expenditures to host open house events [114]. For a house selling, let X_i be the offer on day i and $X_1, \dots, X_n$ be the random variables for n days. The costs per day are $c > 0$, and after every X_n the seller can settle for the highest X_i while X_{n+1} might promise a higher offer.

The utility of n offers is similar to a job search, however, the chance to recall any past offer allows to draw a maximum:

$$\begin{aligned} u_0 &= 0 \\ u_n(x_1, \dots, x_n) &= max(x_1, \dots, x_n) - n \cdot c \quad \text{for } n = 1, 2, \dots \\ u_\infty(x_1, x_2, \dots) &= -\infty \end{aligned}$$

To gain the highest utility, a stopping rule needs to determine an optimal or acceptable $u(X_1, \dots, X_n)$ that serves a seller a high X_i for his real estate. For both cases, the expected value can support a stopping rule since the distribution is known.

Standard Stopping Rules

There are various stopping rule approaches in the cases of a known or unknown distribution function [118–120]. In the following, stopping rules for the case of unknown distribution function are presented. Particularly, stopping rules with a reservation wage, a number of cut off observations, a predetermined candidates count, a successive non-candidates count, and the odds algorithm.

Reservation Wage

A viable stopping rule is to stop when a constant value w, referred to as the reservation wage, is surpassed. A simple example is the rule $u_n \geq w$ with a predetermined w, which means stop if the value of u_n is bigger (or equal) the constant value w (reservation wage) [117].

The reservation wage w can be determined as optimal if the distribution $F(\cdot)$ is known [121]. Then, in a job search in a sample of size N, the reservation wage w must be found such that the following statement holds:

$$c = \int_w^N (x - w) dF(x)$$

Cut Off Rule

A simpler stopping rule is the "cut off rule" [120]. This stopping rule halts with the first observed absolute highest value after m observations have been made. The rule does not consider occurring costs for additional observations.

The chances that the highest value is under the first m is $\frac{m}{N}$ (for a sample size of N). Then, the stopping rule will stop at N as no absolute highest value can be found after m. In contrast, if between position m and the position of the absolute highest value $max(x_1, \ldots, x_N)$ exists an intermediate highest value, this rule will fail in finding the absolute highest value $max(x_1, \ldots, x_N)$.

Generally, m should be a number not too large, in order diminish chances that the highest values is under the first m observations. At the same time, m must be large enough to possibly include some higher results, so the stopping rule ends with an absolute highest number $max(x_1, \ldots, x_N)$.

Candidate Count Rule

Unlike the cut off rule, the candidate count rule [120] does not ignore any observations. Every observation which increases $max(x_1, \ldots, x_n)$ is a candidate and increases the count. The parameter m determines after which candidate count a stopping is initiated. The m-th candidate stops the search and is the the highest value result. If an m-th candidate is not observed, the rule stops after the last observation and the last candidate is the highest value.

Successive Non-candidate Rule

The successive non-candidate rule [120] sets the space between two candidates into its focus. The stopping rule halts after a certain observation sequence. The sequence requires that after candidate i further m non-candidates must be observed. Then the rule stops the search with the $i + 1$-th candidate. If no such sequence occurs, the rule stops after the last observation and the last candidate is the highest value.

Odds Algorithm

For the odds algorithm [119], the probability for each observation to discover the best value must be determined. For each observation of random variable x_i, the variable I_i indicates whether the observed value is a best value candidate ($I = 1$) or not ($I = 0$). A best value candidate is at hand when $max(x_1, \ldots, x_n)$ increases. The probability that variable I_i indicates a best value candidate is expressed by p_i.

The odds algorithm defines $q_i = 1 - p_i$ and $r_i = \frac{p_i}{q_i}$ which are incorporated in the stopping rule decision. A stopping index s is calculated with the sums $Q_k = q_N q_{N-1} \cdots q_k$ and $R_k = r_N + r_{N-1} + \cdots + r_k$. The index s is determined by the first k which lets $R_k \geq 1$ surpass 1. The stopping rule stops at the first $I = 1$ after index s has been passed. The chance of finding the highest value is $R_s Q_s$.

3. State of the Art and Related Work

Before the contributions of this work are presented, this chapter gives an overview of meta-data publicly available for compute services. In addition, the state of the art and related work to assess and maintain meta-data is surveyed.

The first section is concerned with management of compute service meta-data. Particularly, how compute service meta-data can be stored and made available to consumers. Then, the field of research addressing comparative assessments of compute services is explored. In this regard, works that introduce methods to assess compute service in respect of one specific or numerous attributes are presented. Furthermore, methods to compare compute service according to a single or multiple attributes are presented.

The final two sections are concerned with methods available to assess compute services in regards of two particular aspects: (1) performance, (2) VM images. Due to the comprehensive amount of work in the field of compute service performance benchmarking methods, the works most related to this work are highlighted in the former section. Also, approaches to automatically repeat benchmarking processes are explored. Finally, the latter section lays out existing work with approaches to enrich VM images with meta-data and to introspect VMs.

3.1. Compute Service Meta-Data Management

Meta-data is used to describe data and assets and has become more extensive over time [122, 123]. In additions, systems to store and manage the abundance of meta-data are becoming more important, but at the same time more complex, too [122]. For example, the quality of a service in terms of latency and throughput is essential to media file streaming systems and to store observed qualities, a meta-data storage becomes necessary [124].

In the realm of web services, meta-data has been used to describe properties and qualities. The Universal Description, Discovery and Integration (UDDI) specification and related description languages, such as web service description language (WSDL) and web service policy (WS-Policy) as part of WS-* specifications, provide means to describe web services on a technical level or in respect of custom attributes [125, 126].

In a more extensive description approach, semantic technologies attach meta-data to web services by building a whole ontologies [127, 128]. The semantic web community, however, thrives for searchable web services and matching algorithms, while the focus of this work is the assessment and comparison of cloud compute services which provide web service interfaces.

Moreover, the use of a web services frequently comes with an agreement between the provider and consumer. One particular aspect is the service level agreement in which both parties agree on qualities, such as response time and availability, of a service. To enforce an adherence to the agreement, web services are monitored in regards of Quality of Service (QoS) attributes and tested for objective violations with approaches the likes of WS-Agreement [129] and the web service level agreement (WSLA) framework [130].

Similarly, approaches to enforce service level agreements have been proposed for cloud compute services [131–135]. All of the approaches commonly collect QoS attributes of compute services, such as latency, performance, and bandwidth, and implicate the existence of a data model and meta-data storage without describing it.

Moreover, approaches which use semantic technologies to describe compute services exist. Sundareswaran et al. [136] propose a list of 10 meta-data attributes used in a compute service ontology and in a selection method. The approach lacks insights on the management and storage of the meta-data kept in ontologies.

Unlike the aforementioned approaches which miss to describe details of a necessary compute service meta-data management and storage, Rodero-Merino et al. [137] introduce a file based storage for compute service attributes. The approach proposes an extension to the Open Virtualization Format (OVF) to describe compute services with one section of the document (*KPISection*) containing attributes of a compute service freely defined by the provider. The provider is obliged to update the section and containing attribute values frequently.

Similarly, [138] extend WSDL documents to describe compute service attributes and state. Based on the attributes, the approach supports a simple selection of compute service VMs for clusters of machines. By using WSDL or OVF meta-data is distributed to many decentralized files which has the disadvantage of increased effort for browsing and searching for compute services.

In contrast, CloudHarmony [139] offers an extensive central database of performance meta-data and a web application that can be used to browse through performance-related meta-data of diverse compute services. Thereby, CloudHarmony has introduced a performance meta-data management system for compute services. Although CloudHarmony collects the meta-data with own measurement methods, the system offers no integration with measurement instruments available to consumers. Therefore, the use of CloudHarmony is limited to a database of performance meta-data of compute services. Figure 3.1 depicts a screenshot of the performance meta-data database made available through the CloudHarmony website.

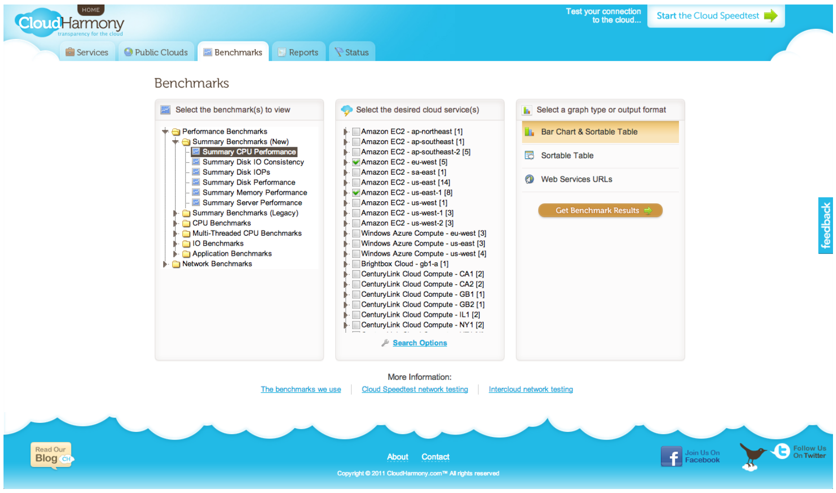

HOME
CloudHarmony
transparency for the cloud
Test your connection to the cloud...
Start the Cloud Speedtest
Services
Public Clouds
Benchmarks
Reports
Status
Benchmarks
Select the benchmark(s) to view
Performance Benchmarks
Summary Benchmarks (New)
Summary CPU Performance
Summary Disk IO Consistency
Summary Disk IOPs
Summary Disk Performance
Summary Memory Performance
Summary Server Performance
Summary Benchmarks (Legacy)
CPU Benchmarks
Multi-Threaded CPU Benchmarks
IO Benchmarks
Application Benchmarks
Network Benchmarks
Select the desired cloud service(s)
Amazon EC2 - ap-northeast [1]
Amazon EC2 - ap-southeast [1]
Amazon EC2 - ap-southeast-2 [5]
Amazon EC2 - eu-west [5]
Amazon EC2 - sa-east [1]
Amazon EC2 - us-east [14]
Amazon EC2 - us-east-1 [8]
Amazon EC2 - us-west [1]
Amazon EC2 - us-west-1 [3]
Amazon EC2 - us-west-2 [3]
Windows Azure Compute - eu-west [3]
Windows Azure Compute - us-east [3]
Windows Azure Compute - us-west [4]
Brightbox Cloud - gb1-a [1]
CenturyLink Cloud Compute - CA1 [2]
CenturyLink Cloud Compute - CA2 [2]
CenturyLink Cloud Compute - GB1 [1]
CenturyLink Cloud Compute - GB2 [1]
CenturyLink Cloud Compute - IL1 [2]
CenturyLink Cloud Compute - NY1 [2]
Search Options
Select a graph type or output format
Bar Chart & Sortable Table
Sortable Table
Web Services URLs
Get Benchmark Results
feedback
More Information:
The benchmarks we use
Cloud Speedtest network testing
Intercloud network testing
Read Our Blog
About
Contact
Copyright © 2011 CloudHarmony.com™ All rights reserved
Join Us On Facebook
Follow Us On Twitter

Figure 3.1.: Screenshot of CloudHarmony Performance Meta-Data Service

Generally, the amount of meta-data management systems specifically for compute service meta-data is scarce. CloudHarmony offers a first approach, but does not publish the data model nor is it integrated with assessment tools that can be used by consumers. Exactly these shortcomings shall be addressed in this work.

3.2. Comparative Assessments of Compute Services

Assessments are conducted in many contexts to map complex concepts to numbers, thereby building a foundation for comparisons. For example, the SERVQUAL framework [140] proposes an empirical assessment method to establish a service's quality score according to consumer perceptions. Similar assessment frameworks have been developed manifold for diverse aspects of services [141–143].

Nonetheless, real world services and e-services differ from compute services in many aspects which renders existing assessment frameworks too unspecific. Unlike other services, compute services are defined by attributes, such as the performance the rented machines offer, available remote interfaces, and software that can be deployed on the service (c.f. section 2.1). An assessment would need to take these attributes into account.

The following subsections explore assessment and comparison methods which are particularly applicable to compute services and consider primarily attributes thereof. First, existing methods to assess compute services at run- and non-run-time are presented. Then, a survey of existing approaches to compare cloud and specifically compute services is carried out.

3.2.1. Assessments of Compute Services

Compute services possess a range of attributes (c.f. sections 2.1 and 2.2) whereof only parts can be determined without the actual use of a compute service. For example, Habib et al. [144] propose a trust management system which uses diverse resources, such as audit certificates and questionnaires, to assess the trustworthiness of a cloud provider. In addition, many other works are concerned with trustworthiness of compute clouds and propose systems to establish trust between providers and consumers [145–147]. Klems et al. [148] incorporate an assessment of compute service price models in order to calculate costs for a given system. Similarly, many other researchers have examined approaches to measure and predict costs [149, 150] and only a subset can be mentioned in this work. All of the aforementioned approaches focus on an evaluation of compute services regarding a specific attribute and draw data from publicly available resources, often primarily meta-data attributes published by the provider.

In the field of migration planning, compute cloud offerings are assessed and eventually compared to identify a preferable fit as a migration target. Frey et al. [151] therefore introduce Cloud Environment Constraints (CEC) which test the suitability of a compute service according to diverse characteristics. The assessment and modeling of characteristics are, however, left to the consumer. Other approaches from the field include cost, risk, and security assessments, and evaluate the expected availability of the system during a migration [152–154]. Nonetheless, none of the approaches gives comprehensive support for the assessment process itself, but rather for the decision based on already assessed data.

All of the aforementioned approaches compile or consider meta-data from properties observable at any time. But, none of the approaches takes attributes of compute services into account which are observable at run-time. In contrast, Ristenpart et al. [155] introduce a method to assess compute services at run-time in respect of cross-VM security and vulnerabilities. Nevertheless, the method lacks a support for reuses of the run-time assessment scheme for other purposes which is a goal of the framework introduced in this work.

Many assessment methods focus on the aspect of compute service performance which is measured at run-time, too. An overview of the plethora of works proposing approaches to measure compute service performance is provided in section 3.3.1. Similarly, section 3.4 presents methods which allow consumers to introspect VM images of a compute service at run-time.

3.2.2. Comparisons of Compute Services

The comparison of competing compute services is a major purpose for meta-data. A range of existing work has proposed methods to compare compute services, but also other cloud services.

Rheman et al. [156] furthermore compare multiple MADM methods in an evaluation of compute services according to meta-data about performance, hardware and costs. The comparison comprises the methods Promethée, TOPSIS, Electre, AHP, min-max/max-min, and compromise programming. The meta-data for a test set of thirteen compute services is drawn from the CloudHarmony website. The meta-data set includes cost, memory (in GB), a benchmark for harddrives and network (IOP), and values for memory and processor performance using the CloudHarmony Compute Scoring (CCS) metric translated into Amazon's ECU (referred to as CCU) [157, 158]. The survey shows the applicability of MCDM methods and, specifically, of the AHP for compute service selection. However, Rehman et al. miss to derive a method that provides means to create custom scoring functions with a consumer-defined selection of criteria and weights.

A literature survey conducted by Rehman et al. [40] presents existing multi-criteria cloud service selection methods. The methods examined in the survey focus on the problems of a provider selection and a service selection. Table 3.1 summarizes the results of the literature survey.

In the following, several of the methods examined in the surveys by Rehman et al. are highlighted and an overview of methods not covered by the survey is given. The work of Han et al. [160] included in the survey presents a recommender method for cloud providers. For each provider, network QoS attributes are assessed for offered software and platform services. In addition, infrastructure services are considered in performance measurements conducted on VMs with three types of benchmarks (floating point, fixed point, and memory). The measured benchmarking results are normalized against a reference value measured on a "native" physical machine. A final ranking of cloud providers is determined from a weighted sum of the averaged network QoS and the averaged performance measurements.

Similarly, Li et al. [32, 159] propose CloudCmp, a comparison method that incorporates compute and storage services, and network qualities in an evaluation of cloud providers. For each of the merits several measurable metrics and measurement methods are defined. CloudCmp provides an approach with the intention to bring measuremnt

Work	Theme	Approach
Gocinski et al. [138]	State of the art in cloud computing, Cloud service discovery and selection	Overview and comparison between cloud, grid and cluster computing
Li A. et al. [32, 159]	Cloud service comparison, application performance prediction and cost estimation	Cloud benchmarks and testing of application prior before cloud deployment
Han et al. [160]	Cloud service recommender system	Network QoS and VM performance of available services are proposed as selection criteria
Zeng et al. [38]	Cloud service selection	Maximum gain and minimum cost optimization
Godse and Malik [161]	Cloud service selection	MCDM, Analytic Hierarchy Process (AHP)

Table 3.1.: Literature Survey of Cloud-Related Multi-Criteria Selection Methods [40]

methods and comparisons closer together, but misses to link both tightly. Besides, an aggregation of the results to a single score of a provider is missing.

Zeng et al. [38] present simple additive weighting functions to calculate a value representing the gain and cost of selecting a certain compute service. The additive weighting functions are then used in a selection algorithm. The algorithm selects the cloud service with the highest value according to the used additive weight function and tests its availability. In case of an unsuccesful initiation of a cloud service the algorithm accepts a degradation and continues with the next service in the list. Generally, the approach remains vague in regards of a definition of a "cloud service" and its applicability in decisions. It claims to be applicable to any sort of cloud service, but lacks providing criteria, measurement instruments, or an implementation.

Garg et al. [162] propose SMICloud, a framework and index score to assess and compare cloud services in regards of diverse criteria. The work describes the notion of a global index score to make compute services comparable and drive competition between providers. Nonetheless, the framework lacks comprehensive support for customization and is not suited for comparisons according to specific attributes and related weights.

A comprehensive approach for comparing diverse cloud deployment models (c.f. section 2.1.1) in respect of diverse workload categories, such as database or web servers, has been proposed with the Multi-Attribute Decision Methodology for Adaption of

Clouds (MADMAC) by Saripalli and Pingali [39]. The approach introduces an additive weighting method that can be used with a catalog of attributes to select from.

The extensiveness of existing work shows the importance of comparisons between cloud services and particularly compute services. Many approaches thrive for a general comparison of services according to a wide range of different attributes. Other approaches excel with very specific metrics involved in a comparison. However, none of the aforementioned approaches provides extensive support to generate a score which unifies various attributes of compute services, such as performance or security. Particularly, the problem of combining attribute values in diverse metrics into a single score is not addressed. Besides, all approaches lack a concept to integrate measurement methods and a comparison method.

Therefore, the present works aims at supporting cloud consumers with a tool to generate custom scores to compare compute services in respect of a certain attribute assessed in diverse measurements and described in various metrics. Moreover, this work integrates measurement tools and comparisons by introducing a meta-data model.

3.3. Compute Service Performance Measurements and Comparisons

As performance is one of the prominent features of compute services, there is a large amount of research works that propose methods of measurement. Due to the inherent hardware changes of compute service data centers and multi-tenant usage of machines, repetitions of measurements can give more extensive insights in respect of variations. Therefore, approaches to automate and repeat compute service performance measurements become necessary.

The following subsections first survey existing approaches to measure the performance of a compute service with benchmarking. Finally, research works coping with the necessity to automate and repeat benchmarking runs in order to capture variations are discussed.

3.3.1. Compute Service Performance Benchmarking

In a virtualized environment as is a compute service, the actual performance of a provided VM cannot be deduced from hardware figures as with physical systems [30]. Consequently, a measurement of an independent and comparable performance benchmark is necessary to see a compute service in relation to other computer systems' performances.

Benchmarking compute services is not an undiscovered territory. Existing work has already succeeded with applying benchmarking to analyze compute services. In particular, Gillam et al. [33] have applied benchmarking to several compute services and conducted a comprehensive study of existing benchmarking software for compute services. Walker [34] and Jackson et al. [163] have benchmarked the high-performance computing capabilities of Amazon EC2, but do not introduce a reusable method.

All of the existing work focuses on providing first measurement results and none has yet continued to cover the aspects of customizability, automation, and reusability. None of the approaches allows a consumer to define a set of performance attributes to be benchmarked and a support of a variety of compute services is absent. In order to use the approaches, still extensive manual interaction and systems setups are necessary. These shortcomings are primary goals of the present work.

Moreover, several further approaches should be mentioned which describe a method for benchmarking that is either specifically targeted at or at least applicable to compute services. Binning et al. [43] propose a cloud benchmark that measures cloud services in terms of the metrics scalability, cost, peak-load handling, and fault tolerance. The approach focuses on web applications, similar to the application described by the TPC-W benchmark [164, 165]. Additionally, different consistency settings on the database layer of a web application are incorporated in the approach. Nevertheless, the proposed benchmarking approach does not address aspects such as performance variations, server locations, or scheduling of benchmarking runs.

Sobel et al. [166] developed CloudStone, a benchmarking method that measures performance on the application layer. Benchmarking results are gathered in a cost-per-user-per-month metric and reflect the varying costs occuring per served user on different Amazon EC2 hardware configurations and with different software configurations. From the results, also the maximum number of users can be implicated. The approach, however, only reflects the overall transaction performance of one specific application. Making predictions about the general performance offered by a compute service is not possible. The method requires a setup of several application servers and a database server.

CloudHarmony [139] publishes a comprehensive database of performance meta-data on the web. Furthermore, CloudHarmony has introduced benchmarking sets comprising multiple benchmarks to cover diverse performance dimensions of compute services. Moreover, a metric named CloudHarmony Compute Scoring (CCS) has been defined based on a benchmarking set that seeks to become a standard for compute service comparisons.

Nonetheless, CloudHarmony does not allow users to request or trigger certain performance measurements. Also, the details of the employed measurement methods are neither available in the public domain nor have they been discussed in the research community. In addition, an integration with measurement tools available to compute service consumers is missing.

Apart from compute service benchmarking, there is existing work that focuses on measuring the effects of server consolidation and, in particular, the performance of VMs and the underlying physical host. For a compute service consumer, the access to the physical host or the hypervisor is typically denied. Nevertheless, an overview of the existing work exploiting access to the physical host or hypervisor shall be given for the sake of completeness as they provide insights on benchmarking of virtualized environments.

Matthews et al. [14], Jin et al. [16], and Iyer et al. [15] have developed benchmarks and methods to test the effects of shared resource sharing between VMs. The benchmark intends to measure the quality of a performance isolation and degradation of VMs when one or more VMs are under heavy workload.

Makhija et al. [167] propose VMmark, a benchmark that determines the capacity of a physical host running a VMware hypervisor by using multiple sets of six different, separated virtual machines that stress the system resources with workloads regarding diverse dimensions including CPU load, RAM load and I/O load, i.e., storage and network traffic. The final score considers the number of virtual machine sets, each consisting of a mail, java, standby, web, database, and file server, and the benchmarking results of each virtual machine in all sets.

Similarly, Casazza et al. [168] define vConsolidate, a benchmark that also aims at measuring performance of different workloads on a consolidated server. Mei et al. [169] propose another approach that focuses on network traffic as the constraining dimension in server consolidation and uses web server workload only.

3.3.2. Automation and Repetition of Performance Benchmarks

The physical hardware in data centers of a compute service is not necessarily homogeneous and performance outcomes can vary over time [80]. Due to overheads induced by the virtualization software, individual measurements can only give an approximation. The observable performance varies depending on the workload occupying the physical hardware shared between VM instances (c.f. section 2.2.1). Generally, workloads processed in parallel by multiple VMs on a physical system can induce a performance degradation [30].

The provided information represented by hardware figures and from indivial benchmarking results is not able to capture potential performance variations. Therefore, it is necessary to detect the performance variation interval in a set of measurements over time. In this regard, none of the aforementioned compute service benchmarking approaches allows to automatically execute benchmarking software and repeat measurements. There exist, however, several research works that cope with repetitions and measurements of performance variations.

Kalibera et al. [28, 170, 171] have introduced the concept of regression benchmarking to determine a precise representative performance grade for a system. The concept assumes a system to be found in a random initial state that leads to a non-deterministic behavior and influences performance outcomes. The proposed regression approach introduces an impact factor to capture the variations in repetitions of benchmark runs with multiple samples each. The impact factor is used to detect an initial state of a system and serves as a corrective factor between benchmarking samples of various runs. Furthermore, the impact factor together with cost estimations help to determine the required number of repetitions to presumably obtain precise and reliable results.

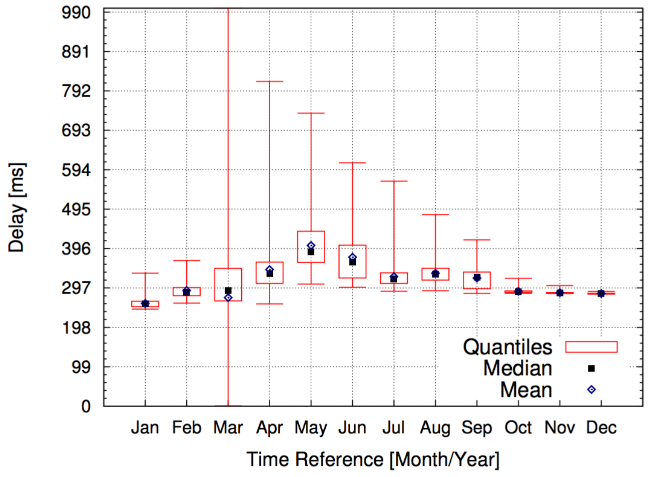


Figure 3.2.: Performance Variations on Google's AppEngine (Python) [13]

Moreover, Iosup et al. [13] benchmarked and recorded variations in the performance of various cloud services from Amazon and Google, including AWS EC2 and Google AppEngine. The statistics collected with more than 10,000 samples each month in 2009 proof that a variability is present. For AWS EC2 the deployment latency of VM instances and for Google AppEngine the execution time of a python Fibonacci computation script was measured. Figure 3.2 illustrates the observed performance variations of Google's AppEngine. In addition, Iosup et al. conducted a detailed performance analysis of VM clusters on AWS EC2 and GoGrid Cloud [44].

Similarly, Schad et al. [29] benchmarked Amazon EC2 over several weeks and present collected performance results and observed variations. For the representation of the variations, Schad et al. employ the coefficient of variation as indicator. The performance meta-data is measured with Ubench for memory and CPU resources, and Bonnie++ for hard disk resources. Besides, the instance startup time is recorded with a custom approach.

Although approaches to measure performance variations exist, in all of the aforementioned approaches an automation of the benchmarking process itself is absent. Nevertheless, several works propose automation for the detection of performance changes.

Kalibera et al. present an approach and generic architecture for automated benchmarks, referred to as regression benchmarking [35]. Regression benchmarking benefits from an automation due to its natural need to repeat measurements in the course of detecting performance changes. The original approach has evolved to BEEN [172], an architecture and system to automate benchmarking of distributed systems. BEEN offers a web interface from which benchmarking runs can be triggered. The results of a run are automatically stored to a database for later evaluations. Also, benchmarking software is automatically transferred to the hosts under test from a central repository. The system architecture is depicted in figure 3.3.

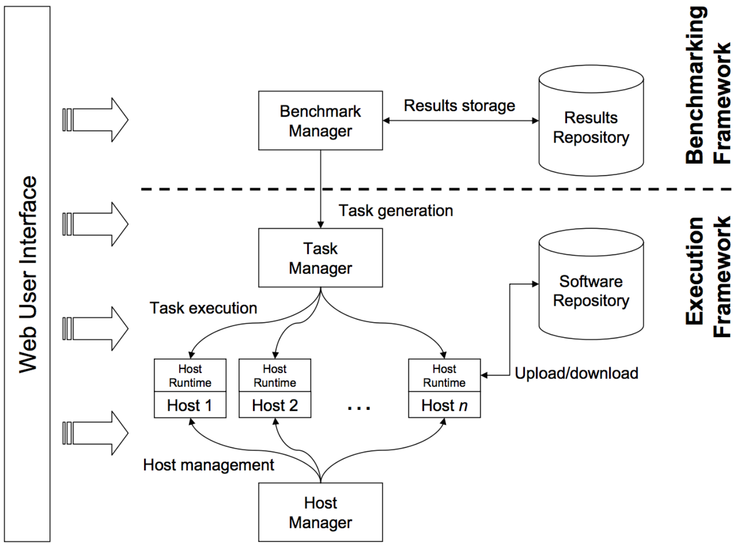
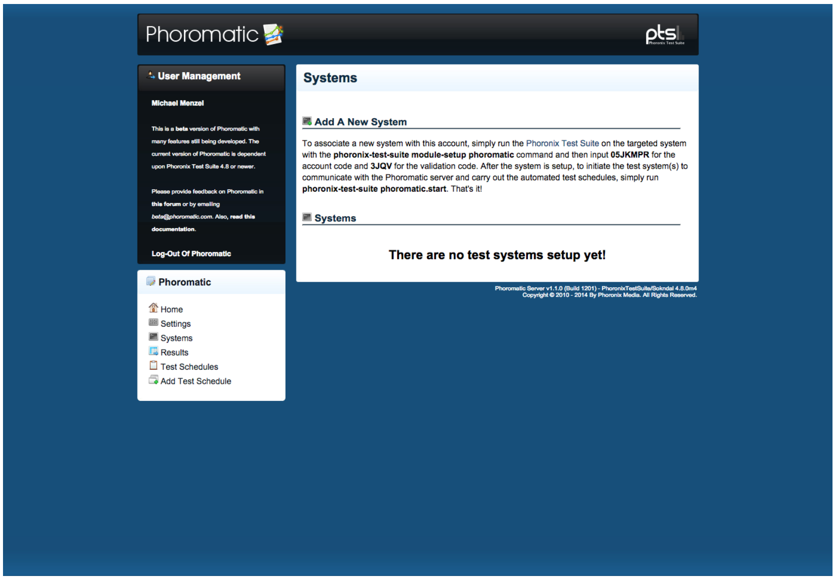


Figure 3.3.: Architecture of the BEEN system [172]

An adaption of the approach from Kalibera et al. has been employed to automate benchmarking runs to detect performance changes of software developed with the mono open-source development framework [173–176]. The daily builds of the development framework have been tested with the regression benchmarking approach for the last few years. Thereby, factors that influence the performance could be detected in every new build of the framework.

Regression benchmarking has further been transferred for middleware benchmarking by Bulej et al. [177]. The approach extends the former approaches by benchmarking middleware in the environment of a simulated bookstore web application. An adoption for cloud compute services has not been developed or published, yet.

In the realm of cloud services and particularly compute services, the research community has not yet provided applicable approaches. Nonetheless, a promising software named Phoromatic[1] aims to fill the void for remote benchmarking management systems [178]. The system allows to register existing systems and schedule automated benchmarking runs using micro-benchmarks from the Phoronix test suite [91]. The Phoromatic software is only available as online service. Figure 3.4 depicts the user interface of the Phoromatic online service.



This is a **beta** version of Phoromatic with many features still being developed. The current version of Phoromatic is dependent upon Phoronix Test Suite 4.8 or newer.

Please provide feedback on Phoromatic in **this forum** or by emailing *beta@phoromatic.com*. Also, **read this documentation**.

Systems

Add A New System

To associate a new system with this account, simply run the Phoronix Test Suite on the targeted system with the **phoronix-test-suite module-setup phoromatic** command and then input **05JKMPR** for the account code and **3JQV** for the validation code. After the system is setup, to initiate the test system(s) to communicate with the Phoromatic server and carry out the automated test schedules, simply run **phoronix-test-suite phoromatic.start**. That's it!

Systems

There are no test systems setup yet!

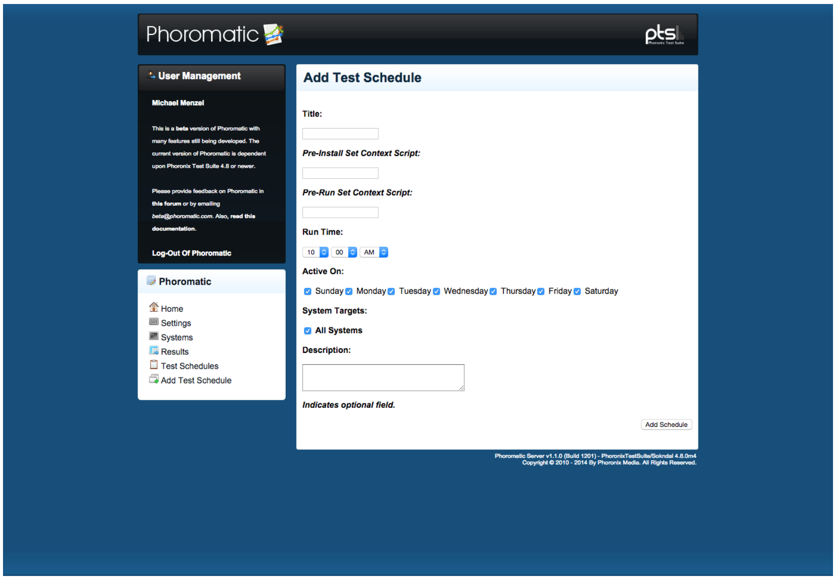
Phoromatic Server v1.1.0 (Build 1201) - PhoronixTestSuite/Sokndal 4.8.0m4
Copyright © 2010 - 2014 By Phoronix Media. All Rights Reserved.

Figure 3.4.: Screenshot of the Phoromatic Online Service

Before benchmarking schedules can be defined, a set of target systems – machines to be measured – must be registered with the service. Phoromatic cannot automatically instantiate virtual machines at a compute service. Therefore, virtual machines must be created manually and a registration of the machines as target systems can be achieved via shell commands and a validation code. Phoromatic requires the Phoronix test suite to be installed on a target system in order to permit a registration.

[1]in beta status, as of 2014-04-01.

Once a set of target systems is registered, multiple benchmarking schedules can be defined with a schedule time and weekdays as well as the set of systems to include in the benchmarking (c.f. figure 3.5). Besides, a set of benchmarks can be assigned to a schedule. Each benchmarking schedule is run on every selected weekday at the defined time for all target systems. During a scheduled run, all assigned benchmarks are executed on the system and the results are automatically uploaded to Phoromatic.



Michael Menzel

This is a **beta** version of Phoromatic with many features still being developed. The current version of Phoromatic is dependent upon Phoronix Test Suite 4.8 or newer.

Please provide feedback on Phoromatic in **this forum** or by emailing *beta@phoromatic.com*. Also, **read this documentation.**

Log-Out Of Phoromatic

Add Test Schedule

Title:

Pre-Install Set Context Script:

Pre-Run Set Context Script:

Run Time:

10 00 AM

Active On:

✓ Sunday ✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓ Saturday

System Targets:

✓ **All Systems**

Description:

Indicates optional field.

Phoromatic Server v1.1.0 (Build 1201) - PhoronixTestSuite/Sokndal 4.8.0m4
Copyright © 2010 - 2014 By Phoronix Media. All Rights Reserved.

Figure 3.5.: Screenshot of Schedule Definition in Phoromatic

Phoromatic is yet the most complete software available to run scheduled and partially automated benchmarks on virtual machines at cloud compute service. Nonetheless, the online service lacks support for automated measurements of compute services. Specifically, a user is required to create and register virtual machines manually rather than an automated instantiation of machines at diverse compute services. Moreover, a definition of schedules by time and weekdays is too coarse to analyze performance variations of compute services.

A more detailed evaluation of the Phoromatic service in comparison to the novel approach presented in this work is given in section 5.5.2. The comparison uncovers the disadvantages and shortcomings of Phoromatic in light of five experiments that stress its automation features.

3.4. Virtual Machine Image Introspections and Comparisons

Like with performance meta-data, a compute service consumer might be interested in VM image meta-data. Particularly, meta-data about image contents is beneficial, as it allows to find images and virtual appliances that align with functional software requirements.

In the following sections, existing work to maintain and assess VM image meta-data is described. First, approaches which aim at packaging meta-data with VM images are explored. Finally, works that follow the concept of run-time introspections of VM to gain meta-data are presented.

3.4.1. VM Image Meta-Data Attachment

Meta-data about contents of VM images, such as installed software and libraries, is rather scarce. Meta-data attributes, such as name and description, are typically used to describe the operating system and software stack contained in a VM image as unstructured text. Frequently, exact information cannot be covered in these attributes due to length and size restrictions. Additionally, raw text attributes are non-structured and can hardly be parsed for specific information, neither by machines nor humans.

One approach is to enrich VM images with meta-data through annotations on a file basis. The Reservoir model proposes an extension to the open virtualization format (OVF) which introduces meta-data annotations to the file format [179].

Similarly, Matthews et al. [180] propose to extend OVF with contract meta-data for various use cases. The extension aligns with OVF's XML nature and introduces diverse aspects that can be defined as meta-data of a VM image, such as network policies or reliability requirements.

Dastjerdi et al. [181] consider VM appliance meta-data in a semantic match-making algorithm. Meta-data on virtual appliances is stored as virtual units in an appliance management service using semantic technologies. However, the approach does not define a catalog of meta-data attributes in addition to information already available from the compute service providers. Also, the approach requires providers or contributors to manually maintain the meta-data.

Another approach that embraces semantic technologies is proposed by Reimer et al. [182] who have recognized a sprawl of virtual machine images. An image is converted to the Mirage image format along a process which incorporates meta-data extraction. The extraction generates VM image content meta-data that is observed with the Unix command *stat*. To store the image, the Mirage Library can be employed [183]. However, the content meta-data of an image is described by checksums of files only, tying the approach to detection of identical files only. Besides, the approach requires the adoption of a certain format for VM images.

In summary, available meta-data and approaches to collect and maintain meta-data fail to incorporate information about VM image contents. An approach to automatically extract meta-data about contents of a VM image is yet absent and therefore a primary goal of this work.

3.4.2. Introspection of VM Image Meta-Data

The concept of introspection describes the process of accessing and monitoring the internal execution environment of a computer system. The notion of introspecting VMs originates from work addressing security concerns in virtualization technology. Garfinkel et al. have developed an approach to detect intrusions using introspections through the Virtual Machine Monitor (VMM) of a VM. During an introspection, the running processes, the memory contents, and the I/O flags of a machine are tested for intrusions. Additionally, Pfoh et al. [24] propose a formal definition of VM introspections in the context of security applications.

As part of an assessment of VM image contents the detection of installed libraries and software is mandatory for a comprehensive meta description. Existing OSCM systems, such as Opscode Chef or Puppet [104–106], are able to detect the fund of libraries and software in a system using available package managers [51, 52]. Thereby, OSCM can list contents of a VM, once granted remote access. Yet, no system exists that is capable to pair and automate this approach with compute services to extract meta-data about VM image contents.

In relation to OSCM systems, Filepp et al. [184] developed a virtual appliance configuration approach that installs missing software on VMs and finds a virtual appliance which requires lowest effort. The approach uses a repository of appliance configurations that has been build based on the Galapagos system [185]. However, the approach is restricted in its flexibility and only detects few well-known enterprise software packages [186].

Moreover, Wilson and Matthew [187] create virtual appliances from multiple software packages with OSCM technologies, but neglect keeping the configuration of appliances in a meta-data database. Liu [188] shows how configuration meta-data allows an automated operating system configuration of appliances, but aims at configuration settings only and misses support for virtual appliances of compute services.

OSCM is a promising technology which is capable to capture the set of available libraries and software of a system. Yet, it has never been adapted or employed to describe contents of VM images and persist the results as meta-data about VM images of compute services. This work seeks to fill the void by adapting OSCM systems to assess compute services in terms of VM image meta-data.

Part II.

Conceptual Framework

4. Framework for Comparative Assessments of Cloud Compute Services

This chapter presents the methods and models which are part of the framework introduced by this work. In particular, this chapter provides an overview of the framework and sheds light on the aspect of interactions and relations between methods in the framework. In addition, the extensibility of the framework is discussed in this chapter.

Moreover, the models and method are explained in detail. First, a meta-data model is introduced that enables consumers to attach various meta-data to compute services. Then, an automation model is presented that provides consumers with an archetype of an automated procedure for run-time assessments of compute service. The framework additionally incorporates a method that is capable of assigning scores to compute services based on available meta-data. The scoring method is furthermore used to facilitate consumers with a stopping rule. In subsequent assessments of compute services, the stopping rule enforces a cost budget and leads to an early stop with the goal to save costs.

Finally, a software prototype implementing the scoring method is presented and used in an evaluation. The evaluation employs the prototype in a case study with the industry and to explore its computational complexity in practical experiments. The results show that the scoring method is applicable in practice and provides trustworthy scores.

This chapter incorporates an evaluation of the scoring method only. Other parts of the framework are evaluated in the context of instantiations in chapters 5 and 6.

4.1. Introduction & Overview

Meta-data captures characteristics and details of a compute service and allows consumers to understand and compare the services they are provided with. In chapter 3, the current state of the art to assess and compare compute services as well as sources and the available amounts of meta-data have been explored. To close the void of assessment methods for compute services that empower the consumer, this work introduces a framework of methods and models that enable consumers to single-handedly assess and compare compute services according to meta-data.

The framework facilitates compute service consumers with methods and models to capture assessed meta-data, enforce cost budgets in assessments, and assign scores in route of a comparison of compute services. The methods serve consumers in assessing compute services in respect of diverse characteristics convertible into meta-data. Particularly two characteristics are explored in detail in this work. Chapter 5 explains the use of the framework for assessments in terms of performance characteristics, and chapter 6 in terms of VM image contents. Both instantiations are built upon a subset of models and methods contained in the framework. Figure 4.1 depicts an overview of the models and methods constituting the framework.

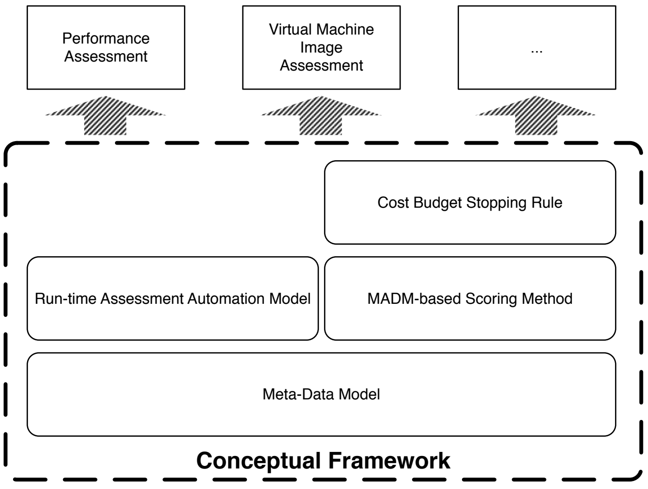

Performance Assessment
Virtual Machine Image Assessment
...
Cost Budget Stopping Rule
Run-time Assessment Automation Model
MADM-based Scoring Method
Meta-Data Model
Conceptual Framework

Figure 4.1.: Overview of the Conceptual Framework

As shown in figure 4.1, the framework comprises several generic methods and models:

- Cost Budget Stopping Rule
- MADM-based Scoring Method
- Run-time Assessment Automation Model
- Meta-Data Model

At the core of the framework is the *Meta-Data Model* which provides the basic structure to attach meta-data to compute services. Any instantiation of the data model can extend the basic structure to fit meta-data about certain characteristics. The *Run-time Assessment Automation Model* incorporated in the framework gives a basic layout and procedure to assess compute services at run-time in an automated manner and store results using the meta-data model. Given structured meta-data, the *MADM-based Scoring Method* incorporated in the framework can assign a score to compute services, thereby making compute services comparable by a single number. Moreover, a *Cost Budget Stopping Rule* reuses the scoring method to calculate utilities for compute services and suggest a stop in a sequential assessment. The stopping rule aims at reducing costs by leading to an early stop during assessments of multiple services and enforces a cost budget.

Adapting the generic models and methods, instantiations can be build that target an assessment of a specific characteristic of compute services. In this work, two instantiations of the models and methods are presented. Both instantiations adapt subsets of the framework's models and methods to assess meta-data regarding performance and VM images respectively.

Nonetheless, the framework leaves space for further instantiations to assess compute service in regards of other characteristics based on the generic models and methods. Besides, new instantiations may develop new methods that can be reapplied in course of assessments regarding further characteristics. The framework's extensibility is discussed in section 4.1.2.

4.1.1. Relations Between Models & Methods

The models and methods are in relation to one-another and can only be used in such. Figure 4.1 reflects the relation between the models and methods. Specifically, the cost budget stopping rule bases on the scoring method to determine a stop. Furthermore, the scoring method draws information from the data model and aggregates it into a single score using MADM.

The instantiations to assess compute services in regards of performance and VM images use the meta-data model and intersecting subsets of the framework's methods. Both instantiations assess compute services at run-time and share the notion of a multi-cloud interface gateway to interact with diverse compute services and providers.

Apart from that, both instantiations shed light on different characteristics of a compute service and, thereby, complement one another. By instantiating the framework's methods, a consumer is able to acquire meta-data single-handedly. Performance attributes and descriptions of VM image contents are attached to a compute service in joinable data models. Thereby, new meta-data about a compute service becomes available in a single database. Consequently, compute services can be compared according to an extensive amount of trustworthy meta-data.

4.1.2. Extensibility of the Framework

Although the framework is complete in its current state, it is generally open to be extended in regards of methods and models. Further methods and models may be included in future versions of the framework. Similarly, methods and models may be advanced in alignment with a changing cloud environment. New methods and models may arise as requirement for an assessment of more compute service characteristics and find its way into the framework as generic concepts.

The assessment of other compute service characteristics would be based on the framework's models and methods. Thereby, the amount of available meta-data about a compute service can be extended to further fund decisions. Additional characteristics must not necessarily be assessed at run-time as is the case for performance and VM images. An added characteristic, such as credibility or support quality provided by a compute service, may be assessed from online resources and interviews.

4.2. Meta-Data Model

The meta-data model introduced by this framework serves as an abstract basis to capture all sort of meta-data about compute services. Using the basic meta-data structure, specific data models extending the basic model can be developed to attach meta-data attributes to compute services. Figure 4.2 depicts a Unified Modeling Language (UML) diagram of the meta-data model provided by the framework.

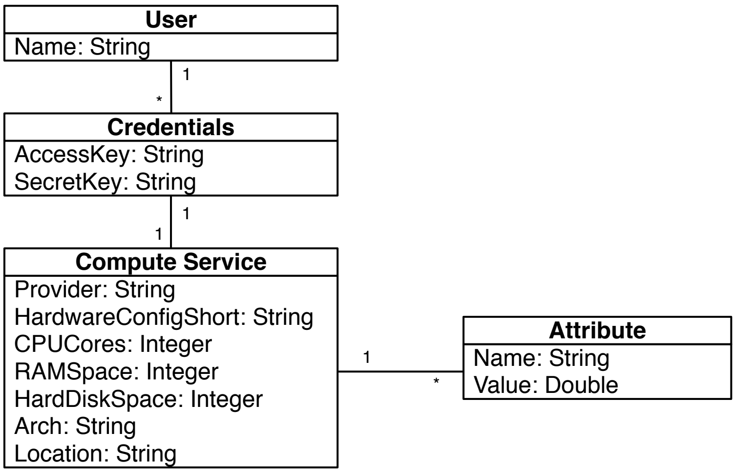


Figure 4.2.: Basic Meta-Data Model

The basic data model to capture meta-data of compute services comprises four entities: (1) *User*, (2) *Credentials*, (3) *Compute Service*, and (4) *Attribute*. The *user* entity represents the consumer who has multiple *credentials* necessary to access compute services. A *compute service* are linked to a certain provider and possesses several natural configuration attributes by which it can be distinguished from other compute service configuration. Additionally, attributes can be attached to compute services which consist of a name-value tuple.

4.3. Run-time Assessment Automation Model

With compute service APIs in place, automation of assessments is an obvious goal to reduce manual effort. Particularly assessments at run-time follow the notion of measuring characteristics of a compute service from within a VM and via its APIs. A compute service consumer is capable of accessing a VM via remote protocols and conduct immediate observations and measurements, which is in contrast to black box settings with no insights at all [189, 190]. Typically, the APIs provided by compute services demand only small effort to setup a VM for run-time assessments.

In contrast, non-run-time assessments cannot rely on exploiting VM operations to assess a compute service and are rather conducted fully manually. The focus of this framework, however, are assessments at run-time following the model for automation presented in this section. The methods and models introduced in this framework target and support automated assessments of compute services at run-time based on the following model.

The automation model describes the general archetypical procedure to conduct assessments of diverse compute services at run-time. A method and a system implemented according to the model needs to comprise components that provide a user interface, coordinate assessment tasks, and grant access to compute services. Besides, repositories to persist user parameters and assessment results are required. Figure 4.3 depicts the general automation model using compute services' APIs.

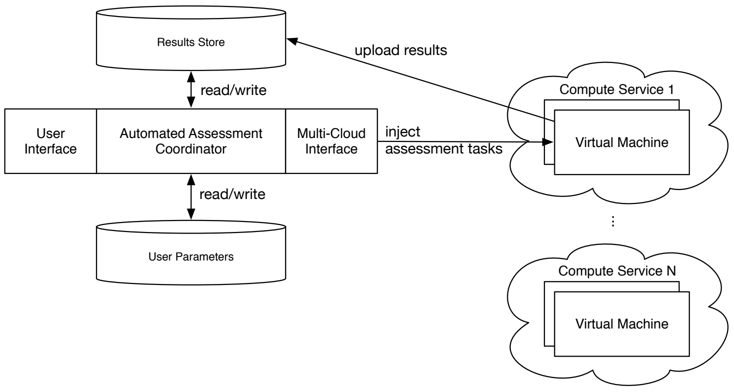
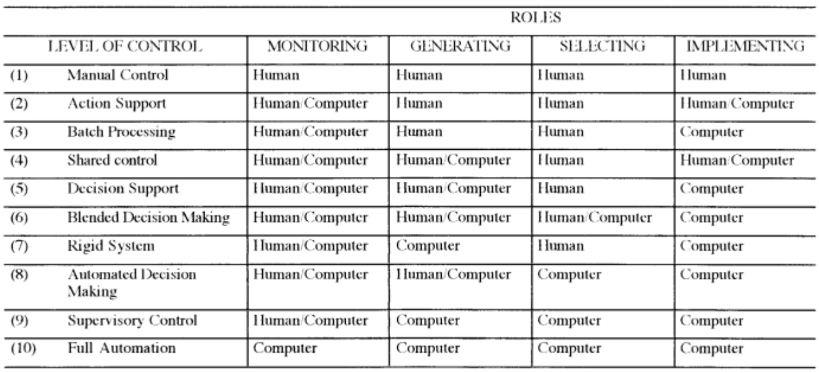


Figure 4.3.: Overview of the Model for Automated Assessments at Run-Time

Furthermore, based on the depicted model a procedure needs to be followed in an assessment. Firstly, the *user interface* gives means to define parameters, such as target compute services, before an assessment. Given the parameters, the *coordinator* accesses a set of compute services to inject assessment tasks. The coordinator must instantiate a VM with a preselected image and execute the task in the machine. Each task gathers insights from within a VM at run-time and uploads the results to a *repository*. The repository implements the meta-data model described in section 4.2 to capture the assessment results. A user can eventually access the results via the *user interface*.

The major goal, which the model and the linked procedure pursue, is automation with a minimum degree of human interaction and a maximum degree of configurability. To characterize the pursued level of automation in more detail, a classification shall be given according to the ten classes defined by Endsley [191–193]. The levels of automation as defined by Endsley are depicted in figure 4.4.



		ROLES			
LEVEL OF CONTROL		MONITORING	GENERATING	SELECTING	IMPLEMENTING
(1)	Manual Control	Human	Human	Human	Human
(2)	Action Support	Human/Computer	Human	Human	Human/Computer
(3)	Batch Processing	Human/Computer	Human	Human	Computer
(4)	Shared control	Human/Computer	Human/Computer	Human	Human/Computer
(5)	Decision Support	Human/Computer	Human/Computer	Human	Computer
(6)	Blended Decision Making	Human/Computer	Human/Computer	Human/Computer	Computer
(7)	Rigid System	Human/Computer	Computer	Human	Computer
(8)	Automated Decision Making	Human/Computer	Human/Computer	Computer	Computer
(9)	Supervisory Control	Human/Computer	Computer	Computer	Computer
(10)	Full Automation	Computer	Computer	Computer	Computer

Figure 4.4.: Levels of Automation [191]

According to Endsley's classification, the general automation scheme achieves the automation level *Batch Processing*. A user's part is to *generate* the implementation of an automated assessment procedure for a particular compute service characteristics. Besides, a user can set the parameters and, thereby, *select* the target compute services and details of the assessment procedure. The implemented assessment procedure will then automatically assess the selected compute services and, hence, *implement* the assessment tasks. Moreover, a user *monitors* the task implementation and retrieves the meta-data resulting from the assessment.

Moreover, prospective assessments using this framework are expected to develop a custom, concrete implementation of an automated assessment at run-time. Therefore, the task executed during the assessment must be defined and developed to be compatible with VM of the target compute services. Specifically, this requires to prepare the task to be executable within the environment of a VM, and to package and upload results to the repository. The data model implemented by the repository must be capable to persist the uploaded data.

4.4. Scoring Method

In this section, the method to assign scores to compute services based on available meta-data using MADM is presented. Initially, the potential of MADM to aggregate multiple values into a single representative value is explored. Then, the Multi-Criteria Comparison Method for Cloud Computing framework is introduced that provides the basis to create custom evaluation functions that enable consumers to apply MADM to numeric meta-data. Finally, the concepts of local and global scoring are explained in regards of compute service comparisons.

4.4.1. MADM for Value Aggregation

Methods from the field of MADM are able to aggregate values in diverse metrics to a single value while considering weights. Commonly, MADM methods support decisions with multiple discrete alternatives by providing comparable single-dimensional values. However, the methods are generic to cover a wide range of decision problems and need an adaptation to the specific scenario. An adaptation incorporates the definition of the alternatives to be compared and the criteria to be considered.

For the comparison of compute service meta-data, the diverse attributes are to be defined as criteria. Given an MADM equipped with meta-data attributes as criteria, an evaluation function can be generated that expects weights and compute service alternatives as inputs.

The present work introduces the Multi-Criteria Comparison Method for Cloud Computing ($(MC^2)^2$) which uses the Analytic Hierarchy Process (AHP) to create an evaluation function used in a meta-data-based scoring mechanism. Thereby, two scores can be created: (1) a local score that reflects the meta-data-based value of a compute service to a consumer relative to other alternative compute services given as inputs, and (2) a global score which compares the value of a compute service to a fixed reference compute service. The framework optionally accepts requirement definitions to filter out ineligible compute services that exceed certain constraints.

The use of the AHP is primarily justified with its capability to normalize values and produce score values on a ration scale. The normalization makes the AHP accept values in any HiB or LiB metric. Besides, the normalization is the key to assign scores to a ratio scale. Thereby, scores can be compared in relation to one-another and statements, such as "x is 2 times better than y" can be made. These features are absent in other MADM methods, e.g., Simple Additive Weighting or Median Ranking Method.

The following subsections present the $(MC^2)^2$ and the evaluation function that considers meta-data attributes as criteria. Also, the local and global scoring mechanisms are presented.

4.4.2. Evaluations with the $(MC^2)^2$

The Multi-Criteria Comparison Method for Cloud Computing ($(MC^2)^2$) presented in this subsection proposes a generic, linear process that ultimately leads to an evaluation function. The process incorporates several steps from a definition of the scenario, alternatives, and criteria to an evaluation and ranking of alternatives. The information gathered within the process is the basis to develop an evaluation function $f(\cdot)$ as a customized instance of a MADM method. The $(MC^2)^2$ can be used to develop evaluation functions of different natures, such as weighted sum and weighted product evaluation functions.

The process to gain an evaluation function $f(\cdot)$ suggests a step-by-step order to gather the needed parameters. First, a scenario and, thereby, a number of alternatives are defined. Next, relevant criteria and requirements are identified. A multi-criteria decision-making method is then chosen and subsequently configured. The configuration incorporates defining the criteria and requirements as parameters of the selected decision-making method. The result is a custom evaluation function that can be used to evaluate the alternatives under consideration.

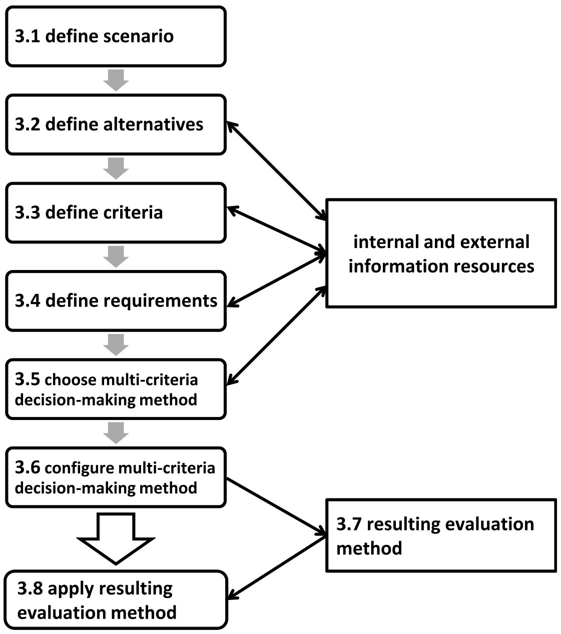

3.1 define scenario
3.2 define alternatives
3.3 define criteria
3.4 define requirements
3.5 choose multi-criteria decision-making method
3.6 configure multi-criteria decision-making method
3.7 resulting evaluation method
3.8 apply resulting evaluation method
internal and external information resources

Figure 4.5.: The Process within the $(MC^2)^2$

Figure 4.5 depicts the process steps, each of which is described in further detail in the following subsections.

Define Scenario

A scenario specifies the particular situation under consideration. Therefore, a scenario description incorporates the business and IT context, the environment, the goals, and the constraints. Thereby, the description of the scenario sets the scope and objectives for the evaluation method to be created. Comprehensive expertise and experience is required to define the scenario, and sufficient effort should be spent to facilitate the subsequent steps.

Define Alternatives

When using $(MC^2)^2$, at least two alternatives must be defined for a given scenario. The process of defining alternatives requires a research of the available options and associated attributes. The alternatives are typically given by the decision scenario.

Every alternative aims at being a solution to the scenario's goals. An attribute is a characteristic or property, preferably common to all alternatives, that can be assigned a value. Every pair of alternatives differs in the value of at least one attribute. Otherwise, the two alternatives are identical or an additional attribute to differentiate the alternatives should be introduced.

Identifying attributes of an alternative involves expertise, creativity, and experience, often from multiple stakeholders. The attribute identification process can also be supported with external information sources, such as literature, databases, or documentations [194]. It is possible to perform this search process systematically by first searching for a set of important attributes and then assigning values to each attribute, for each alternative. At the end, each alternative is defined by the values assigned to its set of attributes.

Define Criteria

Despite attributes providing a data source for an evaluation, in MADM methods a set of criteria must be defined. The definition of criteria selects attributes to be considered in the evaluation function and describes relations and dependencies between attributes. For example, criteria can be structured in hierarchies or networks to model the real world more precisely. Therefore, one of the fundamental steps within $(MC^2)^2$ is the definition of criteria. Having completed this step, requirements can be derived from the criteria.

The set of attributes determined during the definition of alternatives serve as a source to derive the set of criteria. Criteria add information to attributes that are necessary for a use in MADM methods. A criterion consists of a topic (or question) to be examined and has a type, either qualitative or quantitative, and a direction, either positive or negative.

In case of a quantitative criterion, a scale of measurement has to be defined. Qualitative criteria do not necessarily rely on a scale as they are not measurable. Not every MADM method is capable of considering qualitative criteria and require transforming qualitative criteria into quantitative criteria.

Furthermore, criteria influence the value of an alternative either positively or negatively. In particular, a criterion can be positive for an alternative the higher the value (higher is better) or the lower the value (lower is better). According to Saaty [195], criteria should be clustered into four merits: benefits, opportunities, costs, and risks. While criteria of the types benefits and opportunities are positive, criteria of the types costs and risks are negative. Nevertheless, other criteria type categories can be introduced as well. Table 1 shows four example criteria in the realm of performance meta-data where criterion #1 is a quantitative criterion with nominal scale, #2 is a quantitative criterion with ratio scale, and #3 is a qualitative criterion.

#	Question	Type	Possible Values
1	How high is the performance of the system?	quantitative, nominal scale	low, medium, high
2	How many FLOPS can be achieved?	quantitative, ratio scale	0$-∞$
3	How do you like the system's performance?	qualitative	—

Table 4.1.: Example Criteria for Performance Characteristics

To strive towards a precise evaluation of the alternatives, an extensive set of criteria must be defined. Pardee and Kirkwood [196] give three objectives to be pursued during the finding of attributes and corresponding criteria.

1. Completeness and Exhaustiveness
2. Mutually exclusive items only
3. Restrict to criteria of highest degree of importance

After all criteria have been identified and defined regarding their type and direction, they can be set into relation. Depending on the MADM, the criteria can be organized as a set, in a hierarchy or in a network.

Value Type	Req. Type	Boolean Expression
Numerical	Exclusive Max	$\gamma < v^r$
Numerical	Inclusive Max	$\gamma \leq v^r$
Numerical	Exclusive Min	$\gamma > v^r$
Numerical	Inclusive Min	$\gamma \geq v^r$
Non-numerical	Equals	$\gamma = s^r$
Non-numerical	OneOf	$\gamma \in S = \{s_1^r, \ldots, s_n^r\}$

Table 4.2.: Common Requirement Types

Define Requirements

In order to remove ineligible alternatives, requirements can be defined to filter out alternatives which are not feasible within the constraints of the given scenario. A requirement of a scenario is expressed as a constraint that demands a certain value, or defines a minimum or maximum boundary for a value. The definition of a requirement can refer to a criterion or requires the definition of a new criterion outside the criteria set. Table 4.2 lists available requirement types to define constraints.

Requirements can be defined for a value γ of the alternatives regarding a criterion as a numerical boundary v^r or a non-numerical boundary s^r. Numerical boundaries can be exclusive ($<$, $>$) or inclusive ($\leq$, $\geq$). Non-numerical boundaries can be a single value s^r or a set of boundary values S. Instead of the equal operator $=$ which tests for absolute identical string values, more sophisticated operators that weaken the requirement are thinkable, such as a substring test or regular expressions.

The procedure of filtering can be performed for minimum constraints with a conjunctive satisficing method and respectively for maximum constraints with a disjunctive satisficing method [19].

Choose Multi-Criteria Decision-Making Method

A range of MADM methods are eligible upon which a custom evaluation function can be constituted. An overview and comparison of different MADM methods can be found in [197][19]. The method has to be chosen according to the preferences of a decision-maker. For comparisons between alternatives, the evaluation function should produce score values, and not just pair-wise ratios or unvalued ranking. Therefore, the $(MC^2)^2$ recommends the use of weighted sum MADM methods like the AHP. Unlike weighted product functions that only reflect pair-wise ratios of alternatives, weighted sum functions allow for score values that reflect the absolute quality of an alternative. A score has the advantage of result conservation without the need to reapply the evaluation function.

The $(MC^2)^2$ suggests the Analytic Hierarchy Process (AHP) [195][198] as a favorable weighted sum MADM method, due to its ability to incorporate complex criteria hierarchies in an evaluation. Criteria hierarchies enable a more realistic modelling of criteria dependencies. Moreover, AHP employs pair-wise comparisons for normalization and to support the use of qualitative criteria. The use of pair-wise comparisons for normalization succeeds to assign values to both, quantitative and qualitative criteria on a ratio scale. However, qualitative criteria values are derived from subjective ratings.

Configure Multi-Criteria Decision-Making Method

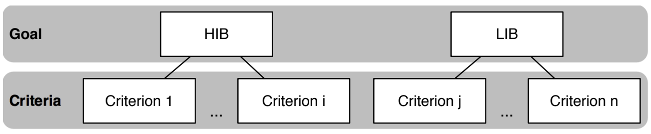
Before an evaluation function can be build and the evaluation results can be calculated using the function, it is necessary to configure the selected MADM method. This typically means setting parameters, such as calculation schemes, criteria, and normalization processes. Whether a configuration is needed or not depends on the MADM of choice.

For example, in the AHP the criteria hierarchy and the index to aggregate multiple hierarchies must be defined. The structure of a criteria hierarchy defined for the AHP entails effects on the ability to weight criteria against one-another and on the computational complexity. Generally, two distinct criteria hierarchy structures are followed: (1) a simple single-level hierarchy, and (2) a hierarchy with criteria groups. Figures 4.6a and 4.6b illustrate the two typical and simple criteria hierarchies.

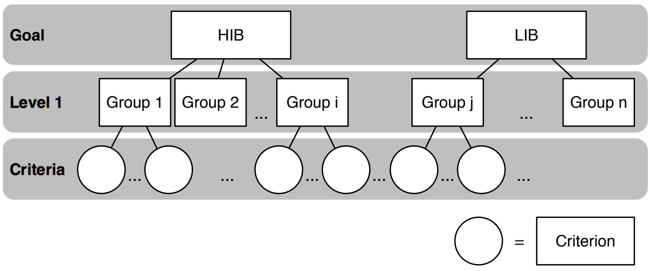
In a single-level criteria hierarchy, all criteria are grouped into two goals according to their metric: (1) positive goal for higher is better (HIB) metrics, and (2) negative goal for lower is better (LIB) metrics. The HIB goal subsumes all criteria that increase the compute service's score the higher the values of the criteria. Oppositely, the LIB goal subsumes all criteria with negative effects. Finally, the evaluation with AHP uses the index $\frac{HIB}{LIB}$ to relate both goals to another and to determine the final score value. The structure of the proposed simple criteria hierarchy is illustrated in figure 4.6a.

A grouped criteria hierarchy introduces an additional level of goals which groups criteria by meta-data characteristic. Thereby, the user can express a preference towards a certain characteristic by assigning a higher weight to the corresponding goal. Again, a $\frac{HIB}{LIB}$ index determines the final score value. The structure of a grouped criteria hierarchy is depicted in figure 4.6b.

Generally, additional levels introduced into the hierarchy through groups change the effort for weighting and how a decision-maker can influence weights of criteria. The local weight of a single criterion (within a group) is multiplied with the weight of the group (and the weight of all other grouping nodes on the shortest path to the root).



(a) Simple Single-level Criteria Hierarchy



(b) Criteria Hierarchy with Criteria Groups

Figure 4.6.: Typical Criteria Hierarchies

Therefore, to assign a specific weight, the global weight must be respected and weights of groups altered if necessary.

Moreover, by grouping criteria, the effort for pair-wise comparisons in the weighting process can be diminished. Depending on the number of criteria n and the number of introduced groups g, the number of pair-wise comparisons required for n criteria exceeds the number of pair-wise comparisons of n criteria divided into g groups. For a single level hierarchy the number of pair-wise comparisons for weighting are $\binom{n}{2}$. In g groups the n criteria implicate $\sum_{i=1}^{g} \binom{n_i}{2} + \binom{g}{2}$ pair-wise comparisons (with $\sum_{i=1}^{g} n_i = n$).

Apart from grouping, other approaches for reduction of comparisons exist [199]. All approaches seek to reduce pair-wise comparisons by asking the decision-maker for a minimal set of preferences to derive criteria weights from. For example, a bubblesort-based mechanism asks a decision-maker to reorder a list of criteria according to importance. All of the alternative approaches replace the default weighting process using pair-wise comparisons and are not necessarily beneficial regarding weighting efforts.

A reduction of the effort originating from pair-wise comparisons can be achieved by deliberately selecting the criteria a decision-maker is particularly interested in. Most criteria remain equally (un-)important and, hence, a weighting in pair-wise comparisons is not necessary. Unweighted criteria will be considered equally important per default by the AHP.

Resulting Evaluation Function

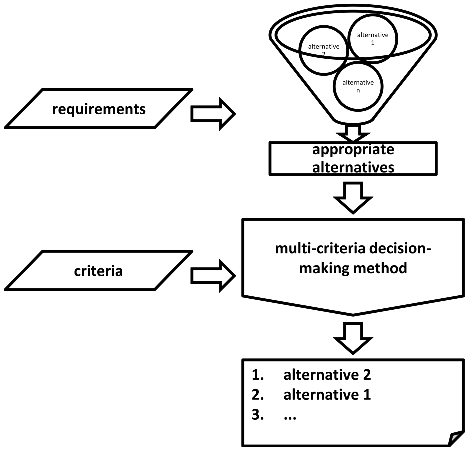


Figure 4.7.: Schema of a Resulting Evaluation Function

A schematic representation of the resulting evaluation function is depicted in figure 4.7. Given requirements are used to filter out alternatives that cannot fulfill one of the requirements. Following, all remaining alternatives are evaluated by the given criteria with the MADM of choice. To rank the alternatives, the alternatives can be sorted by their evaluation results. The alternative ranked as #1 is the preferable choice according to its overall value.

In case a weighted sum MADM method like the AHP is chosen, an evaluation function $f(a_j, W, A_{a_j}, R) \mapsto v_{a_j} \in (0,1)$ is generated with the $(MC^2)^2$. Function $f(\cdot)$ is formulated in Equation 4.1. The function returns v_{a_j}, which is the sum of the weighted values of all attributes α of an alternative a_j. The attributes are matching the corresponding set of criteria that was defined for the evaluation function. If the given requirements R are satisfied then $v_{a_j} \in (0,1)$, otherwise the function returns $v_{a_j} = 0$.

$$f(a_j, W, A_{a_j}, R) = \begin{cases} \sum\limits_{i=0}^{|A_{a_j}|} w_i \chi(\alpha_{i,a_j}) & \forall r \in R : r = true \\ 0 & else \end{cases} \qquad (4.1)$$
$$\mapsto v_{a_j} \in (0,1)$$

The function $f(\cdot)$ has the parameters a_j, which is the j-th alternative to evaluate, and set A_{s_j} of attributes, where $\alpha_{i,s_j} \in A_{s_j}$ is the i-th attribute of s_j. In addition, the set of weights W must be given as parameter, where $w_i \in W$ is the weight of the i-th attribute. For W the constraint $sum^W w_i = 1$ must be satisfied to maintain the normalization.

The value of an attribute is determined with the function $\chi(\cdot)$, for example, $\chi(\alpha_{i,s_j})$ for the i-th attribute of s_j. Furthermore, an optional set R can be given which contains all requirements that need to be satisfied. Table 4.3 lists all parameters of the evaluation function.

Parameter	Description
$a_j \in S$	The j-th alternative to be evaluated by the function $f(\cdot)$
$\alpha_{i,s_j} \in A_{s_j}$	The i-th attribute of compute service s_j
$w_i \in W$	The weight of the i-th attribute
$\chi(\alpha_{i,s_j})$	The function to gain the value of the i-th attribute
$r \in R$	A requirement that needs to be satisfied

Table 4.3.: List of Symbols in the Evaluation Function $f(\cdot)$

An evaluation function $f(\cdot)$ resulting from $(MC^2)^2$ is reusable with custom alternatives and associated attributes, weights, and requirements in decisions within the defined scenario's context.

Apply Resulting Evaluation Function

The resulting custom evaluation function is able to filter out ineligible alternatives and to calculate scores. Based on the scores, substantiated, rational decisions can be made by comparing and ranking alternatives. The evaluation function can be applied to any set of alternatives, associated attributes and requirements which must be given as parameters.

4.4.3. Local Scoring

For the comparison of a set of compute services, an evaluation function created with the $(MC^2)^2$ generates a local scoring. The function would accept the proposed criteria hierarchy and the meta-data as parameters. Each compute service's local score is expressed relative to the other scores on a (0,1) scale, and all scores sum up to a value of 1. The use of normalization also for indices that combine multiple goal scores is assumed.

The function also allows to select different criteria and adjust weights. After a re-computation using the altered evaluation function, updated scores can be gained for the set of compute services. Also, when adding another compute service to the set, the evaluation function needs to be reapplied and the scoring will change.

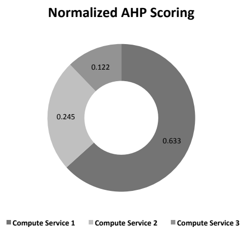
The locality of this evaluation function is due to the scale which is specific to a set of compute services. The sum of all scores in the compute service set will always be 1 for a local scoring. The local scoring immediately reflects the percentage of suitability – in regards of defined criteria and weights – for each compute service. However, results can only be interpreted in the context of the particular set of compute services the scale has been created for. A comparison of local scores from different compute service sets is not valid.

4.4.4. Global Scoring

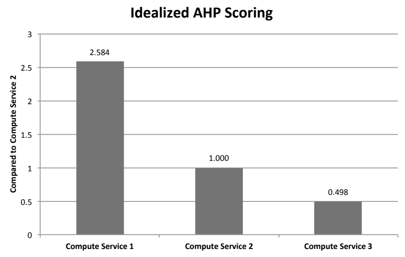
A score determined for a compute service in a local scoring is normalized in relation to other alternative compute services. Consequently, the scale introduced in a local scoring is only applicable to this specific set of compute services. In its original mode, AHP calculates scores on a (0,1) scale per goal where the sum of the scores is 1. When an additional compute service is added, the scores would change to retain the sum of 1.

To avoid the change of scores, AHP can be applied in an ideal mode synthesization method for the calculation of idealized scores. After a first local scoring with AHP, the best compute service can be chosen as ideal compute service and will be scored with a value of 1. Thereby, an idealized scale forms around the ideal compute service serving as the reference score. In ideal mode, any compute services introduced later are scored relative to the ideal compute service. In case the added compute service has a higher score than the reference score, it exceeds the value of 1 to reflect a correct ratio.

Figure 4.8 illustrates a normalized and an idealized scale of three compute services. In this figure, compute service 2 serves as the ideal reference in the idealized scale.



(a) Example of Normalized Scale



(b) Example of Idealized Scale

Figure 4.8.: Example of Normalized vs. Idealized Scale

The ideal mode of AHP can furthermore be used to compare compute services to a reference compute service in a global scoring. Therefore, one certain compute service is declared as a reference or benchmark. Every compute service's global score is calculated by the evaluation function used with two compute services as parameters: (1) the compute service to be scored, (2) the reference compute service.

Every calculated score is to be seen in context of the reference compute service and expressed as multiples of the reference score. Moreover, a global score is only comparable to global scores calculated with the identical set of criteria and weights.

4.5. Cost Budgeting Stopping Rule

In a search of a high scoring compute service with subsequent assessments costs can be saved by ending the process early. Particularly when it seems unlikely that one of the outstanding assessments identifies a compute service with a more favorable score. The stopping rule presented in this section bases its stopping decision on costs, a compute service score from the scoring method (c.f. section 4.4), and a cost budget. First, the cost calculation needed in a stopping decision is explained. Then, the setting of a stopping rule for compute service assessments is defined. Finally, suitable standard stopping rules are presented before a novel bespoke stopping rule adapted specifically to compute service assessments is introduced that considers cost budgets and exploits a preliminary order.

4.5.1. Cost Calculation

To enforce a cost budget, the stopping rule must be aware how much costs accrue with every additional assessment of a compute service. The costs generated by an assessment comprise the bill for using the compute service and the effort to carry out the assessment task. Besides, costs for the preparation and planning of the assessment, as well as for necessary resources occupied or purchased for the assessment project need to be considered. The following cost factors have a considerable effect as operational expenditures in the course of compute service assessments:

- Staff
- Resource usage for Assessment Implementation
- Compute service usage
- Network traffic

Additionally, capital expenditures may occur as requirement to conduct a compute service assessment, such as licenses for assessment and evaluation software.

The overall costs can be divided into expenditures for the project and operational expenditures per compute service. For the stopping rule to work, costs must be broken down per compute service. Therefore, the project expenditures must be divided by the total number of compute services, whereas compute service costs can be assigned directly. Since project expenditures are assigned to each compute service with the same amount, these can be left out. Ultimately, the total costs for assessing one specific compute service i must be defined as c_i.

Nevertheless, a perfect calculation of the per compute service assessment costs can be forfeit in exchange for simplicity. Although the fidelity of the stopping rule may suffer, a simple cost calculation remains a valid option in case costs cannot be determined precisely for all of the cost factors. Possible simplifications are (1) identical costs c_i for every compute service i and (2) inclusion of a subset of operational cost items. For the latter, the compute service usage is a factor immediately observable after a first compute service has been assessed.

4.5.2. Compute Service Scores & Ordering

One important aspect of in a sequential assessment using a stopping rule is the distribution of expected scores among the compute services under consideration. As none or only little meta-data is available beforehand beforehand, a distribution of scores among the compute services is unclear.

As costs for assessing compute services can exceed targeted budgets, only a fraction of the finite compute service set, called the sample, is actually traversed. Hence, it is impossible to assess perfect information in the search for a compute service. Consequently, the sample of traversed services includes the absolute highest scoring compute service only by chance.

For this reason, a deliberate choice of compute services to be included in the sample can increase the chance to include highest scoring compute services in the search. The sample can be influenced by defining a strict order for the sequential assessment process. An approximation of the ideal order can be achieved by preferring compute services that potentially score higher according to priorly available information.

Although the available meta-data regarding a compute service is scarce without a single-handed assessment, providers and other public sources provide numbers and descriptions that may allow to anticipate a scoring order. When no prior knowledge about meta-data is available, a random ordering must be assumed. Implicitly, however, the chances to include the highest performing compute service in the sample decrease.

Generally, there are two situations that lead to different sequential assessment orders:

- Manual order according to prior information and use of custom probability distributions
- Random order and use of single probability distribution

The following subsections will give insights on the two situations and describe the effects of an anticipated order and probability distribution.

Manual Order According to Prior Information

With prior information about meta-data attributes of any of the compute services, the set of services can be transmuted into an ordered list. Such information can be drawn from sources, such as the SLAs published on the providers' websites, results of assessments in the past, or experiences from other consumers. Composing an ordered list involves picking services and adding them to the list in a descending order, starting from the highest guessed score. The guessing of a score must be aligned with the defined evaluation function, particularly under consideration of the criteria and weights incorporated in the evaluation function.

The guessed score for $\mathcal{S}_i$ resembles the μ of the custom distribution function for $\mathcal{S}_i$. Additionally, with a σ the confidence of the guessed value can be expressed. The guessed scores interpreted as μ and confidence values σ leads to different distribution functions $F_i(\cdot)$ for all $\mathcal{S}_i$.

Figure 4.9 depicts the observed distribution of scores for an example list with N compute services that was ordered according to prior information. The distribution of the scores is not strictly descending in this example. Since prior information is commonly incomplete and imprecise, the fidelity of an approximated order is commonly unpredictable.

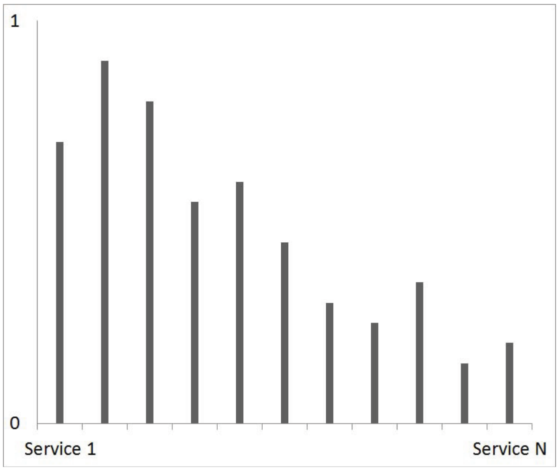


Figure 4.9.: Distribution of Expected Scores Among Ordered Compute Services

Nevertheless, an approximated order has advantages over a random order. Although the fidelity is unpredictable, chances are that the order outperforms a random order in various regards. Costs may reduce due to an earlier stop of the sequential assessment process. Besides, the chances increase that the actual highest scoring compute service is in the traversed sample.

Random Order

In absence of any prior information, the scores can be expected to be distributed randomly among all compute services. The actual value s_i of a variable $\mathcal{S}_i$ is observed with an assessment of the compute service and determined as score value by applying the evaluation function $f(\cdot)$. The randomly distributed variables $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are described by a distribution function $F(\cdot)$. Generally, the actual distribution function $F(\cdot)$ is typically not know for compute services but can be approached by standard distributions, such as a Gaussian or Poission distribution. The distribution of the score values among the traversed compute service sample influences the behavior of the stopping rule.

Assumed the random distribution is a well known distribution function, such as a Gaussian or Poisson distribution, the reservation wage stopping rule is able to determine the stopping index in advance (c.f. section 2.4.2). The costs accruing in the use of other stopping rules are unforeseeable. The number of traversed compute services is unknown in advance for a random order and random distribution. An example of observed score values for N compute randomly ordered services is depicted in figure 4.10.

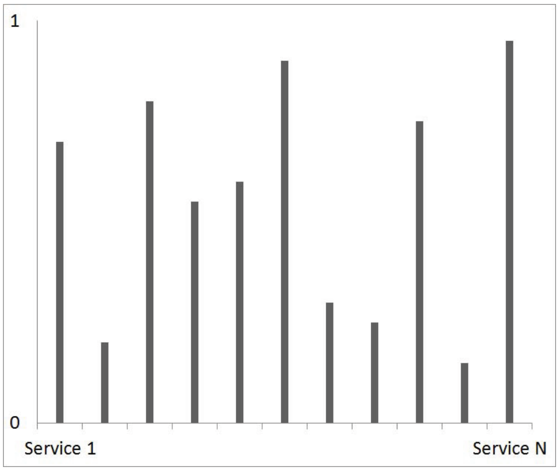


Figure 4.10.: Random Distribution of Expected Scores Among Unordered Compute Services

The example shows the observed random score values of a sample of N compute services. Every service is represented by one score value which is the result of applying an evaluation function to aggregate the assessed meta-data attributes.

It shall be noted that in absence of a distribution function $F(\cdot)$ some common stopping rules cannot calculate a halt. For example, the optimal reservation wage cannot be calculated without knowing the distribution $F(\cdot)$ together with μ and σ.

4.5.3. Stopping Rules

A stopping rule for the assessment of compute services faces a finite horizon problem and will always lead to a stop. When sequentially assessing different compute services, a sequence of n random variables $\mathcal{S}_1, \ldots, \mathcal{S}_n$ is mapped to the observations $s_1, \ldots, s_n$:

$$\mathcal{S}_1 = s_1, \ldots, \mathcal{S}_n = s_n$$

The observed s_i are a number representing a compute service. Potentially, the number of is calculated with an evaluation function $f(\cdot)$ from the scoring method which is applied to the available multi-dimensional meta-data. Thereby, the compute services are represented by number comparable on a ratio scale which is required by the stopping rule. Furthermore, for every assessment of a compute service corresponding to $\mathcal{S}_i$ costs of c_i occur. In addition, for the traversed sample of observations, highest utility can be recalled with a sequence of utility functions $u_i(\cdot)$:

$$u_0, u_1(s_1), u_2(s_1, s_2), \ldots, u_n(s_1, s_2, \ldots, s_n)$$

with $u_i = \max s_i$ being the utility of the highest performing provider according to scoring function $f(\cdot)$ after conducting n service benchmarks. The utility is on a scale defined by $f(\cdot)$ and, hence, not comparable to the costs occuring in the assessment. The following equation shows the expression seeked to be maximized by a stopping rule for costs c_i increasing with each additional step:

$$\max u_n - \sum_i^n c_i$$

Depending on the scale of the scoring function $f(\cdot)$, the absolute utility may change over time. The scoring function can be configured with a local and global scoring mode. The global scoring variant requires to define the first assessed compute service as the reference with score = 1. In the global scoring variant, the absolute utility and scores are maintained. The local scoring variant implicates that all measured compute services are re-scored relative to one another with every additional assessment result. Thereby, the utility and scores can change over iterations.

Standard Stopping Rules

The stopping problem is similar to the class of house selling problems with recall [114]. The observed result of any previously traversed, hence assessed compute service in a sample can be recalled. The distribution of the scoring outcomes retrieved in an assessment are typically unknown. Thus, the problem can be categorized as a finite horizon problem with no information. However, an order can often be derived from previous assessments or publicly available meta-data.

For standard stopping rules, similar problems, such as the house selling or job search problem [114, 117, 121], optimize the utility in relation to the occured costs. Given the utility u_n can be mapped to a cost metric, well-known standard stopping rules for the class of house selling problems with recall could be applied [114, 117]. Besides, a distribution function for the random variables needs to be defined.

However, with the scoring function $f(\cdot)$, the utility and costs remain on different scales and cannot outweigh one another. Generally, mapping assessment results to a monetary value requires a specific mapping function which needs to be identified first. For every decision-maker a assessment result has a different value and a mapping to a monetary value seems an unfeasible task. Additionally, the random distribution of the variables $\mathcal{S}_i$ is not clear and rather non-deterministic.

Therefore, only standard stopping rules focusing on u_n and without the need for a distribution function are considered as applicable stopping rules. The standard stopping rules presented in the following are the *reservation wage rule*, *cut off rule*, *candidate count rule*, *successive non-candidate rule*, and the *odds algorithm*. Nonetheless, it shall be noted that standard stopping rules lack a support for a cost budget.

Reservation Wage Rule

For the reservation wage, a certain expected outcome must be defined based on the global scoring. To define a reservation wage, a certain compute service or comparable system could be used as a reference. Then all scores are determined in relation to the initial score of the reference system. The rule will stop with the first compute service which scores at least as high as the reference score.

The rule is very well suited to find a compute service which satisfies specified requirements. However, for finding the highest scoring compute service, the rule is highly depending on the reference system. A very low scoring reference system may lead to an early stop and the highest scoring compute service is not observed. In contrast, an extremely high scoring reference system will lead to an observation of all compute services and result in maximum costs.

Cut Off Rule

The cut off rule is generally applicable to a wide range of stopping problems. In this case, the parameter m determines how many compute service score observations are skipped before the first upcoming candidate is accepted. A candidate increases u_n with a new highest score $\mathcal{S}_n$.

If the set of compute services to traverse is ordered according to prior information, the highest scoring compute service may be included in the first m observations. Then, the cut off rule would not find another candidate after m observations and generate maximum costs.

Candidate Count Rule

Given the candidate count parameter m, the stopping rule ignores the first m candidates and stops at the $m + 1$-th candidate. A candidate is an observation that increases u_n with its score $\mathcal{S}_n$ being the absolute highest so far.

The candidate count-based rule may struggle with compute service sets that are ordered according to prior information. The probability that the highest ranked compute service is observed under the first candidates depends on the correctness of the order. If the decision-maker provided an order with few errors, the rule traverses all compute services before it stops and thereby generates maximum costs.

Successive Non-Candidate Rule

The successive non-candidate rule seeks for the sequential pattern of at least m non-candidates in between two candidates. The latter candidate leads to the stop and is the final result of the compute service search.

The rule expects candidates to have a certain distance and to be distributed over the set of compute services. In a compute service set ordered according to prior information, the candidates are more likely to be found in the beginning of the observations. Therefore, the rule can lead to a maximum expenditure in an assessment when the distance m turns out to be defined as too large.

Odds Algorithm

For the odds algorithm, a definition of a probability that $\mathcal{S}_i$ is a candidate must be defined for all $i \in [1, N]$. But even with prior information an estimation of the probabilities is impossible.

Generally, however, the probability to discover a candidate with the new highest value can be described as $p_i = 1/i$. Thereby, the discovery of a candidate in the next observation is as likely as any of the previous observation to be the candidate. The probability $1/i$ declines with every further observations which reflects that it becomes less likely that a new candidate is found with fewer observations left.

Other probability setups can thought of as well. For example, $p_i = 1/2$ for all i which expresses that every additional observation has a 50:50-chance to become a candidate.

Similarly, $p_i = 1/N$ is thinkable which defines an identical likelihood for all observations to be a candidate. Then, however, the odds algorithm will observe all $\mathcal{S}_i$ and generate maximum costs.

Stopping Rule for Compute Services Assessments

All of the existing standard stopping rules are unable to explicitly consider cost budgets. Also, part of the standard stopping rules expect the score outcomes to be described by distributions (e.g., uniform, Gaussian). Most importantly, all existing stopping rules ignore priorly available information that indicates an anticipated order of compute services. And, the score outcomes need to be mapped to a currency in order to compensate costs. Therefore, the present work introduces a heuristic-based stopping rule that exploits ordered compute service sets and treats scores and costs in a distinguished manner. In addition, the stopping rule considers a predefined cost budget and determines a duly end of the assessment process.

Based on the assumption that the order of the compute service set is correct, the first assessed compute service would achieve the maximum score and the score decreases with every later observed service. Therefore, the principal of the stopping rule is to continue the assessment process while the observed score values increase. Thereby, the order of the set is proved incorrect and, thus, the assessment must continue to seize the opportunity of finding a service with a new maximum score.

For the trend analysis of the score, a function χ is needed that detects an increase in the overall utility in step n. Function χ_n returns 0 if observation s_n increases the utility by adding a new maximum score:

$$\chi_n = \begin{cases} 1 & u_n(s_n, \ldots, s_n) \leq u_{n-1}(s_1, \ldots, s_{n-1}) \\ 0 & else \end{cases}$$

To relax the stopping decision according to the utility increment detection and to allow for a configuration of the stopping rule, the increment detection function χ is used in a sliding window of size ω. The sliding window comprises the last ω observations $s_{n-\omega}, \ldots, s_n$. The function λ_n reflects the number of missing utility increments (or consecutive assessed scores lower than the current maximum) within the sliding window:

$$\lambda_n = \sum_{i=n-\omega}^{n} \chi_i$$

The parameter ϵ can be declared to enforce a maximum allowed lambda.

Using the defined cost budget c_{max}, a sliding window size of ω, and the permitted number of outstanding consecutive score increments ϵ, the stopping rule ϕ can be defined as a sequence of binary decision functions. Given the observations $s_1, \ldots, s_n$, it is denoted as:

$$\phi = (\phi_0(\epsilon), \phi_1(\epsilon, s_1), \ldots, \phi_n(\epsilon, s_1, \ldots, s_n))$$

with

$$\phi_n(\epsilon, s_1, \ldots, s_n) = \begin{cases} 1 & \neg(\lambda_n \leq \epsilon \wedge \sum_i^n c_i \leq c_{max}) \\ 0 & else \end{cases}$$

where value of 1 leads to a stopping after n observations and a value of 0 to a continuation of the assessment. A halt is triggered by the stopping rule when either the costs surpass the cost budget c_{max} or the number of missing utility increments within the sliding window surpasses ϵ. The probability of stopping after n observations thus is defined as:

$$\psi_n(\mathcal{S}_1, \ldots, \mathcal{S}_n) = \left[\prod_{j=1}^{n-1} (1 - \phi_j(\mathcal{S}_1, \ldots, \mathcal{S}_j))\right] \cdot \phi_n(\mathcal{S}_1, \ldots, \mathcal{S}_n)$$

To serve the reader with a more readable representation, the formal definition of the stopping rule is transformed into an algorithm. In the following, a pseudo code representation of the stopping rule algorithm is presented:

```
Require: $\omega \leftarrow$ window size, $\epsilon \leftarrow$ min. number of increments
Require: $\mathcal{T} \leftarrow$ compute services set, $\mathcal{C} \leftarrow$ assessment costs
Require: $\lambda \leftarrow 0$, $increment \leftarrow false$, $window \leftarrow queue[\omega]$
  for $\mathcal{S}_i \in \mathcal{S}(window), c_i \in \mathcal{C}$ do
    $s_i \leftarrow f(\mathcal{S}_i)$
    if $s_i \geq \max\{s_i \in window\}$ then
      $\lambda \leftarrow \lambda + 1$
    else
      $\lambda \leftarrow \lambda - 1$
    end if
    if $\neg(\lambda \leq \epsilon \wedge \sum_i^n c_i \leq c_{max})$ then
      BREAK
    end if
    $window$.push($s_i$)
  end for
```

4.6. Software Prototype implementing the Scoring Method

The Hoopla software prototype aims at providing the necessary basis for comparisons of compute service meta-data. The prototype is an instantiation of the $(MC^2)^2$ framework which is used for comparisons in the presented compute service assessment framework.

The implementation of Hoopla is divided into one component providing a library that implements the AHP and into a web application which wraps the library and extends it with a graphical user interface to support the comparison process.

Both, the library called AotearoaLib, following the name Aotearoa of an early $(MC^2)^2$ implementation, and the web application Hoopla are presented in the following subsections. First, the library is presented that gives the decision-making basis and, then, the web application is introduced.

AotearoaLib Library

The AotearoaLib library has been created to serve as a basis for later instantiations of concepts in the framework that incorporate decision-making and evaluation. The library is reused in a prototype to instantiate the stopping rule for automated benchmarking which is presented in section 5.4.

AotearoaLib implements the AHP to support criteria hierarchies, pair-wise comparisons, quantitative and qualitative criteria, and diverse index calculations. The library is implemented in Java and provides a programming interface to drive *Decisions*. Also, *Decisions* can be transmuted into *DecisionTemplates* for later reuse. Every *Decision* consists of two ordered lists that contain multiple *Alternatives* and at least one *Goal*. Each *Goal* spans a tree of criteria where a *Goal* has multiple *Criterion* children, again in an ordered list. Also, every *Criterion* can have multiple *Criterion* children. The described entities in the data model of the programming interface and their relations are depicted as a UML class diagram in figure 4.11.

The programming interface furthermore includes the *AnalyticHierarchyProcess* class that implements the evaluation logic of the AHP. A new process is started with providing a *Decision* object in the constructor. The class discloses the method *evaluate* which expects a list of evaluation matrices as parameters. The matrices represent the results of pairwise comparisons between the alternatives regarding the leaf criteria under a goal of the criteria hierarchy. The matrices list must be give in the order of the goals and leaf criteria in the ordered tree of *Criterion* objects. Additionally, the comparison matrices must follow the order of the alternatives in the *Decision* object.

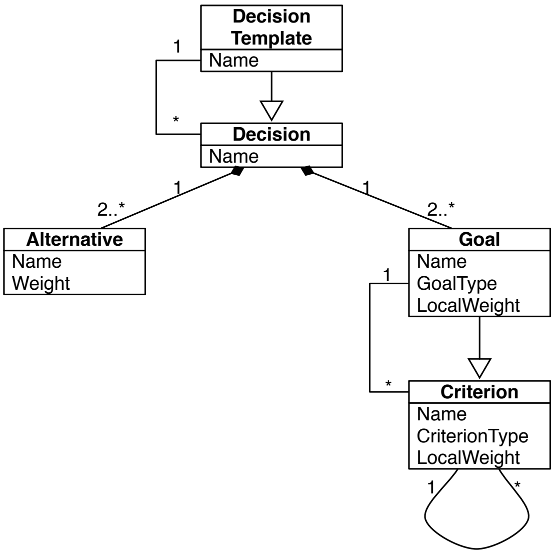


Figure 4.11.: The AotearoaLib Data Model

Moreover, matrices can be defined to set the weights of the criteria. Alternatively, the weights can be defined in the data model. The entities *Alternative*, *Goals*, and *Criterion* possess an attribute *weight* (or *localWeight*) that exposes this functionality.

To allow the use of matrices without manually creating multi-dimensional arrays, the library employs the JAMA package [200] that provides constructs from linear algebra as Java classes. The matrices are not only used as an input parameter for evaluation matrices, but also internally in value normalization using matrix multiplications and to compute the AHP evaluation results in parallel threads.

The implementation of the AHP returns four result maps each mapping alternatives to normalized values. The four result maps comprise the calculated values of all alternatives regarding the positive goals, the negative goals, the additive index, and the multiplicative index.

A code example showing the general usage of the AotearoaLib programming interface is given in section B.1 in the Appendix.

Web Application

The Hoopla additionally provides a web application that comes with a graphical user interface. While the AotearoaLib library allows developers to evaluate alternatives and apply the AHP, the web application exploits its user interface to guide decision-makers through a structured decision process.

An early-stage prototype was developed under the name Aotearoa. The premature version helped to collect requirements from users and evolve the Aotearoa prototype to the Hoopla web application over multiple iterations. After conducting interviews with the industry, the requirements became clear and the Hoopla was developed that extends Aotearoa with support for templates. From the interviews a first template for cloud infrastructure decisions has evolved, which is due to the decision-makers involved in the interviews.

The Hoopla has been developed on the Google AppEngine Platform [201] using the Google Web Toolkit (GWT) [202] to enable a rich graphical user interface. For complex and custom components, smartGWT [203] provides a a library of complex components. With the help of smartGWT, a complex criteria hierarchy input component was developed which allows to define and select criteria and sort them in a hierarchy. The frontend serves a user with a graphical user interface and guides through the decision-making process based on $(MC^2)^2$. The backend incorporates the aotearoaLib to manage data in a data model and to compute evaluation results using the AHP implementation. To persist the data model, the AppEngine datastore service is used. To allow for user accounts in the system, the AppEngine user service is used. Figure 4.12 illustrates the architecture of the AppEngine-based Hoopla web application.

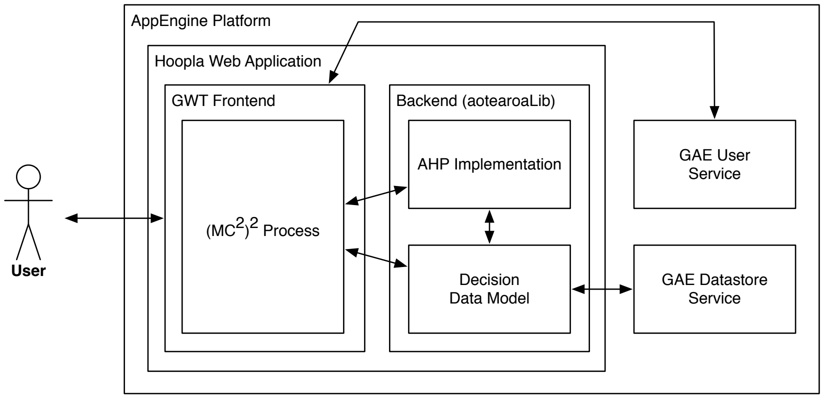
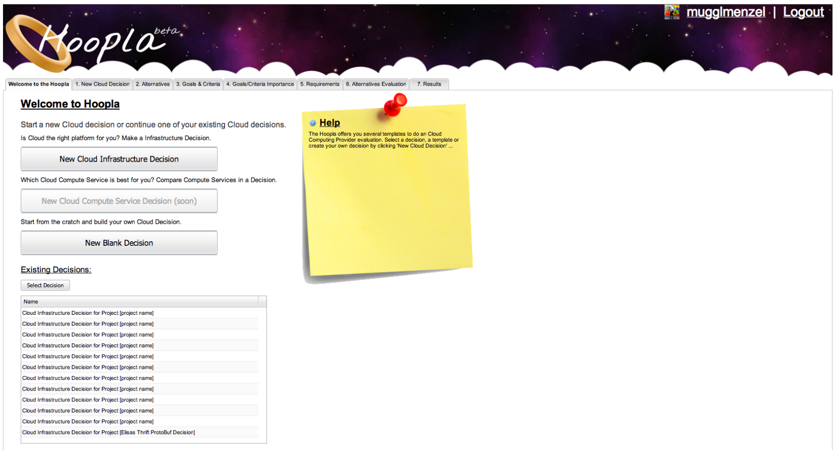


Figure 4.12.: The Hoopla Architecture

The web application reflects the process of $(MC^2)^2$ in its graphical user interface by following a 7-step process. A user first starts a new Decision by assigning a name and description. Then, alternatives are defined in a subsequent step. In step 3, the goals and criteria hierarchy can be created with a custom input component. For each goal and criteria pairing derived from the hierarchy a user can specify the importance according to his preferences in step 4. A slider input component supports a *[-10, 10]*-Likert scale to simplify the preference definition. Optionally, requirements can be set in step 5. Step 6 is the last step that requires a user to input data. The attribute values regarding the defined criteria must be given for all alternatives, before the AHP implementation has enough data to compute values. The final step 7 presents the results for the different indices and shows the calculated weights of the criteria hierarchy. Figure 4.13 depicts a screenshot of the graphical user interface offered by the Hoopla web application.



Welcome to Hoopla

Start a new Cloud decision or continue one of your existing Cloud decisions.

Is Cloud the right platform for you? Make a Infrastructure Decision.

Which Cloud Compute Service is best for you? Compare Compute Services in a Decision.

Start from the cratch and build your own Cloud Decision.

Existing Decisions:

Name
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [project name]
Cloud Infrastructure Decision for Project [Elisas Thrift ProtoBuf Decision]

Help

The Hoopla offers you several templates to do an Cloud Computing Provider evaluation. Select a decision, a template or create your own decision by clicking 'New Cloud Decision' ...

Figure 4.13.: Screenshot of the Hoopla Web Application

The Hoopla web application is publicly accessible via the URL `http://hooplaride.appspot.com`. Only a google or facebook account is needed to login. The source code of the Hoopla is publicly available and maintained as a Github repository under `https://github.com/mugglmenzel/Hoopla`.

In addition to the web application, also as standalone version of the web application has been developed for portable usage. Therefore, the Google AppEngine environment is imitated for the use on a desktop computer. Unix and Windows scripts automate the execution and deployment of the Hoopla on a local AppEngine environment. Thereby, the portable version can be setup and started on most desktop computers with a simple double-click.

4.7. Evaluation of the MADM-based Scoring Method

The MADM-based scoring method is the basis for comparisons of compute services according to meta-data attributes. The proposed method calculates a scoring value with an AHP-based aggregation of meta-data attributes to address research question $RQ1b$:

> *RQ1b. Can MADM support the aggregation of multiple meta-data attributes into a subjective value for a compute service?*

The implemented software prototype (c.f. section 4.6) asserts the feasibility of the scoring method concept. Furthermore, the scoring method approach has been evaluated with decision-makers from the industry in a case study. Interviews with the decision-makers give insights in the applicability of the approach, particularly in terms of perceived effort and trust. The software prototype and the case study have already been published and discussed with peers in the research community [204].

In a comparison of meta-data attributes of multiple compute services, the approach induces a certain computational complexity to determine the score of each compute service. In research question $RQ1c$, the computational complexity is demanded to be examined:

> *RQ1c. How is the computational complexity of using MADM to compare compute services regarding meta-data?*

The evaluation of the scoring method includes an analysis of the computational complexity. Specifically, a formal definition using the big O notation and experiments with the software prototype shall lead to a response to $RQ1c$. Peers in the research community have already involved in a discussion about the computational complexity of the presented AHP-based scoring method and its applicability in practice [205].

This section is divided into two subsections: the first subsection presents the results from interviews conducted with decision-makers in the industry. In the final subsection, an analysis of the computational complexity of the scoring method is driven and findings from experiments to verify the analytic examinations are presented. Particularly, the computational complexity is defined using the big O notation and then examined in experiments concerning the influence of variable factors, such as the number of compute services, criteria, and criteria grouping. Also, insights in the time complexities in practical scenarios with performance meta-data are given.

4.7.1. Case Study with the Industry

To evaluate the suitability of the scoring method for decision-makers, interviews with actual users were to be conducted. Initially, suitable and available participants for a case study needed to be contacted. A research project in collaboration with Telekom Laboratories offered a platform to engage participants from the industry. One of the project's goals was the development of a scoring method in the cloud context.

After a web application (c.f. section 4.6) had been developed as a software prototype, participants were needed which would use the prototype in practice. The web application supports the definition of alternatives, goals & criteria hierarchies, and allows users to weight goals and criteria with Likert-scale sliders in pair-wise comparisons. From the given parameters, the AHP is followed by the software. With manually provided attribute values, a scoring is calculated and presented to the decision-maker. The partner *Telekom Laboratories* invited suitable decision-makers from T-Systems, an international IT and communications technology provider, for participation. The web application was announced as prospective new software to support consulting projects with customers.

The selection process for the study aimed for experts in favor of quantity. Consequently, the case study was conducted with two consultants from T-Systems who have many years of experience with consulting customers regarding IT infrastructure. Both participants invested several man-hours per week over several months to test the software prototype and participate in the study.

The participants have used the MADM web application to develop a scoring in an infrastructure decision with the alternatives *on-premise data center* and *T-Systems infrastructure services*. The case study resembles a frequent decision faced by the participants in consultancy projects with customers of T-Systems. The two participants developed a goals & criteria hierarchy and applied a weighting. Also, attribute values originating from active projects were fed to the web application to gain a scoring.

After the participants had finished testing the tool over several weeks, a qualitative survey was conducted to gain insights in the perceived quality of the approach. A structured feedback form (c.f. chapter C in the appendix) prepared to conduct the survey was discarded in favor of in-depth individual interviews [206]. The individual semi-structured and open interviews were carried out via telephone in 1-2 hour discussions. The topics of the semi-structured interviews are based on the structured feedback form. Particularly, the participants were asked for open feedback in the following resorts:

Transparency How transparent and comprehensible is the scoring process?

Effort/Speed How much time is consumed to arrive at a scoring?

Trust How much do you trust the correctness of the presented score?

Both participants were confronted with the resorts and the according question. In their response, they were free to describe their experiences with the tool and particularly pros and cons. Also, a comparison to the currently active decision-making or scoring process was a suggested part of the feedback.

The responses in the individual interviews after a case study with participants from the industry have been captured as written protocols. The following list represents an aggregation of the feedback provided by the case study participants:

Transparency Albeit the feature being yet absent in the software prototype, the approach promises to allow a precise documentation of a decision. The participants emphasized the possibility to recall the weight factors determined by a decision-maker to be an important characteristic. It allows to argument the reason of a scoring outcome and to drive a sensitivity analysis in iterations. However, the weights defined by the decision-maker should be presented in a comprehensible form. The general transparency is perceived higher than in a decision without tool support. The possibility to compare scores on a ratio scale has comforted the participants in their decision-making.

Effort/Speed The overall effort of scoring several alternatives is acceptable from the perspective of the participants. Over the case study, the participants learned that the criteria hierarchy heavily influences the later effort for a weighting of the criteria. The number of criteria and grouping in the hierarchy are factors to consider, when it comes to a reduction of the weighting effort. Also, filling out all attribute values manually can become unbearable. The participants see a need to integrate data sources to feed attribute values automatically into the system.

Trust The participants explained that since the approach has a scientific background and has been discussed in diverse works in the mathematical field, a trust in the correctness of the score is given. The participants expect a decision-maker to trust the criteria hierarchy (and set of criteria) if both have been prepared by experts.

Over the in-depth interview, the participants stated that the approach is applicable to find a scoring for alternatives and to support funded decisions. They expect the approach to be used in addition to the current decision-making in industry projects. Primarily, the approach supports the currently made decisions with a number-oriented perspective. A Web application software prototype helps to explore past scorings and, thereby, allows a decision-maker to use the tool for documentation purposes.

In summary, research question $RQ1b$ has partly been explored with decision-makers from the industry in a case study. Although the case study focuses on IT infrastructure decisions, insights in respect of the applicability of the MADM-based approach could be gained. The overall results from in-depth interviews with industry experts indicate that the approach provides a support to carry out a structured and funded scoring of alternatives and facilitates comparisons. Given the participants are trusted industry experts, the statements and results of the case study are significant and representative to fund the practical applicability of MADM-based scorings.

A scoring of compute services has not been part of the case study. However, the general applicability of the scoring using a Web application interface has been shown. A scoring of compute service according to meta-data attributes is structured similarly. The set of criteria is predetermined by the set of available meta-data attributes, and the alternatives are defined by the set of available compute services. The following subsection, which examines the computational complexity of the approach, applies the approach particularly to compute services. Consequently, the complexity examination is proof that a scoring can be applied to compute services, too.

4.7.2. Computational Complexity

In the light of $RQ1c$, the computational complexity has been explored using the Java library developed for MADM scoring (c.f. section 4.6). The library implements the AHP in a multi-threaded manner and exposes a data model to define custom scoring functions. In the implementation, the row/column sum and the Saaty normalization methods were employed with an enforced 5-digit precision. To accommodate the data models and temporary results, a Java heap space of 4GB RAM was granted.

Using the library, several experiments aim to verify the defined $O(\cdot)$ and seek to give insights into the behavior of the scoring method for complex scoring applications. Every experiment has been repeated 20 times to ensure stable results. Final values are represented by the average of the aggregated 20 measurements. For the computational complexity experiments, diverse decision models have been defined. An overview of the decision models is given in table 4.4.

The first decisions model, referred to as DM-GS, aims to explore the behavior for a growing number of compute service alternatives in a scoring. The model comprises a flat, one-level criteria hierarchy (one criteria group) with a fixed number of 100 criteria. All criteria are weighted equally and the attribute values of the compute services are generated synthetically from random values $\in (0, 1000]$ to resemble benchmarks. The number of compute service alternatives is variable.

The second decision model, referred to as DM-GC, aims at exploring the time complexity behavior for a growing number of criteria. The decision model comprises a fixed set of 100 virtual compute services. The attribute values are again generated randomly from values $\in (0, 1000]$ to resemble benchmarks. The criteria hierarchy has a flat, single level structure and the number of criteria is variable.

The DM-VG decision model focuses on the structure of the criteria hierarchy and the impact on the computational complexity. Unlike the decicions models DM-GS and DM-GC, a fixed number of 1000 criteria and 100 compute services are defined in the decision model. The attribute values are generated synthetically with random values $\in (0, 1000]$. The criteria hierarchy structure changes over the experiments.

Decision Model	# of Criteria	# of Compute Services	# of Criteria Groups
DM-GS	100	variable	1
DM-GC	variable	100	1
DM-VG	1000	100	variable

Table 4.4.: Decision Models used for Experiments

The first set of experiments explores the computational complexity in regards of the compute service set and attributes set involved in the scoring. A second set of experiments researches the computational complexity concerning the grouping of criteria in a criteria hierarchy. Finally, results from experiments testing the time consumption to compute a scoring for realistic numbers of compute services and attributes are presented.

General Computational Complexity

The computational complexity of the MADM-based scoring method is related to the number of scored services in S, number of requirements in R, as well as the number of criteria (with linked attributes) in A. According to the big O notation, the $O(\cdot)$ of the AHP-based scoring method can be described as following:

$$O\left(\underbrace{|S| * |R|}_{\text{requirements check}} + \underbrace{\binom{|A|}{2}}_{\text{weights normalization}} + \underbrace{|A| * \binom{|S|}{2}}_{\text{value normalization}} + \underbrace{|A| * |S|}_{\text{score calculation}} \right)$$

The $O(\cdot)$ resembles the theoretical upper bound, whereas in practice, depending on the implementation, the computational complexity can be lower than the O function. Specifically, as only a range of subsets of attributes must be normalized. When attributes are grouped within the hierarchy, only attributes under the same parent criterion or goal need to be compared.

Using the software library, the general computational complexity could be explored in experiments based on the decision models DM-GS and DM-GC. For the decision model DM-GS, the number of compute service alternatives is increased by additional 100 virtual compute services in each iteration. In contrast, the decision model DM-GC fixes the number of compute service alternatives to 100 and increases the number of criteria. In each iteration, 100 additional criteria and linked attribute values are introduced to the criteria hierarchy. A requirements check is discarded in both models as it is optional in a scoring and would diminish the number of compute services.

A plot of the growing computation time for the decision model DM-GS with the row/-sum method is depicted in figure 4.14, and with the Saaty normalization method in figure 4.15. The graph plots the computation times to create the decision model and to compute the scoring of the compute services. On the horizontal axis, the number of compute services in the decision model is shown. The left vertical axis represents the milliseconds of computation time, the right vertical axis the computational steps calculated with $O(\cdot)$. The black line in the graph shows the expected computational complexity function $O(\cdot)$.

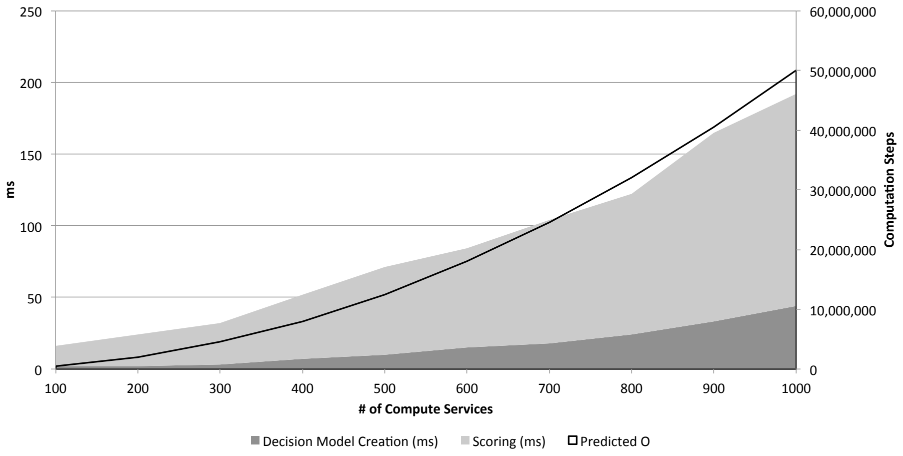


Figure 4.14.: Computational Complexity with 100 Criteria and Growing Number of Services (Row/Sum Method)

Similarly, figures 4.16 and 4.17 depict the experiment results plots for the DM-GC with row/sum and Saaty normalization methods. The metric on the horizontal axis is the number of criteria, on the vertical axes milliseconds and computation steps, respectively. The black line indicates the predicted complexity by means of the $O(\cdot)$ function.

The results show that a growing number of compute services and criteria affects the computational complexity of the scoring method. Also, the size of the compute services set has a bigger influence than the size of the criteria set. The results gained with experiments based on the decision models DM-GC and DM-GS confirm the prior complexity predictions manifested in $O(\cdot)$.

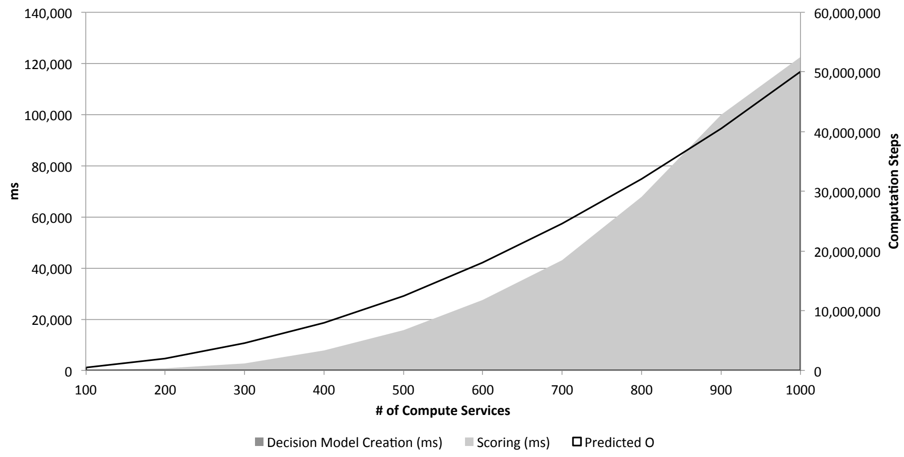


Figure 4.15.: Computational Complexity with 100 Criteria and Growing Number of Services (Saaty Method)

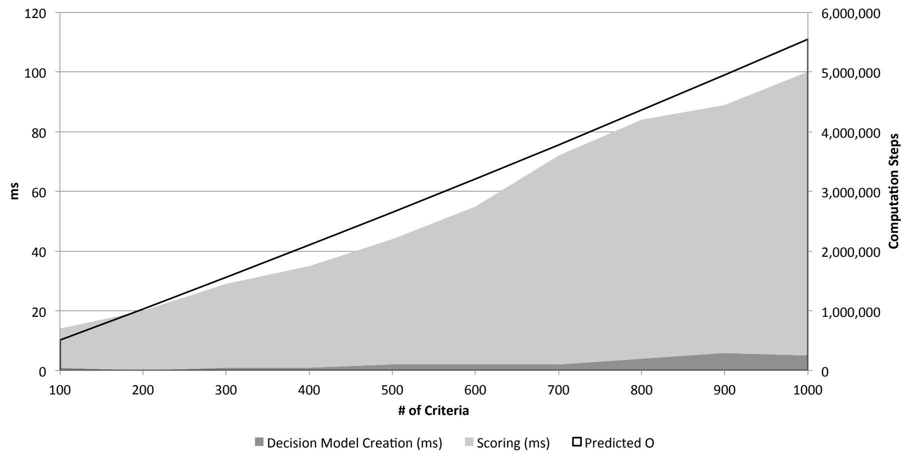


Figure 4.16.: Computational Complexity with 100 Compute Services and Growing Number of Criteria (Row/Sum Method)

Effect of Criteria Grouping

Apart from the set of compute services and the set of attributes that affect the computational complexity, the structure of the criteria hierarchy may have an influence on the computational complexity, too.

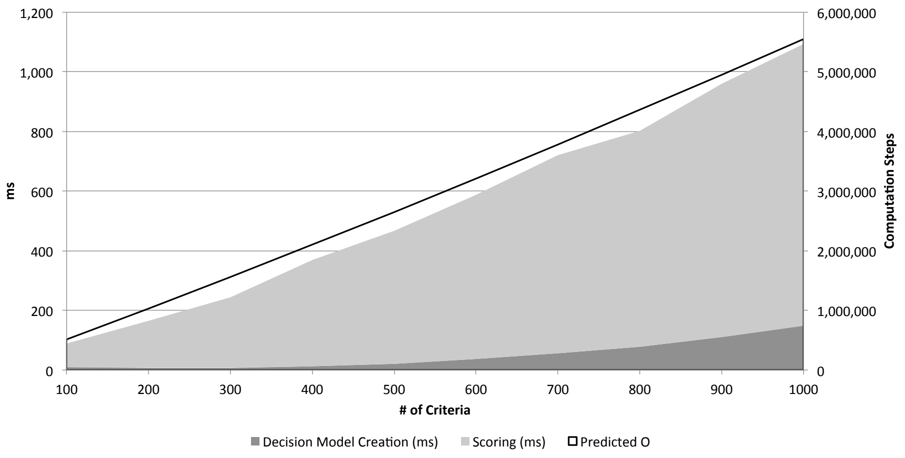


Figure 4.17.: Computational Complexity with 100 Compute Services and Growing Number of Criteria (Saaty Method)

A closer analysis of the pair-wise comparisons involved in a score calculation reveals the influence of the criteria hierarchy and particularly grouping of criteria. While a single-group hierarchy implicates $\binom{|A|}{2}$ pair-wise comparisons during a normalization, a hierarchy with g groups leads to pair-wise comparisons described by the following expression:

$$\sum_{i=1}^{g} \binom{|A_i|}{2} + \binom{g}{2} \text{with} \bigcup_{i=0}^{g} A_i = A$$

With g criteria groups, the $O(\cdot)$ needs to be adjusted to the following:

$$O\Big(\underbrace{|S| * |R|}_{\text{requirements check}} + \underbrace{(\sum_{i=1}^{g} \binom{|A_i|}{2} + \binom{g}{2})}_{\text{weights normalization}} + \underbrace{|A| * \binom{|S|}{2}}_{\text{value normalization}} + \underbrace{|A| * |S|}_{\text{score calculation}} \Big)$$

To explore the effect of criteria groups, experiments were conducted that stress the software library with different hierarchy structures. A static number of compute services and attributes in the experiments permits to observe only the effect of criteria groups.

For the experiments, the decision model DM-VG provides the basis with a set of 100 compute services and 1000 criteria. The row/sum method is used for normalizations. The experiments distinguish in their focus on the criteria hierarchy structure from the earlier experiments. Particularly, the effect of grouping criteria in subsets of diverse sizes under a goal in the criteria hierarchy is explored.

In each iteration, the number of criteria groups and, thereby, the number of criteria within a group is altered. Thereby, the number of pair-wise comparisons to normalize the attribute values linked to criteria in a group is affected.

Figure 4.18 depicts the results from the experiments. The horizontal axes reflect the growing number of criteria within a group and the total number of criteria groups in the hierarchy, respectively. The metrics used on the vertical axes are *milliseconds* for the measured computation times and *computation steps* for the predicted O.

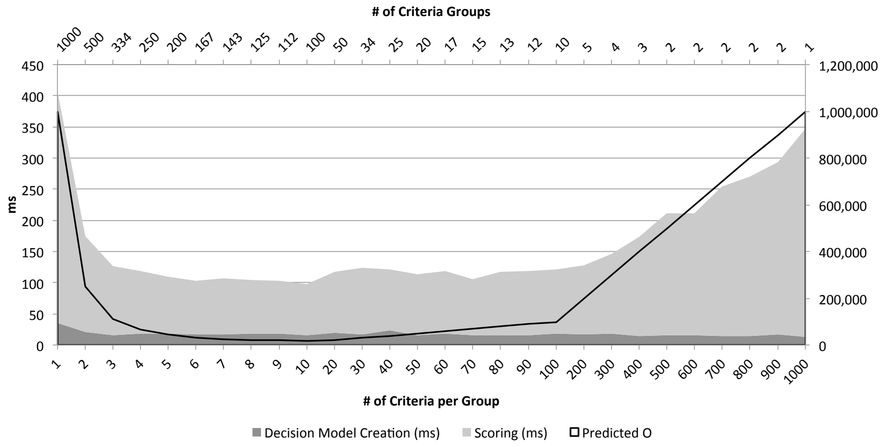


Figure 4.18.: Computational Complexity with 100 Compute Services, 1000 Criteria and Growing Group Size

The experiment results shed light on the effect of criteria groups. As the results show, a sweet spot exists for the criteria group sizing. Also, the most extreme settings with one group for all criteria or a single group per criterion have a maximum negative effect on the computational complexity.

Computation Times in Practice

Aside the experiments concerning the computational complexity, an experiment to acquire computation times for realistic scenarios has been carried out. Using the decision model DM-GC as a basis, this experiment measures the computation times for varying number of criteria and compute services. The relevant scenarios selected for the measurements are combinations of compute services and criteria sets with 5, 10, and 20 items. The numbers align with the sizes of CloudHarmony Summary Benchmark sets

[139] and the OpenBenchmarking.org test suites [207] and, thus, resemble exemplary but realistic scenarios of performance meta-data comparisons.

As observed in first test runs of the experiment, high variations occur due to small computation times of under 1 ms and overhead from java classes used in the software library. Therefore, the number of repetitions has been increased to 10,000 to guarantee reliable results.

Table 4.5 lists the observed computation times for realistic criteria and compute service set sizes. In addition, table 4.6 lists the results for the Saaty normalization method instead of the row/sum method. The computation times comprise the creation of the decision model and the evaluation using the AHP implementation in the software library.

Criteria/Compute Services	5	10	20	30
5	0.46 ms	0.51 ms	0.53 ms	0.80 ms
10	0.99 ms	0.97 ms	1.14 ms	1.26 ms
20	1.58 ms	1.76 ms	2.25 ms	2.13 ms
30	1.74 ms	1.91 ms	2.21 ms	2.61 ms

Table 4.5.: Computation Times in 4 Realistic Scenarios (Row/Sum Method)

Criteria/Compute Services	5	10	20	30
5	0.43 ms	0.49 ms	0.58 ms	0.83 ms
10	0.83 ms	0.92 ms	1.13 ms	1.47 ms
20	1.47 ms	1.64 ms	1.97 ms	2.63 ms
30	1.53 ms	1.86 ms	2.49 ms	3.60 ms

Table 4.6.: Computation Times in 4 Realistic Scenarios (Saaty Method)

As the results in tables 4.5 and 4.6 reveal, the computation times in realistic scenarios with 5-30 compute services and criteria are rather neglectable. Even with the more complex Saaty normalization method, times do not exceed 3.6 ms for realistic scenarios. In a comparison of compute services according to diverse meta-data attributes, a scoring can be gained almost instantaneously.

Part III.

Instantiation

5. Assessments of Compute Service Performance at Run-time

In this chapter, an approach for assessing and comparing cloud compute services according to performance is introduced. The approach describes how performance benchmarks can be applied to capture performance attributes of diverse compute services.

Moreover, the scoring method is employed with the Analytic Hierarchy Process to aggregate multiple performance attributes into a score to compare compute services. The scoring method is further used to build a utility function in a stopping rule. Thereby, measurements can be conducted under a cost budget in a stopping rule approach. Figure 5.1 summarizes the models and methods used from the framework.

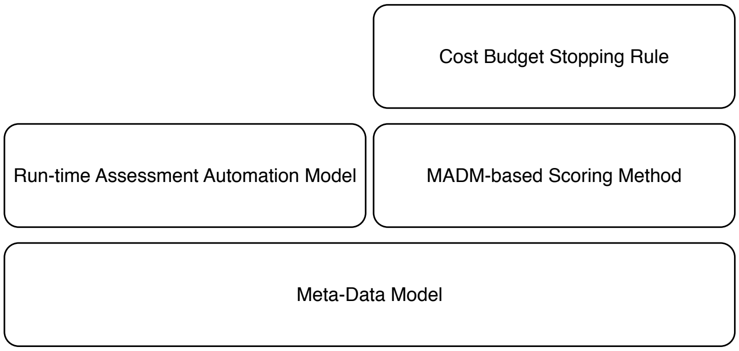

Cost Budget Stopping Rule
Run-time Assessment Automation Model
MADM-based Scoring Method
Meta-Data Model

Figure 5.1.: The Framework's Models & Methods used for Performance Assessment

The diverse contributions regarding run-time assessments of compute services' performance presented in this chapter are the following:

Automated Performance Benchmarking

The concept of automated performance benchmarking of compute services with repetitions and scheduling orients on the framework's automation model and extends the framework's meta-data model for its purposes. Particularly, an extension of the compute service assessment framework's meta-data model is introduced to attach performance meta-data, gained from benchmarks, to compute services. To realize an automation of performance benchmarks, the automation model of the framework is extended and adapted to execute benchmarking software as assessment task. Furthermore, user parameters defined in the Roadmap Definition Language provide means to define repetitions and to schedule start times for automated performance measurement runs.

Scoring & Comparisons

Assessed performance attributes allow for comparisons between competing compute services. Using the benchmarking results from the extended meta-data model, compute services can be compared in multiple dimensions of benchmarks, e.g., concerning the result of a LINPACK or a Whetstone benchmark. A single-dimensional value on a ratio scale comforts comparisons and a sorting of compute services according to the entirety of benchmarking results captured. The framework's scoring method using MADM, specifically the AHP, is employed to realize a mapping of multiple benchmarking results to a comparable, single-dimensional score.

Stopping Rule

Besides, costs accrue by leasing compute service resources over the time span of a performance benchmarking. For performance measurements, the framework's stopping rule is configured to enforce a cost budget and exploit priorly known hardware figures of compute services.

This chapter is structured as follows: first, a method to automate, schedule and repeat compute service performance benchmarking is introduced in section 5.1. Then, section 5.2 explains how the framework's scoring method can be employed using the meta-data gained from the measurement method. Besides, an adaption of the framework's stopping rule with cost budget is presented in section 5.3.

Finally, a software prototype implementing the automated performance measurement method and the stopping rules is introduced in section 5.4 to proof the feasibility of the concepts. Using the software prototype, evaluations of the automated performance benchmarking method and the stopping rule are carried out in sections 5.5 and 5.6. Experiments validate the capabilities of the automated performance benchmarking method and explore its promptitude. An evaluation of the stopping rule proofs its ability to enforce a cost budget and shows its advantages over standard stopping rules.

5.1. Automated Performance Benchmarking of Compute Services

The assessment of a compute service's performance can be achieved in distinct ways. The performance of a system can be analyzed in a simulation and emulation, or can be measured in-situ and with benchmarking (c.f. section 2.2.1). For compute services, generating a system model appears an unnecessary act generating overhead since the real system, the compute service, can be accessed via APIs with small effort.

In regards of workloads, compute services can be monitored and measured under real workloads (in-situ) or with synthetic workloads with benchmarking. To gain in-situ results, a software application with a monitoring component would be needed that is stressed with workloads from users or a workload generator. A benchmarking approach generates a synthetic workload with small software programs without preparation efforts. Measurements with benchmarking, hence, can be conducted with less effort and are the approach pursued in this work.

The basis for an assessment of performance attributes of compute services is the novel automated performance benchmarking method presented in this section. For that matter, particularly the aspect of heterogeneity in compute service performance quality is of interest. To address that aspect, scheduling of performance measurements in regards of time frames and frequency is incorporated in the method.

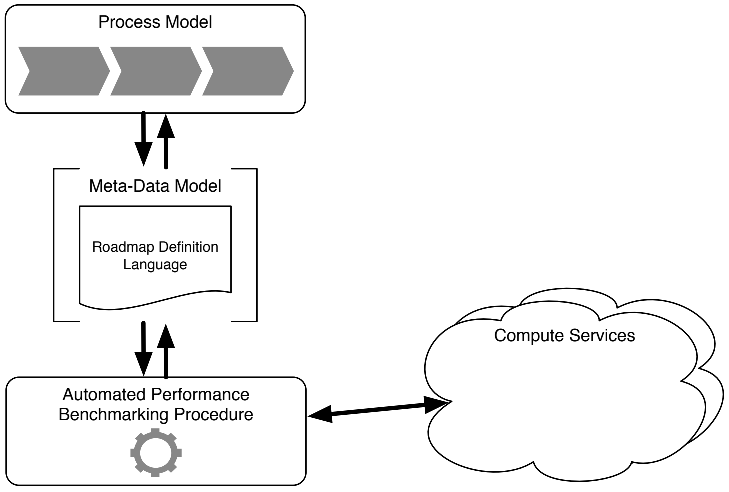

Process Model
Meta-Data Model
Roadmap Definition Language
Compute Services
Automated Performance Benchmarking Procedure

Figure 5.2.: Overview of the Automated Performance Benchmarking Method

As part of the method, a procedure to execute automated measurements is presented which builds upon the automation model of the assessment framework (c.f. section

4.3). Furthermore, a Roadmap Definition Language is introduced to describe performance measurement schedules and repetitions. Besides, an extension of the meta-data model to persist measurement results and roadmap definitions is described. Moreover, a process model that leads consumers through the planning and execution of assessment with the method is introduced.

From the process model a consumer defines parameters expressed in a roadmap definition in the data model and hands it to the automated benchmarking procedure. The procedure considers the parameter and conducts benchmarking runs of the selected compute services accordingly. The results are then persisted in the data model and become available to the consumer. Figure 5.2 depicts the parts of the method and their interactions.

This section is structured as follows. First, the procedure, which bases on the framework's automation model, is explained. Then, with the procedure in mind the aspects of repetition and result aggregation are discussed and the Roadmap Definition Language is introduced. Following, the extended meta-data model is presented which, aside from benchmarking results, can hold definitions in the priorly introduced Roadmap Definition Language. The process model then guides a consumer in his interactions with the procedure and data model within the method. Finally, diverse aspects of the approach, such as costs and limitations, are discussed.

5.1.1. Automated Performance Benchmarking Procedure

While performance benchmarks have already been adapted for compute services (c.f. chapter 3), a method to automate and aggregate compute service measurements over time is novel. An automation of performance measurements calls for a procedure that incorporates an execution of benchmarks on compute services without user interaction. This subsection presents a procedure that builds upon the blueprint automation model contained in the framework presented in section 4.3.

The original procedure blueprint must be extended to meet a range of requirements to fit the context of automated compute service measurements. For measuring multiple compute service targets regarding diverse characteristics with various schedules, a way to parameterize the measurement procedure must be in place. Initially, the set of compute services and benchmarks to consider for measurements need to be defined. Furthermore, to level out heterogeneity in a compute service's hardware and average multiple measurements, a schedule of multiple measurement phases must be definable. As past measurements verified, only a range of measurements over multiple time spans can guarantee enough data to gain meaningful results.

Multiple components are involved in the procedure of measuring compute services with performance benchmarks. Figure 5.3 illustrates the components involved in the procedure of the automated performance measurement approach.

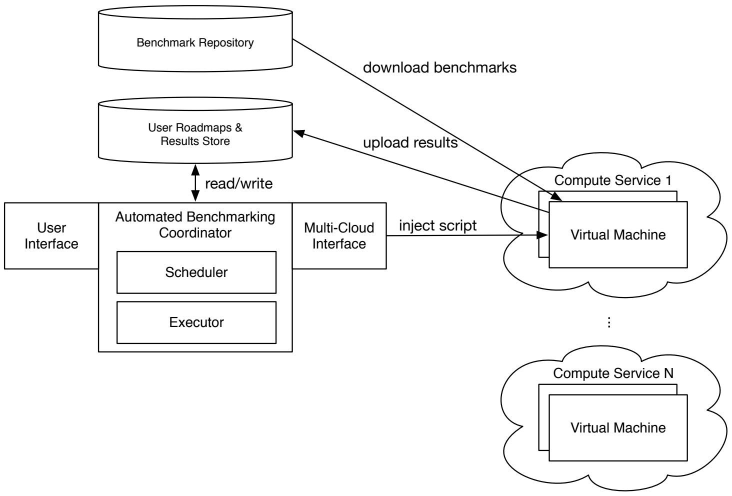


Figure 5.3.: Overview of the Automated Performance Measurement Procedure

The main component for automated measurements, the *Automatic Benchmarking Coordinator*, coordinates benchmarks according to user preferences using a multi-cloud interface. With an included *Scheduler* component, performance benchmarks can be repeated over time given scheduling rules. Executable benchmarking software is maintained in a *Benchmark Repository* that is accessible publicly and, therefore, also from any compute service.

In order to define parameters, a user interface allows to define roadmaps, which schedules measurements of a selection of compute services with a set of benchmarking software. Roadmaps as well as results can be stored and retrieved from the *User Roadmaps & Results Store*.

Given the roadmaps defined by a user, the main component coordinates a measurement triggered by the scheduler. Then, by using the *Multi-Cloud Interface* component, a virtual machine at each selected compute service is instantiated. Access to the machine via remote protocols like SSH enables an injection of scripts that download and execute the set of benchmarking software. Finally, the results of a benchmarking run are uploaded to the *Results Store*.

The data model to store user preferences and results in the *User Roadmaps & Results Store* is introduced in section 5.1.3. The process model, which describes the interaction of the user and the components in detail, is presented in section 5.1.4.

5.1.2. Repetitions and Benchmarking Result Aggregation

The procedure introduced in the previous section provides means to automatically conduct performance measurements and, thereby, assess compute service performance meta-data. Such measurements can even be collected over long time spans using roadmaps. With repetitions of a measurement, the confidence that the measured value is accurate can be assured. In compute services, a variability between multiple measurements is expected due to a non-deterministic behavior caused by environmental influences, such hardware upgrades or varying simultaneous workload from multiple consumers [13, 80]. One particular influence is the fact that a VM provided by the service shares physical resources with other consumer's VMs . Although the virtualization layer aims to allocate resources to certain VMs, an interference can often not be eliminated [30].

The influences can change over time, depending on the resource usage of other consumers. During periods of high demand, resources may see a high utilization by consumers. For example, web servers on compute services may be accessed more frequently in the evening, or servers hosting online shops are facing high workloads in the month before holidays.

The following subsections address the need to repeat performance measurements over time and present functions to aggregate multiple measurements. First, the roadmap definition language is introduced to allow consumers a definition of benchmarking schedules. Furthermore, the aspects of variability, repetitions and the aggregation of multiple benchmarking results are discussed.

Roadmap Definition Language

The present approach introduces roadmaps to capture external influences over time. While measurements can be spread and repeated over months or years, variations in performance benchmarking results can also occur within short time spans. Therefore, the approach supports a definition of repetitions on different levels of a roadmap. Figure 5.4 illustrates the levels of a roadmap that allow for repetitions.

While the two upper levels aim to cover repetitions over years, months and days, the lower level aims at subsequential repetitions within minutes or seconds. The illustrated structure reflects the components of a roadmap. Within the approach, a roadmap is an entity that contains all information to define repetitions of performance measurements.

To define a roadmap, multiple roadmap entries can be defined in a scheduling language called roadmap definition language. The language includes concepts to define repetitions on all levels, and to reference previously defined benchmarking sets

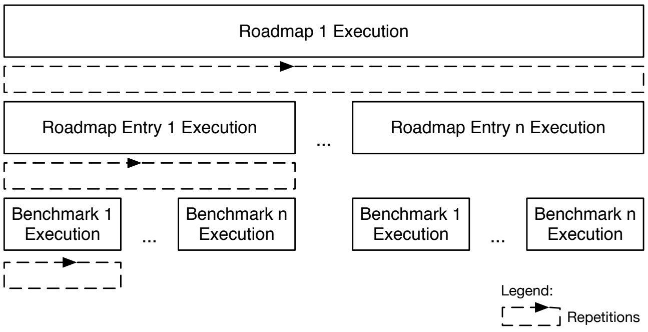


Figure 5.4.: Available Levels to Induce Repetitions

The syntax of the roadmap definition language is a derivation of the format used in the Unix cron job scheduler. The grammar of the roadmap definition language can be expressed in Backus-Naur-Form as follows:

$$
\begin{aligned}
\langle\text{roadmap entry}\rangle &\models \langle\text{time}\rangle\ \langle\text{days}\rangle\ \langle\text{months}\rangle\ \langle\text{repeat}\rangle\ \langle\text{identifiers}\rangle\ \langle\text{metrics}\rangle \\
\\
\langle\text{time}\rangle &\models \langle\text{minutes}\rangle\ \langle\text{hours}\rangle \\
\langle\text{days}\rangle &\models \langle\text{days of month}\rangle\ \langle\text{days of week}\rangle \\
\langle\text{months}\rangle &\models * \mid */\langle\text{month}\rangle \mid \langle\text{month}\rangle \\
\langle\text{repeat}\rangle &\models \langle\text{repeat entry}\rangle\ \langle\text{repeat benchmarks}\rangle \\
\langle\text{identifiers}\rangle &\models \langle\text{benchmark set}\rangle\ \langle\text{CS set}\rangle \\
\langle\text{metrics}\rangle &\models \langle\text{aggregation function}\rangle\ \langle\text{variation metric}\rangle \\
\\
\langle\text{minutes}\rangle &\models * \mid */\langle\text{minute}\rangle \mid \langle\text{minute}\rangle \\
\langle\text{hours}\rangle &\models * \mid */\langle\text{hour}\rangle \mid \langle\text{hour}\rangle \\
\langle\text{days of month}\rangle &\models * \mid */\langle\text{day of month}\rangle \mid \langle\text{day of month}\rangle \\
\langle\text{days of week}\rangle &\models * \mid */\langle\text{day of week}\rangle \mid \langle\text{day of week}\rangle \\
\langle\text{benchmark set}\rangle &\models \textit{benchmark set identifier} \\
\langle\text{CS set}\rangle &\models \textit{compute service set identifier} \\
\langle\text{aggregation function}\rangle &\models \text{harmonic} \mid \text{geometric} \mid \text{max} \mid \text{min} \\
\langle\text{variation metric}\rangle &\models \text{standard deviation} \mid \text{Coefficient of Variation} \\
\\
\langle\text{minute}\rangle &\models 0 \ldots 59 \\
\langle\text{hour}\rangle &\models 0 \ldots 23 \\
\langle\text{day of month}\rangle &\models 1 \ldots 31 \\
\langle\text{day of week}\rangle &\models 0 \ldots 6 \mid \text{Mon} \ldots \text{Sun} \\
\langle\text{month}\rangle &\models 1 \ldots 12 \\
\langle\text{repeat entry}\rangle &\models 0 \ldots 99 \\
\langle\text{repeat benchmarks}\rangle &\models 0 \ldots 99
\end{aligned}
$$

Like the cron syntax, the roadmap definition language incorporates the use of wildcards (*) to facilitate scheduled repetitions. Every roadmap entry can have a fixed number or a wildcard for the minutes, hours, day of month (or day of week), and months parameter. A fixed number leads to an execution at that specific point. For example, when setting the first parameters as "1 1 1 1 1" the scheduler will run the benchmarks on 01:01h at 1st January if it is a Tuesday. With wildcards, the execution is processed for every value of that parameter. For example, when setting the first parameters as "* * * * *" the scheduler will run the benchmarks every minute of every hour at every day of every month at any weekday.

Additionally, the number of repetitions for benchmark execution and entry execution can be defined. The number of benchmark repetitions defines how often each benchmark in a benchmark set is repeated during the execution. The number of entry repetitions determines how many consecutive executions of the roadmap entry are triggered.

Variability, Repetitions and Aggregation

To gain representative data using roadmaps, the defined number of repetitions needs to be sufficing. At the same time, the more measurements are made the higher the resource costs and efforts. This leads to the questions how many measurements must be made at least.

The variability of a system's performance behavior can be expressed by the variance of a measured sample, represented by s^2. If the distribution of the performance variability is known beforehand, the number of necessary measurements to guarantee a certain accuracy can be determined. Given a system's performance variability with mean μ and variance s^2 in a normal distribution, the minimal sample size n can be calculated in regards of the required confidence [31].

The demanded accuracy must be defined as r in percent, e.g., 5%. Additionally, an allowed chance of error α must be defined as the confidence level $100(1-\alpha)$. Given α, the confidence interval is determined by the allowed chance of error in the left part of equation 5.1. The z-score of a $100(1-\alpha)$ confidence level can be found by referring to z-score tables. Given the required r, the according confidence interval can be calculated using the right part of equation 5.1.

$$\overline{x} \pm z_{100(1-\alpha)} \frac{s}{\sqrt{n}} = \overline{x}(1 \pm \frac{r}{100}) \tag{5.1}$$

Equation 5.1 is the basis to determine the minimal size of a sample n. To achieve a certain accuracy r and chance of error α, equation 5.1 can be transformed into equation 5.2.

$$n = (\frac{100 s z_{100(1-\alpha)}}{r\overline{x}})^2 \tag{5.2}$$

Without prior knowledge of a system's performance variability, the minimal size of the sample can only be determined iteratively. Then, benchmarking may continue until the results undercut a certain boundary of the sample standard deviation (s), e.g., 0.03. A s-boundary has the advantage that it can be formulated in the same metric as the mean.

Aside from s boundaries, limits can also be expressed for the Coefficient of Variation (CV). The CV is dimensionless and allows to express a boundary independently from the metric used in a measurement. Equation 5.3 defines the CV regarding the mean μ and standard deviation σ (or s for samples).

$$CV = \frac{\sigma}{\mu} \tag{5.3}$$

Independently from requirements in terms of accuracy, there should always be a fixed maximum of repetitions to restrain effort and resource costs. If the variability of the results pertain, further benchmarking will not improve the accuracy and measurements can be stopped. The variability of the system must be accepted and captured using the standard deviation s or the CV.

The set of benchmark results originating from multiple repetitions, however, needs to be converted into a single value. As results from repetitions are all expressed in the identical metric, an aggregated value can be drawn from averaging functions. For that, the geometric and harmonic means, amongst others, can be applied.

5.1.3. Data Model for Roadmap Scheduling and Measurement Results

In order to retain gathered benchmarking results and to define benchmarking tasks, the framework's meta-data model is refined to the needs. The data model defines which data is relevant to the automated benchmarking procedure. Also, it describes the structure to guarantee a uniform access to data for the components involved in the automated benchmarking procedure. A UML class diagram representation of the data model is depicted in figure 5.5.

To embrace the definition of benchmarking tasks, the data model incorporates entities of the type *Roadmap* and the related type *Roadmap Entry*. A roadmap comprises a set of roadmap entries and is owned by a *User*. Every entry consists of a *Benchmark Set* and a *Compute Service Set* which are combined with a *Schedule*.

Benchmark and compute service sets can be referenced with an identifier. The identifier simplifies the process of defining roadmap entries via the user interface component. Using the identifiers, a roadmap entry can be composed from a compute service set and a benchmark set identifier, and a schedule expressed in a text format (see section 5.1.4). Consequently, a roadmap entry defines at which time certain compute services must be measured with benchmarking software.

A set of benchmarks is described by an identifier and comprises a list of benchmarking software represented by *Benchmark* entities. Each benchmark entity has a name and a metric. Similarly, a set of compute services has an identifier and incorporates a list

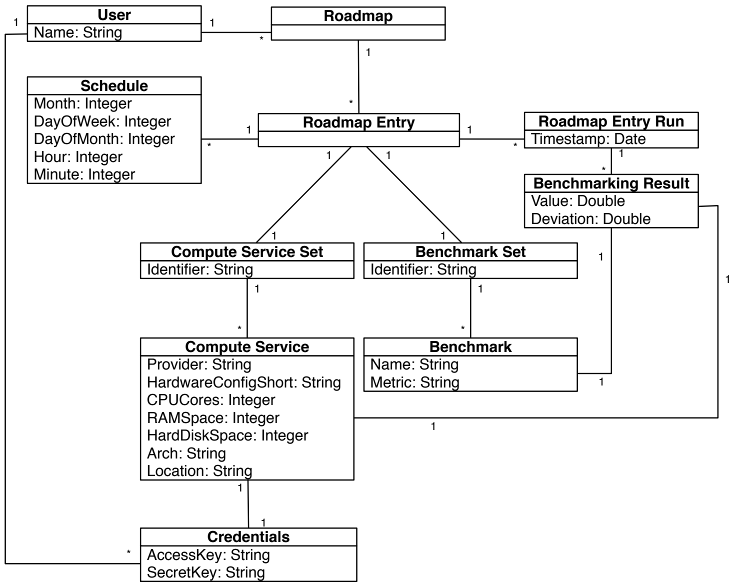


Figure 5.5.: Data Model of the Automated Performance Benchmarking Method

of compute services. A *Compute Service* is defined by a hardware configuration, a location, and optionally a short name as a label for the hardware configuration. Besides, a user can attach *Credentials* to each compute service which is required for automated access through the multi-cloud interface component.

Aside from benchmarking task definitions, the data model is capable of capturing benchmarking results. Roadmap entries will lead to the execution of benchmarking software on compute services which produces a measured values. For every combination of a benchmarking software executed on a compute service, a value needs to be retained.

In addition, it must be possible to relate a measured values to a roadmap entry. Therefore, upon execution of a roadmap entry, a *Roadmap Entry Run* entity with a timestamp can be added to the data model. For every recorded run, measured values can be stored as *Benchmarking Result* entities. The results are each in relation to a roadmap entry run, thus, allowing to relate results to a timestamp and roadmap of a user. This enables the access and an aggregation of the performance meta-data gained from the benchmarking process.

5.1.4. Process Model for Using the Automated Benchmarking Procedure

For using the performance measurement procedure, an extended view that involves a consumer describes preparation steps and sets them in context of the automated measurements. Aside from steps performed by a compute service consumer, a system implementing the procedure is involved. A representation of the process model in BPMN [208] is depicted in figure 5.6.

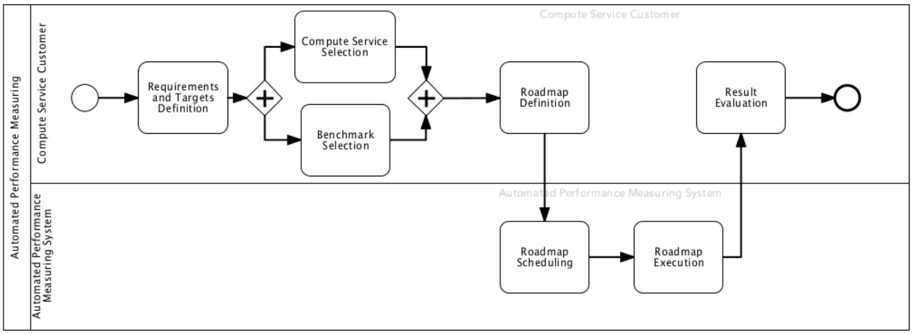

Automated Performance Measuring
Compute Service Customer
Automated Performance Measuring System
Compute Service Customer
Requirements and Targets Definition
Compute Service Selection
Benchmark Selection
Roadmap Definition
Result Evaluation
Automated Performance Measuring System
Roadmap Scheduling
Roadmap Execution

Figure 5.6.: User Involvement in an Automated Benchmarking Process

The process model reflects the two participating parties in an automated performance measurement of compute services: (1) a user with the intend to gain performance results, (2) an implementation of the measurement procedure. On the user-side, steps incorporate parametrizing the system to have it measure compute services with the benchmarks of interest to the user. In conjunction, on the system side, an implementation of the procedure executes measurements according to the user's parameters. The following subsections describe the steps involved in the measurement process in more detail.

Requirements and Targets Definition

Initially, the goals of a performance measurement need to be defined. Requirements for compute services, such as only compute services in Europe or only 64-Bit platforms, help to determine the set of compute services later on. In parallel, requirements for benchmarks, such as floating point metric or memory random write metric, help to determine suitable benchmarks. Additionally, rules regarding the frequency and period of measurements need to be clarified, e.g., every 4 weeks or Monday to Friday at 12am and 4pm. The rules are the basis to define a roadmap to schedule measurements.

Compute Service Set

Given a list of supported compute services a subset needs to be filtered out which adheres to the requirements defined priorly. Apart from the requirements, costs play a significant role in the definition of the compute service set. Every measurement implies cost for the use of the particular compute services differing in prices. Hence, the size of the set should fit the budget available for measurements.

Besides, the compute services included in the set can be selected by numerous criteria, depending on the preferences and goals of the consumer. Criteria like required minimum hardware configurations and locations of the hardware used in a compute service. In the end, a set of compute services from various providers must be defined. Every compute service is defined by a provider, a provider-specific identifier, and a provider-specific location. For example, a compute service could be defined by "Amazon EC2", "m2.xlarge", and "us-east-1". A user can be supported in the definition with lists of valid values.

To grant access to compute services, a user must give his consumer credentials to the system. Typically, a SSH key is needed to remotely access machines. Besides, an authentication token (typically a pair of an access key and a secret key) is required to access the API of a provider for a user. Using the API, the benchmarking approach can instantiate new VMs for measurements or access existing machines.

Benchmark Set

In parallel to the compute service set, the set of benchmarks to conduct on the compute services needs to be defined. The set should respect the requirements formulated in the first step. Again, since benchmarks are not executed in parallel on a single machine, the cost for the use of the compute service increases with every benchmark. Depending on the pricing precision – by second, hour or day –, the number of benchmark executions fitting in a segment varies. In case of significant deviations between benchmarking rounds, the duration of a benchmarking execution, and hence the cost, becomes unpredictable.

Roadmap Definition

In order to observe performance attributes over a certain time span, a roadmap needs to be be defined. A roadmap represents a manual schedule of benchmarking runs with periodical repetitions. In order to define a roadmap, the planned repetitions must be expressed in the roadmap definition language (c.f. section 5.1.2). The formulated schedule statements set the frequency of repetitions for the execution of a set of benchmarks on a set of compute services. Also, the number of repetitions of a benchmark set and each benchmark can be defined.

To decrease the length of a statement, a priorly defined set of benchmarks and set of compute services can be referenced via identifiers. The benchmark set and compute service set identifiers declare which benchmarks to run on which compute services. The sets have been defined in the previous steps.

To express complex schedules, multiple entries for the same benchmark and compute service set can be created. In addition, the use of extended wildcards (i.e., */[expression]) allows to define repetitions explicitly. For example, when setting the first parameters to "*/5 * * * *" an execution for every five minutes is scheduled. Following more examples of roadmap entries with valid syntax:

- */15 * * * * benchmark-set-1 compute-service-set-7
- 0 12 * * * benchmark-set-3 compute-service-set-2

Roadmap Scheduling

The scheduler considers roadmap entries made available in the *user roadmaps database* (cf. figure 5.3). Based on the database entries, the scheduler derives a schedule order and when to trigger a benchmarking run. The scheduling timeline is divided into one minute segments, which is the smallest unit possible in the roadmap definition language. Every minute, the schedule queue is consulted to find out if roadmap entries are scheduled for execution. If the next entry is to be executed in the current minute, the execution of a roadmap entry takes place.

Roadmap Execution

In the execution phase, a roadmap entry is handed over from the scheduler to the executor component. The executor initiates an automated benchmarking process which executes a benchmarking suite on a Virtual Machine. Once the benchmarking suite finished its measurements, the executor component terminates the process.

From a roadmap entry, the compute service set identifier leads to the compute services considered by the executor. For every compute service, a VM is instantiated via the multi-cloud interface. Using given SSH keypairs, the machine can be accessed and prepared by the executor.

Once a connection to the instantiated VM has been established, a benchmarking suite can be compiled using the benchmark set identifier. Therefore, the benchmarks included in the defined benchmark set are downloaded from the *benchmark repository* and compiled into a benchmarking suite.

The benchmarking suite can then be executed as a single program to conduct measurements. After the benchmarking suite has finished, the measurement results need to be uploaded to the *user roadmap & results store*. Ultimately, the executor is signaled the termination of the benchmarking and ends the process. Therefore, the VM is stopped via the *multi-cloud interface*.

Depending on the implementation, multiple compute services are measured in parallel or subsequently. Roadmap entries scheduled at the same time should be processed in parallel.

Result Evaluation

Ultimately, the user can access collected performance meta-data from the benchmarking process via the results store. The user interface can provide raw data downloads and include graphical visualizations for the benchmarking results, such as histogram charts or tables. Using the collected meta-data, a user can conduct further analysis or incorporate the benchmark results in decisions.

5.1.5. Costs, Timeliness & Target Audience

The costs for the benchmarking primarily accrue from leasing machines from the compute services. The Amazon Web Services (AWS) EC2 compute services, for example, charge between $0.01 and $7 per hour per VM. Depending on the number of benchmarking scripts, one or multiple hours are needed to assess the performance. In the worst case of five repetitions, a duration of three hours per benchmarking run, and the most expensive EC2 compute service, costs can sum up to $105 for benchmarking a single compute service. When defining a roadmap, the number of repetitions should be aligned with the available cost budget as costs may run out of hand, specifically, for expensive compute services.

As the example with EC2's most expensive compute service shows, permanently benchmarking all available compute services with a range of benchmarking scripts and an ensuring number of repetitions can exceed even biggest cost budgets. The approach is primarily suitable to benchmark a few compute services regarding specific characteristics employing well-selected benchmarking scripts. A wider use is only achievable by pooling the cost budgets of various compute service consumers and benchmark all services in a joint effort.

It is important to note that the approach can, furthermore, only provide representative and timely results to a certain level. This is due to two major factors: (1) the approach can only benchmark a sample, and (2) compute services are constantly changing. Even with many repetitions or parallel executions, only a subset of the physical machines and

VMs of a compute service are benchmarked. Therefore, the benchmarking results are only representative for the sample of VMs observed at the time of the benchmarking. Moreover, as cloud providers purchase new hardware to maintain and expand their data centers, every future benchmarking result may show different results. The frequency and dimensions of hardware upgrades are different between providers and cannot be predicted.

Nevertheless, scheduled benchmarking runs help to identify a trend in the long-term (months or years) influenced by a provider's hardware purchases. In contrast, repetitions can only determine the gravity of performance variations due to heterogeneous physical machines in a provider's data centers. Again, the heterogeneity expressed in metrics, such as the standard deviation, is only valid for the examined sample.

All in all, the representativeness and timeliness of a compute service benchmarking approach are limited. Nonetheless, consumers can benefit from the results of smaller samples and determine a minimum and average performance capacity available for a system under development. As several experiments show (c.f. section 5.5.4), variations in benchmarking results of compute services are existent but limited in their gravity. Hence, measured performance attributes can support predictions and are valuable in planning tasks, such as capacity planning. Moreover, benchmarking results from samples of different compute services give consumers a basis for more precise comparisons than using the scarce performance meta-data published by cloud providers.

5.2. Compute Service Performance Scoring

Performance benchmarking results are commonly expressed as comparable numbers. Therefore, the MADM-based scoring method offered by the framework can adapted for performance scorings. Thereby, compute services become comparable according to scorings based on performance meta-data.

In this section, a MADM-based scoring method is introduced that uses the AHP to map multiple benchmarking results to a single score. First, the advantages and reasons for a single-dimensional value are explored. Then, the $(MC^2)^2$ framework is presented that provides a process to define a specific evaluation function that enables as scoring. Finally, two scoring mechanisms for compute service performance meta-data are presented.

5.2.1. Scoring for Performance Meta-Data

When assessing performance meta-data of compute services with various benchmarks, the outcome is an array of performance attributes. A comparison of compute services based on an array of performance attributes is rather cumbersome. A consumer might set a focus on some of her preferred attributes. But then a range of assessed meta-data is ignored in a comparison.

Furthermore, a comparison according to an array of assessed performance attributes while pertaining distinct preferences for attributes is complex. A consumer needs to look at pairs of compute services and compare them according to each attribute. While doing so, he would need to remember which attributes are more important. For few attributes and compute services, a manual comparison seems legit. However, the problem becomes intricate when a consumer tries to identify a favorite from large arrays of attributes and numerous compute services. Then, a mathematical scheme that aggregates the performance attributes into a single-dimensional value is able to decrease the complexity of comparisons for a consumer.

5.2.2. Aggregation Scheme Requirements

The major challenge for an aggregation scheme is to map multiple performance attributes to a single value while guaranteeing an identical influence from every attribute. Otherwise, the single-dimensional value reflects one attribute more than others. As performance attributes are measured on distinguished scales, attributes with large numbers would influence the aggregated value stronger than attributes with small values. Besides, some attributes are more favorable the higher the number, such as operations per second, and others are less favorable the higher the number, such as seconds. Therefore, the direction of an attribute's scale, which is either positive or negative, must be considered.

The effects of mixed scales become clear with an example. When mapping two performance attributes declared as Whetstone (HIB) and Dhrystone (LIB) benchmark results (cf. section 2.2.1) to a single-dimensional value by summing up both benchmark values to a total, the scales must be equal to gain comparable results. Let two compute services be considered in this example. In a comparison, the sum of the performance attribute values would be 2,000,120 (Whetstone: 2 Mio. MWIPS, Dhrystone: 120 sec.) and 1,900,030 (Whetstone: 1.9 Mio. MWIPS, Dhrystone: 30 sec.).

Consequently, the first compute service shows a higher total values. Now, let the Dhrystone benchmark be slightly more important to a consumer. Then, the better single-dimensional value contradicts the expected outcome. Actually, the second compute service shows a better result for the Dhrystone which is not reflected in the overall score. The contrarily directed scales with very different values lead to irritating statements with a simple sum function.

Therefore, an aggregation scheme is needed that can map an array of mixed performance metrics to a single-dimensional metric, while normalizing unequal and contrary scales. In addition, the preferences of a consumer regarding the assessed performance attributes must be considered as well.

5.2.3. Weighted Compute Service Performance Scores

The $(MC^2)^2$ provides the basis to generate an evaluation function for scoring mechanisms. To facilitate comparisons of compute service configurations regarding performance meta-data, the $(MC^2)^2$ incorporated in the assessment framework can be used to generate an evaluation function. For that purpose, it is necessary to follow the steps and gather the input parameters to configure the evaluation function. As suggested in the $(MC^2)^2$, the AHP is used as a MADM method that supports normalization of distinct metrics into a single score. The advantage of the AHP is to support hierarchies which gives more flexibility in the definition of criteria relations and, thereby, a more precise weighting.

The scenario to be defined in the $(MC^2)^2$ is the comparison of compute service configurations according to assessed performance meta-data. Clearly, the compute service configurations are the alternatives to compare. Moreover, performance meta-data must be defined as attributes and serve as the basis for criteria. The AHP requires to structure criteria in a hierarchy. Thereby, the configuration of the evaluation function is completed. Consequently, for generating an evaluation function with $(MC^2)^2$ criteria must be defined in a hierarchy as the solely left task.

To apply the evaluation function, compute service configurations together with associated performance attributes, weights, and requirements must be provided as parameters (c.f. 4.1). The performance meta-data, particularly the benchmarking results, are assigned to the function as attributes. In addition, weights need to be defined which are determined in pair-wise comparisons of criteria. Besides, requirements derived from the criteria can optionally be defined and are an instrument left to a decision-maker in compute service comparisons.

The following subsections describe the configuration of a $(MC^2)^2$ evaluation function for the purpose of compute service scoring. In this regard, the use of performance attributes as criteria, the structure of the criteria hierarchy and the determination of weights are explored. Moreover, two distinct scoring mechanisms, local and global scoring, using different forms of evaluation functions are presented.

Attributes, Criteria Hierarchy and Weights

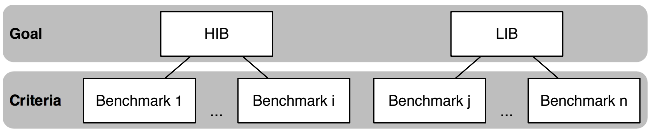
The benchmarking results of a compute service serve as attributes and the regarded compute services are mapped to alternatives. While defining alternatives and a scenario in the $(MC^2)^2$ is trivial, it is demanding to specify and structure the set of criteria. The performance attributes of compute services that are linked to benchmarking results (c.f. 5.1) can be transformed into criteria for a comparison of compute services. In the transformation of a benchmark result to a criterion, both, the benchmark result value and the observed variance factor, can be included as criteria. The criteria derived from the benchmarking results should maintain the original metrics which are commonly describing a ratio scale.

Apart from the definition of criteria, finding an adequate criteria hierarchy depends on the set of considered performance meta-data respectively benchmarks. This work proposes two schemes to structure criteria in hierarchies: (1) a simple single-level hierarchy, and (2) a hierarchy that groups benchmarks by resource type.

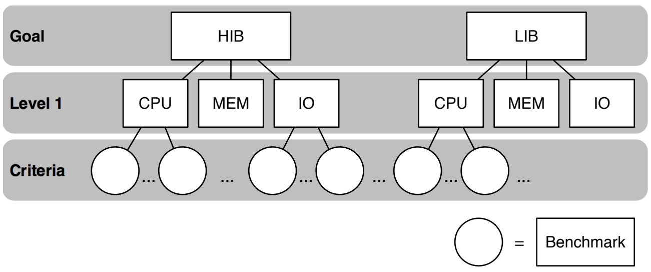
The structure of a criteria hierarchy should be aligned to the number of criteria. With many criteria it is reasonable to introduce groups and assign weights to the groups. For fewer criteria, a simple single-level hierarchy is sufficient and facilitates maximum influence on the importance of criteria through immediate pair-wise comparisons for weighting.

Local & Global Scoring

The resulting evaluation function for scoring is used with AHP and, thus, can be employed in two distinct modes: (1) local scoring, (2) global scoring. While a local scoring



(a) Simple Criteria Hierarchy of Performance Criteria



(b) Criteria Hierarchy with Resource Type Groups

mode gives means to compare multiple compute services on a [0,1] scale, a global scoring mode chooses a reference compute service. Thereby, a global scoring mode supports portable results as any compute service is measured against the reference score. In contrast, the local score is very good to show how much a certain compute service dominates other compute services in regards of their scores.

For the global mode to be used for portable and globally comparable scores, the criteria hierarchy of performance benchmarks and assigned weights need to be fixed. In other words, a very specific evaluation function needs to be defined that comprises a fixed set of benchmarks in a hierarchy and fixed weights. The capability to conduct portable global comparisons and hence traded with a lack of customizability.

For comparisons of compute services according to performance, both modes are employable depending on the situation. The global scoring mode requires a benchmarking set and scoring standard and thereby enables consumers to exchange and compare results. In contrast, the local scoring mode helps a consumer to retain easily interpretable scores of a set of compute services that he measured single-handedly with a custom set of benchmarks.

5.3. Performance Benchmarking with the Cost Budgeting Stopping Rule

In the search for a highest performing compute service, the costs of benchmarking a large set of compute services with many benchmarks can grow to overburdening amounts. Incorporating both, the priorly introduced automated benchmarking method and the scoring method, the presented stopping rule considers cost budgets in the benchmarking process. By employing the automated benchmarking method, a continuous process sequentially assesses the performance of compute services. After each assessed compute service, the stopping rule uses the scoring mechanism to compute the chance of finding a better performing compute service. At the same time, the costs of continuing the performance meta-data assessment need to be considered.

As compute service provider publish performance meta-data to a certain extent, prior knowledge can be used in the process. Although the published data is not sufficient for decisions and system engineering tasks, it indicates a rough preliminary order regarding the performance score of the compute services.

The following subsections propose a cost calculation scheme applicable with the automated benchmark method and describe the influence of prior knowledge about performance meta-data. Furthermore, the use of the scoring method for observations in the context of the stopping rule is explained. Also, the influence of the distribution of performance scores among the compute services is discussed. Finally, the use of the framework's stopping rule for performance assessments of multiple compute services is explained.

5.3.1. Cost Calculation

Costs caused by the benchmarking of a compute service mainly relate to efforts in planning, executing and evaluating benchmarks. The following cost factors have a considerable effect as operational expenditures in the course of performance assessments:

- Staff
- Resource usage for *Automated Benchmarking Coordinator* and Databases
- Compute service usage
- Network traffic

Additionally, capital expenditures may occur as requirement to conduct a performance assessment, such as licenses for benchmarking and evaluation software.

The aggregated sum of all expected costs adds up to the total costs for benchmarking a single compute service $c_i > 0$ (referred to as *per compute service benchmarking costs*). Typically, each service raises different costs c_i due to different compute service pricing models. In a simpler but less precise variant c_i can be identical.

5.3.2. Performance Scores as Observation of Compute Services

An employed stopping rule needs to come to a Boolean decision regarding the continuation of the benchmarking process. A stopping decision is commonly made based on all priorly and the currently observed value. For a benchmarking process, it is necessary to define which values are observable and how these can be employed as an indicator for a stopping rule.

In the beginning of a benchmarking process, every compute service is represented by a random variable $\mathcal{S}_i$ before it has been observed. During an observation, the compute service is benchmarked regarding multiple benchmarks to assess a range of performance attribute values.

With multiple performance attributes, such as multiple benchmarking results, diverse compute services might indicate a best performance in the multitude of attributes. Therefore, the local scoring evaluation function $f(\cdot)$ that maps multi-dimensional performance attribute values to a single score value is used as the representation (or observed value) of a compute service (c.f. section 5.2).

According to the scoring method, a score is determined in either of two possible modes: (1) idealized, (2) normalized. The idealized variant requires to define a certain compute service as the reference with score = 1. The normalized variant implicates that all measured compute services are scored relative to one another. Thereby, the scores can change with every additional assessment result in the process. Therefore, for the stopping rule the idealized is favorable with the first compute service representing the reference score. Thereby, also the recalculation of scores in every step is avoided.

Given the scores, a rank of the highest performing compute service can be determined. The score enables a stopping rule to identify the absolute highest performing compute service within the set of priorly observed compute services at any stage of a assessment process. Given the scores $\mathcal{S}_i$ of all benchmarking results, the currently highest observed utility is u_n.

5.3.3. Compute Service Order and Prior Information

As the stopping rule provided by the framework benefits from a custom order, scores and ranking must be anticipated given the available information from public sources.

Although the sources with performance meta-data about compute services are yet scarce, several sources, such as blogs, benchmark databases, and provider websites, disclose hardware figures which can be used to derive a ranking of the compute services set (c.f. section 2.2.1). Higher hardware figures in terms of numbers, such as CPU cores or amount of memory, indicate a ranking.

Which hardware figures a considered for an anticipated ranking depends on the evaluation function and the criteria incorporated in the scoring. Only hardware figures related to incorporated criteria are of interest. Besides, the weights in the evaluation function determine which hardware figures are more important. The set of compute services must then be ordered by the rough hardware figures, in descending order of expected scores.

5.3.4. Configuration of the Stopping Rule

The stopping rule provided by the framework can be employed to urge an early stop in sequential benchmarking process without any major modifications. The set of compute services and the costs for benchmarking need be defined as stopping rule parameters.

Besides, the parameters for a cost budget of c_{max}, a sliding window of ω, and score increment control constant ϵ remain available. Thereby, the stopping rule benchmarks a set of compute services subsequently while monitoring score increments within the sliding window and enforcing the cost budget.

As mentioned before, the utility gained from each additional benchmarking step is defined through the scoring function $f(\cdot)$, which is based on the benchmarking results. The goal is to maximize the utility u_n under consideration of the sum of all costs c_i and the cost budget.

The following equation shows the expression that the stopping rule seeks to maximize:

$$\max u_n - \sum_{i}^{n} c_i$$

5.4. Software Prototype for Automated Compute Service Performance Benchmarking with Stopping Rule

The Balloon Race implements the concept of automated, scheduled compute service performance meta-data assessment with a stopping rule. The Balloon Race incorporates the AoetaroaLib to implement a stopping rule that is enabled when a cost budget was defined. The Ballon Race exposes a reusable library called LibIntelliCloudBench and a web application wrapping the library together with a user interface.

5.4.1. LibIntelliCloudBench Library

The LibIntelliCloudBench library contains the logic to automatically start a VM instance, to inject and execute benchmarking software, and to download the results. For instantiating and connecting to VMs, the library uses jClouds, a multi-cloud interface library [9]. The jClouds library maintains a support for various cloud providers and offers a unified interface to manage and connect to VMs on different compute services. Furthermore, jClouds provides a database of available compute services, geographical regions and compute service (hardware) configurations which is reused in LibIntelliCloudBench.

The Phoronix benchmarking software is employed to bundle and execute sets of micro-benchmarks on a VM. The Phoronix benchmarking software (version 5.2.0) is bundled with the LibIntelliCloudBench library and the benchmarking scripts are hosted in a repository managed by the publishers of the Phoronix software. Alternatively, the benchmarking scripts and custom scripts can be placed in an online storage (e.g., an AWS S3 bucket). Any VM downloads only the necessary benchmarking scripts from the repository.

The set of available micro-benchmarks is slimmed down to scripts that are known to succeed on remote server machines. Particularly, micro-benchmarks for measuring graphics and battery performance are excluded from the set. A detailed list is given in section B.2 in the appendix.

Moreover, the stopping rule approach is implemented and included in the LibIntelliCloudBench library supporting cost budgets and sliding window sizes $\in [0, 7]$. Which stopping rule to use can be specified via the API of the library. By default, a non-stopping rule is selected which traverses all compute services specified in the set for benchmarking. An overview of the library's architecture is depicted in figure 5.8.

The library exposes a programming interface in Java that allows to trigger benchmarking executions. The use of the *CloudBenchService* demands a minimal configuration

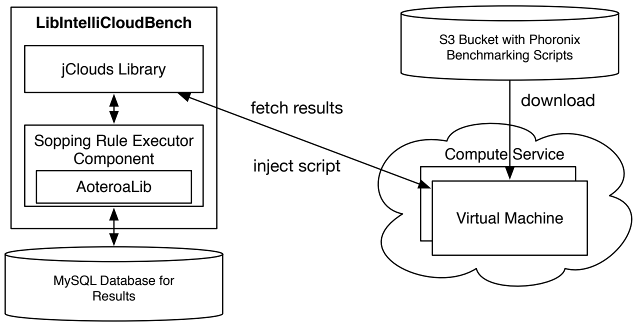


Figure 5.8.: Architecture of LibIntelliCloudBench

via the *CloudBenchServiceConfig* class before an execution. A *CloudBenchService-Config* comprises a *StoppingRuleConfig*, where the sliding window size and cost budget can be defined, and a set of *ComputeService*s from the jClouds library and *Benchmark*s. After a successful execution, the *CloudBenchService* allows to recall a *CloudBenchSer-viceResult*. A result is defined by the *ComputeService*, the *Benchmark*, and the *Result-Data*. The *ResultData* is extracted from the XML files produced by the Phoronix micro-benchmarks and converted into a Java data structure. For every *Benchmark* and *Com-puteService* combination, multiple corresponding *ResultData* items can exist. Each item represents the extracted result value and metric measured in a benchmarking run. A simplified representation of a UML class diagram of the programming interface is illustrated in figure 5.9.

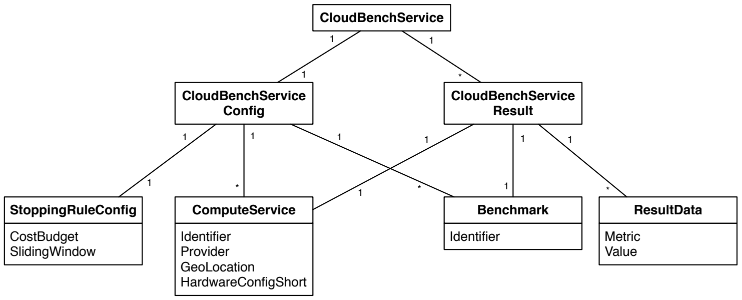


Figure 5.9.: Data Model LibIntelliCloudBench

The class diagram shows the two major attributes: (1) the configuration, and (2) the set of results. To initiate a benchmarking process, the configuration options need to be given and then the *startBenchmarking()* method must be called. Eventually, the process finishes and signals it with an optional callback that can be set in the configuration. By now, the results data structure has been filled from the XML files that were generated by the Phoronix micro-benchmarks on the VMs.

The source code of the library is available as part of a Github repository under `https://github.com/mugglmenzel/intellicloudbench`.

5.4.2. Web Application

The Balloon Race web application has been developed with the intention to provide an interface to automated benchmarking for non-programmers. Therefore, the Balloon Race is realized as a web application with a graphical user interface using the Vaadin framework [209]. The Vaadin framework provides components for a graphical user interface similar to and compatible with GWT [202] and smartGWT [203].

Moreover, the LibIntelliCloudBench is integrated into the Balloon Race web application to execute benchmarking processes. The library takes compute service and benchmarking script sets as configuration parameters, and accepts an optional stopping rule definition. In a stopping rule definition, the sliding window size and a cost budget can be defined.

The Balloon Race has been implemented for the Google AppEngine [201] and for Tomcat application servers [210] with a MySQL database [211]. Figure 5.10 depicts the architecture for the Tomcat version of Balloon Race.

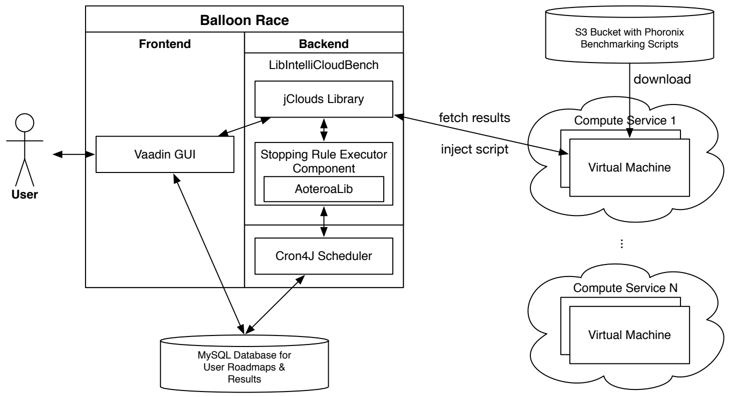
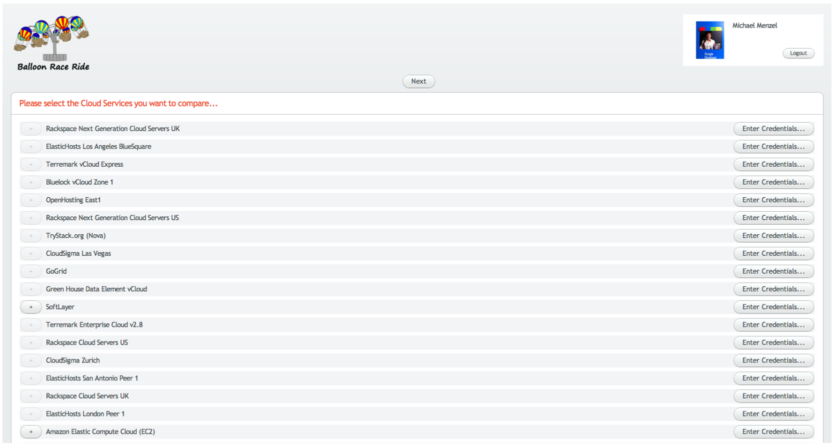


Figure 5.10.: Architecture of the Ballon Race Web Application

In the graphical user interface, a user is guided through three steps, from compute service selection via benchmark selection to stopping rule configuration. The listed compute service configurations are drawn from the jClouds library and differentiated by provider, geographical location, and hardware configuration. A user can select multiple compute services and provide the credentials for the diverse providers, in the first step.

In the second step, the list of micro-benchmarks matches the ones available from the employed Phoronix software. The last configuration step allows to define a cost budget and define the sliding window for the stopping rule. By setting no cost budget and sliding window, the stopping rule can be omitted. Figure 5.11 depicts a screenshot of the web application.



Please select the Cloud Services you want to compare...

Rackspace Next Generation Cloud Servers UK	Enter Credentials...
ElasticHosts Los Angeles BlueSquare	Enter Credentials...
Terremark vCloud Express	Enter Credentials...
Bluelock vCloud Zone 1	Enter Credentials...
OpenHosting East1	Enter Credentials...
Rackspace Next Generation Cloud Servers US	Enter Credentials...
TryStack.org (Nova)	Enter Credentials...
CloudSigma Las Vegas	Enter Credentials...
GoGrid	Enter Credentials...
Green House Data Element vCloud	Enter Credentials...
SoftLayer	Enter Credentials...
Terremark Enterprise Cloud v2.8	Enter Credentials...
Rackspace Cloud Servers US	Enter Credentials...
CloudSigma Zurich	Enter Credentials...
ElasticHosts San Antonio Peer 1	Enter Credentials...
Rackspace Cloud Servers UK	Enter Credentials...
ElasticHosts London Peer 1	Enter Credentials...
Amazon Elastic Compute Cloud (EC2)	Enter Credentials...

Figure 5.11.: Screenshot of the Ballon Race Web Application

In regards of the roadmap definition language, the prototype spares a graphical interface to define roadmaps, but provides a programming interface. Definitions are stored in the database and translated into scheduled jobs with the cron4j library [212]. A definition takes a set of compute services and benchmarking scripts, and a cron scheduling string with repetitions parameter as defined by the roadmap definition language. The definition is stored in the database with the sets and the cron scheduling string including the repetitions parameter. When a scheduled job is triggered by cron4j, it reads the compute services and benchmarking scripts sets from the database and uses them as parameters for a new benchmarking execution. On every boot of the web application all roadmap entries in the database are scheduled as cron4j jobs. New roadmap entries are scheduled upon creation.

The source code of the Balloon Race is available as a Github repository under `https://github.com/mugglmenzel/BalloonRace`.

5.5. Evaluation of the Automated Performance Benchmarking Method

To evaluate the proposed automated performance benchmarking method, this section explores the validity and quality of the approach, particularly in the light of research question $RQ1a$:

> *RQ1a. How can compute services be assessed at run-time in an automated manner?*

More specifically, research question $RQ3$ is explored in this section to answer $RQ1a$ in regards of performance assessments. Thereby, not only the applicability of the framework is proofed but also the feasibility of run-time assessments of compute services in general.

> *RQ3. How can performance be assessed at run-time from multiple compute services in an automated manner?*

The effective implementation of a software prototype (c.f. section 5.4) serves as proof of the feasibility of the concept. The software prototype and the concept of the method have already been reviewed by and discussed by peers in the research community [80]. The results published in the research community showed the validity of the approach according to a performance benchmarking of several compute service over time. The following sections structure the results from the validation according to experiments with the software prototype. Moreover, this work adds a comparison with existing approaches and examines the promptitude of the approach using the software prototype.

First, the procedure to validate the approach in five experiments and the observed validation results are presented. Then, a comparison of the proposed approach to existing approaches is established. Next, the promptitude of the prototype is explored and the results from the observed overhead measurements are presented. In the final section, the findings from a range of measurements are presented that indicate variations in performance outcomes of compute services.

5.5.1. Validation of the Automated Performance Benchmarking Approach

The proposed automated performance benchmarking approach shall be validated by example using the software prototype. Therefore, the approach is tested in five use case experiments that stress the claims and capabilities of the approach.

First, the experiments and the experiment setup are described and a hypothesis is formulated. Finally, results are presented and a conclusion is drawn.

Experiments & Hypothesis

The set of experiments increase in complexity with each step. Specifically, the subsequent experiments vary in respect of the number of micro-benchmarks and compute services involved. Thereby, the prototype is stressed in regards of its basic support of various micro-benchmarks and of multiple compute services.

Repetitions are a considerable aspect of the proposed approach that enables to measure the performance of a compute services in the long term together with variations. A further experiment tests the capabilities to repeat micro-benchmarks. Additionally, the scheduling feature of the approach is covered in a specific use case with a performance benchmarking task scheduled in the future.

EXP-A. Basic Experiment The basic experiment test the applicability of the prototype for running a single micro-benchmark on a single compute service.

EXP-B. Multiple Benchmarks Experiment An extended experiment involves multiple, distinct micro-benchmarking results to be measured from a single compute service.

EXP-C. Multiple Compute Services Experiment An experiment in which multiple micro-benchmarking scripts must be executed against multiple, distinct compute services.

EXP-D. Repetitions Experiment The experiment tests the capability to repeat micro-benchmarks.

EXP-E. Scheduling Experiment In order to test the ability to schedule performance benchmarking tasks, an experiment comprising a measurement of one micro-benchmark on a single compute service three hours later is setup.

The overarching validation test constitutes from all four experiments and tests the following two hypothesis in the light of the research question *RQ1a*.

H_0 The approach respectively the software prototype does not provide support for automated performance benchmarking of compute services or only limited support and, thus, fails in one or more experiments.

H_1 The approach respectively the software prototype provides support for automated performance benchmarking of compute services and, thus, succeeds in all experiments.

The null hypothesis H_0 holds the default position that the proposed approach cannot proof the claim of enabling automated performance benchmarking of compute services. In contrast, the hypothesis H_1 is proofed if the prototype succeeds in the experiments altogether. Given the experiment results, a first evaluation in terms of *RQ3* and *RQ1a* can be stated using H_0 and H_1.

To research the hypothesis H_1, the software prototype has been tested in experiments. Therefore, in each experiment a small program was implemented that uses the software prototype. To realize the experiments, the small programs employ the software prototype with distinct configuration parameters, namely the set of selected micro-benchmarking scripts, the compute services set, and the repetition and scheduling parameters.

For experiment EXP-A, the compute service AWS EC2 *t1.micro* in region *us-east-1* was selected and the micro-benchmarking script Sudokut. In experiment EXP-B, the scripts Crafty and Sudokut were selected for the AWS compute service *t1.micro* in region *us-east-1*. Experiment EXP-C extends EXP-B by adding the Rackspace *Standard1* in *London* to the set of selected compute services.

The experiments EXP-D and EXP-E address the repetition and scheduling parameters of a roadmap. All experiments EXP-A to EXP-C used a default repetition count of three. In experiment EXP-D, the experiment EXP-A was conducted with a repetition count of five. For the scheduling task in experiment EXP-E, the roadmap definition language was used to execute experiment EXP-A every full hour, which succeeded three times in a row.

Results & Conclusion

From the experiments, several observations could be made and conclusions can be drawn. One aspect revealed by the experiments is the level of automation provided by the prototype. The experiments EXP-A to EXP-E were conducted by configuring the software prototype using programs in experiment setups. From there on the execution continued in an automated manner. Implicitly, the observed level of automation can be described as *batch processing* according to Endsley [193].

Furthermore, the experiments aim at confirming hypothesis H_1. The experiments EXP-A to EXP-E conducted with the software prototype show that the prototype and, hence, the approach fulfills its claims in respect of hypothesis H_1. All experiments were completed with positive results, implicating an automation and support for repetitions and scheduling are present. To guarantee reliable results, the experiments with the software prototype were each repeated multiple times, at least three times each.

The confirmation of hypothesis H_1 implies an answer to research question *RQ3* can be given. The software prototype shows a viable solution to assess the performance of a compute service at run-time in an automated manner. Thereby, indirectly *RQ1a* is addressed which questions the feasibility of an automation of assessments. The existence of a solution to automated assessments of compute services' performance proofs also the existence of automated assessments in general.

5.5.2. Comparison to the State of the Art

For a further evaluation of the approach, existing approaches are examined regarding the experiments *A-D*. Together with the results from the experiments with the proposed approach a funded comparison can be made.

The approaches selected for the comparison are the state of the art available for automatically benchmarking the performance of compute services (c.f. chapter 3). Namely, the BEEN architecture from Kalibera et al. and the online service Phoromatic [178] are considerable alternatives to compare against. Other approaches that succeeded in measuring compute service performance do not provide an automation, focus on a certain compute service (or a small set), and miss a publicly available implementation.

Unfortunately, the BEEN architecture does not offer an implementation for compute services, nor does it describe how to apply the method for the measurement of compute service performance. Therefore, the Phoromatic implementation is the only subject available for a comparison between the proposed approach and the state of the art.

For a comparison, the set of experiments EXP-A to EXP-E serve as the basis. By testing the state of the art according to the experiment set, a comparison with the proposed approach can be carried out. As described in the validation in section 5.5.1, the proposed approach succeeded in completing all experiments EXP-A to EXP-E successfully in multiple observations.

In the test series with the Phoromatic approach the experiments EXP-A to EXP-C completed successfully with limitations and experiments EXP-D and EXP-E did not succeed. The results are summarized in comparison to the proposed approach in table 5.1.

Approach	Exp-A	Exp-B	Exp-C	Exp-D	Exp-E
Proposed Approach	✓	✓	✓	✓	✓
Phoromatic	✓[1]	✓	✓	×	✓[2]

Table 5.1.: Decision Models used for Experiments

Phoromatic does allow to automatically install and execute performance micro-benchmarking scripts on remote systems of diverse compute service providers. However, the Phoronix test suite software must be installed manually on the target system and registered with `www.phoromatic.com`. Besides, the VMs at the compute services must be instantiated manually, too.

[1]Requires a manual VM instantiation and preparations.
[2]Restricted to weekdays-time schedules.

A support for scheduled execution is available, but restricted to weekday and time selection. Thereby, a daily execution of performance measurements is achieved. More complex scheduling definitions, such as monthly or hourly schedules, are not available.

Also, parameters for defining repetitions and scheduled runs of certain micro-benchmarking script executions is absent. Repetitions can partially be achieved by creating many scheduled executions with identical or similar starting times.

5.5.3. Results of the Promptitude Measurements

To gain further insights, in the experiments EXP-A to EXP-C the duration to complete the steps involved in a performance measurement were recorded. Additionally, for EXP-E the observable delay and punctuality of the cron4j-based implementation were measured. With insights in the promptitude of the software prototype, an evaluation of the practical applicability of the approach is pursued.

By injecting a time logging mechanism, the practical promptitude of gaining performance benchmarking results from the software prototype could be measured. Therefore, the durations to complete the sequential steps involved in a benchmarking run are captured by logging a timestamp before and after each of the steps.

The promptitude constitutes from the duration of the four steps *instantiate VMs*, *deploy software*, *download benchmarking scripts*, *run benchmarking scripts*, and *upload results*. The first three steps can be summarized as preparation overhead, while the latter two steps are involved with the bulk of measuring the performance of a compute service.

Figure 5.12 depicts the results from the promptitude measurements for all experiments EXP-A to EXP-C as stacked bars. For experiment EXP-C which benchmarks multiple compute services, the total duration equals the maximum duration of benchmarking the whole compute services set as measurements are conducted in parallel.

The results show that the preparation overhead affects the total time to conduct the benchmarking with less than 220 seconds in all experiments. Particularly, the overhead accounted for 45.56% of the total time in experiment EXP-A, 17.92% in EXP-B, and 21.59% in EXP-C. The overhead constitutes from instantiating VMs and deploying necessary software and libraries. The duration to complete the preparation overhead was exhibited as rather stable over all experiment repetitions. However, in experiment EXP-C the duration and, thus, the overhead increased due to the VM instantiation delay of the compute service Rackspace *Standard1*.

As the experiments reveal, the duration of the subsequent steps for downloading and executing benchmarking scripts, and uploading measurement results is immediately linked to the number of selected scripts involved. During the experiments, a variation

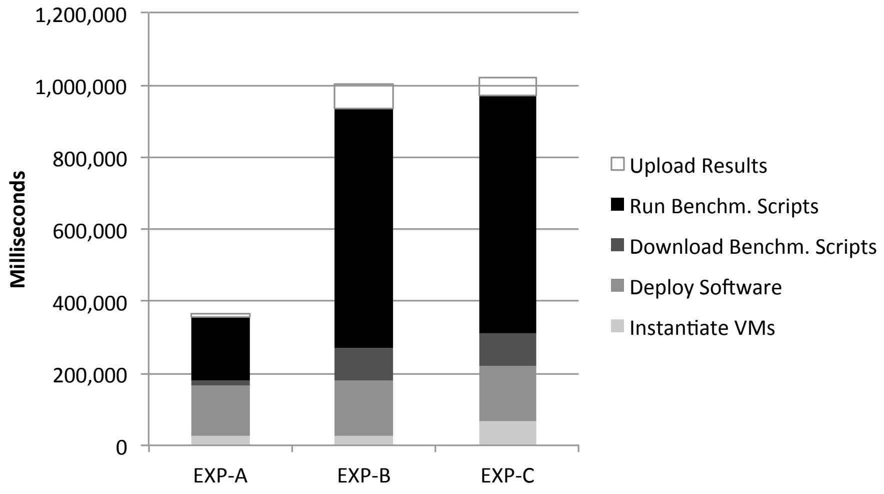


Figure 5.12.: Measured Durations in Experiments EXP-A to EXP-C

in the number of selected micro-benchmarks resulted in diverse total times. Similarly, experiments with multiple compute services confirm that the duration of the benchmarking step depends heavily on the performance of the compute service under test (c.f. section 5.5.4).

As experiment EXP-C shows, the deployment of the benchmarking software and setup of the host VM depends mainly on the performance of the VM. The setup on the Rackspace *Standard1* compute service took half the time of the worse performing AWS *t1.micro* compute service.

In contrast, the network bandwidth between the system orchestrating the performance measurement process and the host VM at the compute service is neglectable at today's standards already. The necessary basic software is about 800 kilobyte large. The benchmarking scripts, however, can be many megabytes large (e.g., Crafty is 45 megabytes) and network bandwidth of the compute service can become a source of delay during the *download benachmarking scripts* step.

For experiment EXP-E, an average delay of 9 milliseconds and a maximum delay of 24 milliseconds could be observed over 100 scheduled test executions. The delay is induced by the scheduling business logic, which is based on the cron4j library [212], and occurs before the measurement process itself is triggered.

5.5.4. Statistics for AWS EC2 Compute Services

In addition to the validity and promptitude analysis, the software prototype has been used to gain performance meta-data for AWS EC2 compute services over several months. Particularly, statistics about the compute services *m1.small*, *m1.medium*, and *m1.large* were assessed with repeated performance measurements. For the measurements, multiple micro-benchmarking scripts from the Phoronix test suite were used.

With the software prototype's support for repetitions, the measurements could be conducted over several months to explore potential performance variations. The results for the the *m1.small*, *m1.medium*, and *m1.large* compute services concerning the *lower-is-better* processor micro-benchmarks "Crafty", "dcraw", and "Sudokut" are depicted in figure 5.13. Figure 5.14 depicts the results for the *higher-is-better* processor micro-benchmarks "John The Ripper", "OpenSSL", "Opstone Singular Value Decomposition (SVD)", and "Opstone Sparse-Vector Scalar Product (SVSP)". The figures also include error indicators that reflect the standard deviation of the results gained from multiple runs over month.

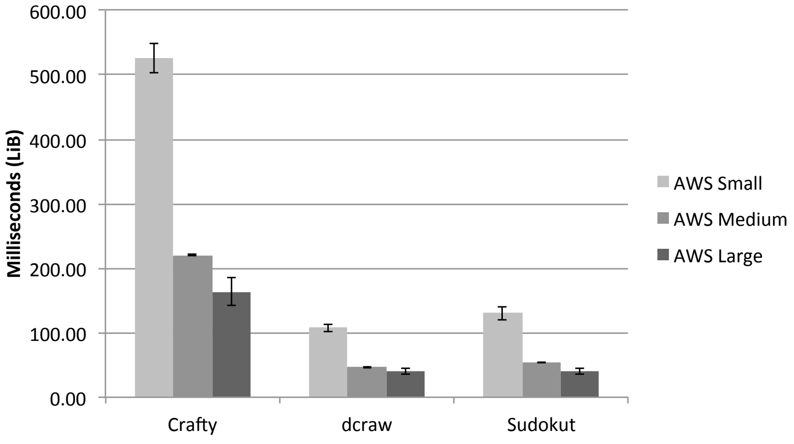


Figure 5.13.: Assessed Performance Meta-Data for EC2 Compute Services (LiB)

The results confirm a difference in available computation capacity for the compute services *m1.small*, *m1.medium*, and *m1.large*, in this particular order. Between *m1.small* and *m1.medium*, the difference can be as much as double the performance or half the computation time for tasks, respectively. Besides, as indicated by the standard deviation, the gravity of performance variations differs between the compute services. The *m1.small* compute service configuration proofs to show the most stable performance outcomes over the multiple iterations. Depending on the micro-benchmark, the variations of the *m1.small* and *m1.large* compute services expressed as standard deviation

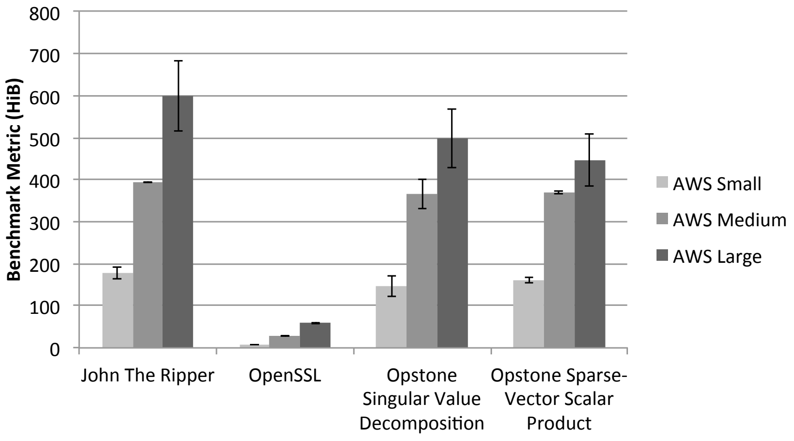


Figure 5.14.: Assessed Performance Meta-Data for EC2 Compute Services (HiB)

can be up to 16% (*m1.small*) respectively 14% (*m1.large*).

A further analysis of the available processor hardware disclosed by the employed Ubuntu operating system provided additional information to identify a possible reason for performance variations. For that the software prototype has been extended with a custom script that executes an additional command on the host system to retrieve and upload the CPU information. Table 5.2 lists the odds of observing a certain processor type of the EC2 compute services *m1.small*, *m1.medium*, and *m1.large*. The results are based on run-time inspections of 462 *m1.small*, 72 *m1.medium*, and 186 *m1.large* VM instantiations. Besides, the processor scores published by "Passmark" [213, 214] originating from processor hardware tests are included in the table.

Processor Type	Passmark	m1.small	m1.medium	m1.large
AMD Opteron 2218 HE	1471	32.9%	-	-
AMD Opteron 2374 HE	4642	-	-	0.54%
Intel Xeon E5345	2882	-	12.5%	-
Intel Xeon E5410	3448	-	87.5%	-
Intel Xeon E5430	4008	67.1%	-	99.46%

Table 5.2.: Frequency of Processor Types per EC2 Compute Service

As the numbers in table 5.2 indicate, the processor types assigned by the compute services when instantiating a VM are inconstant. While the *m1.large* provides a "Intel Xeon E5430" processor type in more than 99% of the cases, the *m1.small* may return an "AMD Opteron 2218 HE" instead in almost every third VM instantiation. In addition, the *m1.medium* compute service alternates between the "Intel Xeon E5345" and "Intel Xeon E5410" processor types.

All in all, the "Passmark" scores and the benchmarking results assert that different processor types mean different performance outcomes. This is particularly interesting as EC2's *m1.small* compute service assigns very different processor types and charge the same price. An indicator for alternating processor types or variable performance cannot be found on AWS's website or in any white papers.

The results from AWS EC2 confirm the importance of a novel approach for single-handed assessments of compute service performance. As with AWS, the performance figures published by compute service providers may not be trusted. The actual performance outcomes can vary over time or simply differ tremendously from the officially announced numbers.

5.6. Evaluation of the Performance Benchmarking Stopping Rule

A primary feature of the stopping rule presented in this work is to consider a cost budget in a sequential performance measurement process with multiple compute services. Besides, the rule aims at saving time and cost when an approximate order of the compute service set by performance outcome is given.

The stopping rule has been developed to address research question $RQ2$:

RQ2. How can costs for run-time assessments of multiple compute services be budgeted?

The concept of the stopping rule has been presented to and discussed in the research community [215]. The implementation of the stopping rule in a software prototype (c.f. section 5.4) asserts the feasibility and conceptual validity of the approach. Furthermore, the suitability of the proposed stopping rule in compute service benchmarking scenarios is tested in a validation by example which focuses primarily on the cost budget support. Besides, a comparison with standard stopping rules allows for a detailed evaluation of the stopping rule's limitations and shortcomings.

The first subsection presents the validation procedure that inspects the stopping rule's cost budget feature. Then, a comparison with standard stopping rules is conducted. Finally, the results and findings from the validation and comparison are presented.

5.6.1. Validation of the Performance Benchmarking Stopping Rule

A common requirement for stopping rules is to always lead to a stop (*eventual stopping*). In the case of compute service benchmarking, the stopping rule is always applied to a finite number of services. The heuristic of the proposed stopping rule only continues if the set of compute services is not completely traversed. After a full traversal of the finite compute service sample set, the stopping rule will stop the performance benchmarking procedure and the requirement of eventual stopping is thereby satisfied.

Aside from eventual stopping, the stopping rule claims support for cost budgets. Therefore, the ability to stop within a cost budget is tested in an experiment. For the experiment, a sufficiently large compute service set must be defined that exceeds the cost budget according to a rough estimate. The stopping rule is employed in a sequential benchmarking of the compute services with respect to a single micro-benchmark.

The outcome of the experiment is tested against the following hypotheses in the light of research question *RQ2*:

H_0 The stopping rule respectively the software prototype exceeds a given cost budget.

H_1 The stopping rule respectively the software prototype stops before a given cost budget is violated.

While the null hypothesis H_0 assumes the stopping rule to fail in applying a cost budget, the hypothesis H_1 is proofed when a cost budget is enforced successfully. To test the hypothesis, an experiment with the software implementation of the stopping rule is conducted.

For the experiment, the software prototype implementing the stopping rule has been altered to not stop based on calculated utility values from the performance measurements. The performance results are collected to maintain the task and context, but ignored by the rule. Thereby, the prototype assured that the process of performance measurements proceeds until the stopping rule intervenes only due to the given cost budget. Besides, to reduce the time for the execution of the experiments, the stopping rule didn't wait for the compute service instances to finish the performance measurements. Instances were started subsequently until the stopping rule terminates due to the cost budget.

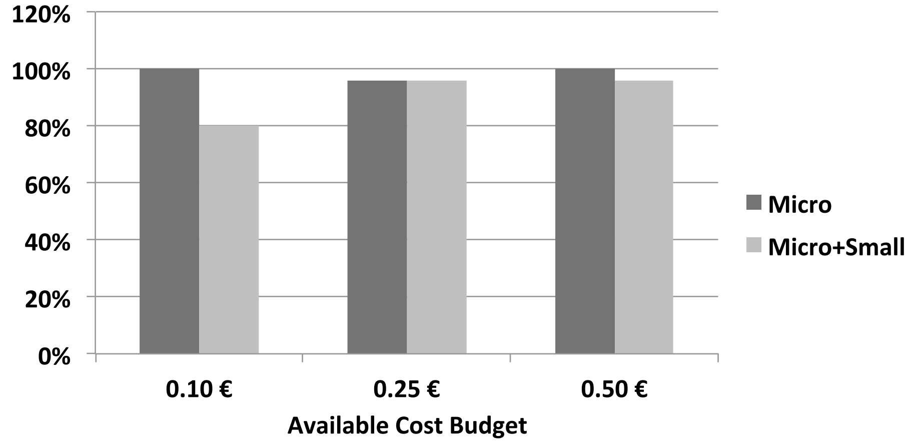


Figure 5.15.: Observed Cost Budget Consumption (in Percent)

Two different compute service sets based on the instance types offered from Amazon Web Services have been defined for the experiments. The first set, referred to as "Micro", comprises only micro instances (instance type *t1.micro*) with a price of $0.02 per hour. The second set, referred to as "Micro+Small", includes an iterating mix of micro and small instances (instance type *m1.small*) with $0.02 and $0.06 per hour. A total of 100 instances were included in the compute service sets which leads to maximal costs of $2 respectively $4 for the mixed compute service set.

The proposed stopping rule's cost budget feature was tested for cost budgets of \$0.1, \$0.25, and \$0.5 with the aforementioned compute service sets. In all six possible combinations of compute services and budgets, the proposed stopping rule did not permit a violation of the cost budget. Depending on the compute service set and the hourly costs, the available budget was consumed to a certain degree. How much of the budget was consumed is in direct relation to the granularity of the hourly costs. Figure 5.15 depicts the cost budget consumption for the two sets "Micro" and "Micro+Small" in percent.

The results confirm the hypothesis H_1 at least for the 6 explored experiments. Thereby, the results give a foundation to state that the proposed stopping rule serves as an answer to research question *RQ2*. The proposed stopping rule shows that costs can be budgeted in the course of measuring the performance of multiple compute services.

5.6.2. Comparison to Standard Stopping Rules

The stopping rule proposed in this work incorporates a heuristic that exploits an approximate ordering of the compute services. To evaluate the quality of the heuristic, a comparison with existing standard stopping rules is conducted. The comparison is capable of identifying the existence of a potential advantage of the *proposed stopping rule (PSR)* over the standard stopping rules *cut off rule (COR)*, *candidate count rule (CCR)*, *successive non-candidate rule (SNCR)*, and *odds algorithm (OA)*.

For the comparison, the standard stopping rules and the proposed heuristic-based stopping rule are compared in several scenarios. The scenarios differentiate in the order of n incorporated compute services, with $n = 10$. Theoretically, $n! = 3628800$ possible orders could be explored to compare the rules.

For an in-depth examination of the stopping rules, four particular orders are selected: *perfect order*, *perfect order with one switched pair*, *perfect order with two switched pairs*, and *reversed perfect order*. Particularly, the first three orders are expected to resemble very common cases when prior information, such as hardware figures, is available. In each of the order scenario, the observations the stopping rules are going to make are predetermined for the sake of repeatable experiments. The exact performance score values of the predetermined ten observations are defined by the order scenarios.

In each of the scenarios, the proposed stopping rules and the standard stopping rules are compared according to the factors *best compute service found*, *rank of best candidate*, and *stopping index*. The first factor indicates whether a stopping rule succeeded in finding the highest scoring compute service in the set. In case it was not found, the second factor reflects the global rank of the best candidate that was identified by the stopping rule. The last factor aims at exposing how much cost the stopping rule generates due to the number of measured compute services.

The stopping rules are tested with every possible parameter value, e.g., cut off observations, candidate count, or non-candidate count. Unlike the other stopping rules, the OA has no value-oriented parameters, but a formula must be defined for parameter p_i. Therefore, the OA is constantly used with the parameter $p_i = \frac{1}{i}$. In the comparison, the parameters setups achieving the best results are presented for each stopping rule. The results of an arbitrary parameter setup are presented if all parameters setups of a stopping rule return the same results.

Scenario 1: Perfect Order

In the perfect order scenario, the situation that prior information proofs correct is created. The observed score values are decreasing with every additional measurement result of a compute service. This scenario is presumed to occur frequently when available hardware figures of compute services indicate the perfect order. The observed scores in descending order are depicted in figure 5.16.

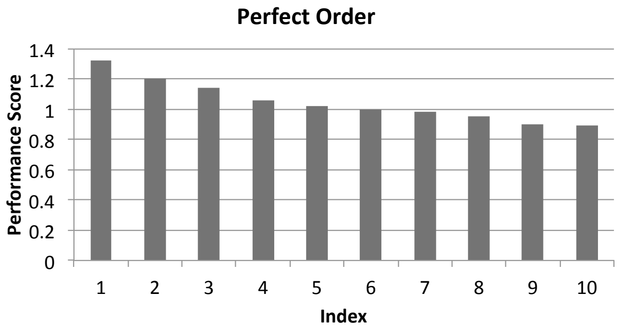


Figure 5.16.: Perfect Order

Based on the perfectly ordered score observations, the stopping rules halt at different indices in the benchmarking process. The best stopping outcomes have been achieved with $m = 1$ for all rules. Table 5.3 contains the results for all stopping rules in the best parameter setup.

The results show that particularly the proposed stopping rule (PSR) outperforms all other stopping rules clearly. In a perfect order scenario, all other stopping rules traverse the whole compute service set. Albeit the index of the highest scoring compute service is 1, all stopping rules (except the proposed rule) incur maximal costs in their search.

Stopping Rule	Best Found	Rank Best	Stopping Index
PSR-1	✓	1	2
PSR-2	✓	1	3
COR-1	✓	1	10
CCR-1	✓	1	10
SNCR-1	✓	1	10
OA	✓	1	10

Table 5.3.: Results for Scenario 1

Scenario 2: Perfect Order with One Switched Pair

A further presumably realistic scenario is a perfect order with a single modification. To impose a slight error, one pair of compute services switch position in the sequential order. Thereby, the case of misleading hardware or performance figures is explored for a certain scenario of a single error. The order and compute service scores are depicted in figure 5.17. The compute services with index 2 and 3 have changed positions.

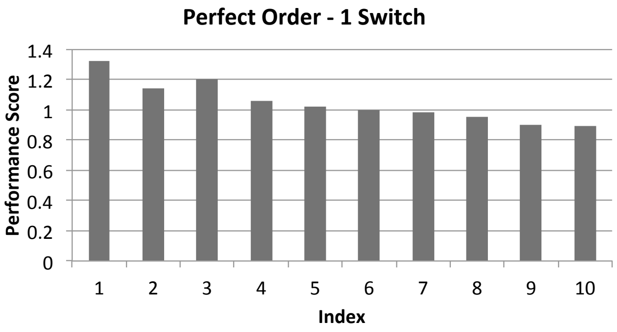


Figure 5.17.: Perfect Order with One Switched Pair

The application of the stopping rules to the almost perfectly ordered compute service set shows unequal performances. The best performances have been achieved with $m = 1$ for all rules. The results from the experiments are listed in table 5.4.

The results show that the proposed stopping rule (PSR) outperforms all other stopping rules in scenario 2. Even with a slight error in the perfect order, all other stopping rules observe all compute services and generate more costs than the proposed rule.

Stopping Rule	Best Found	Rank Best	Stopping Index
PSR-1	✓	1	2
PSR-2	✓	1	5
COR-1	✓	1	10
CCR-1	✓	1	10
SNCR-1	✓	1	10
OA	✓	1	10

Table 5.4.: Results for Scenario 2

Scenario 3: Perfect Order with Two Switched Pairs

In scenario 3, the error rate in the perfect order is further increased. Therefore, an additional pair switch is imposed. Thereby, the case of imperfect information about the performance score of 4 compute services due to misleading hardware or performance figures is created. Figure 5.18 presents the order that includes two switched compute service pairs and is derived from the perfect order. The indices 1 and 2, and 3 and 4 have switched positions in comparison to the perfect order.

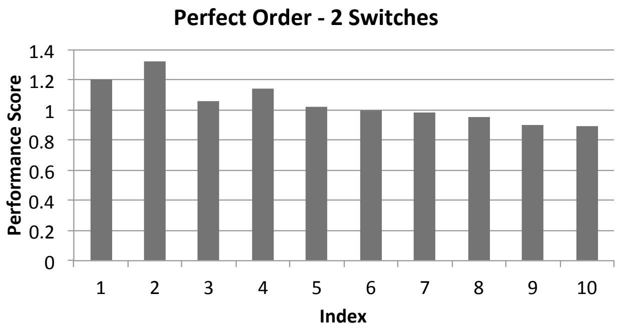


Figure 5.18.: Perfect Order with Two Switched Pairs

The stopping rules have been applied with all possible values for m (except for the OA). The best results were achieved with $m = 1$. Table 5.5 lists the finding for all stopping rules. For detailed explanations the table includes the results for $m \in 1, 2$ for most rules.

For two errors in the perfect order, two stopping rules, namely COR-1 and CCR-1, are able to outperform the proposed rule. Instead of 3 observations, COR-1 and CCR-1 only need 2 observations to stop and find the best scoring compute service. Nevertheless, when increasing m to 2, the proposed stopping rule outperforms all other rules.

Stopping Rule	Best Found	Rank Best	Stopping Index
PSR-1	✓	1	3
PSR-2	✓	1	6
COR-1	✓	1	2
COR-2	✓	1	10
CCR-1	✓	1	2
CCR-2	✓	1	10
SNCR-1	✓	1	10
OA	✓	1	10

Table 5.5.: Results for Scenario 3

Scenario 4: Reversed Perfect Order

To give a contrasting scenario, the compute service set of the perfect order is reversed. Thereby, a stopping rule needs to measure all compute services to discover the absolute highest scoring service. Although the order is obviously the reversed copy of scenario 1, the reversed order is depicted in figure 5.19 nonetheless.

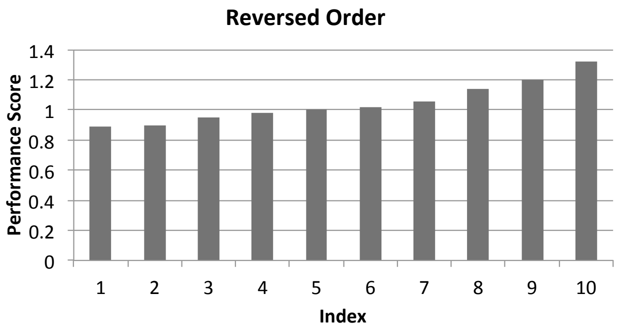


Figure 5.19.: Reversed Perfect Order

For the comparison, the same stopping rule setups with $m \in 1, 2$ are used to show their performances in the contrasting reversed perfect order. The findings with regards to the performance of the stopping rules are listed in table 5.6.

As the results show, only the proposed stopping rule and the rule SNCR-1 are able to determine the absolute highest scoring compute service in the set. All other rules halt before the 10th index and miss the highest scoring service. The cut off and candidate count rules succeed with $m \geq 9$, the odds algorithm would need a different p_i definition to succeed.

Stopping Rule	Best Found	Rank Best	Stopping Index
PSR-1	✓	1	10
PSR-2	✓	1	10
COR-1	×	9	2
COR-2	×	8	3
CCR-1	×	9	2
CCR-2	×	8	3
SNCR-1	✓	1	10
SNCR-2	✓	1	10
OA	×	3	8

Table 5.6.: Results for Scenario 3

The reversed order scenario shows that the proposed rule not only succeeds in perfect order, but also in a very imperfect order. Other rules need a scenario-specific configuration of the m variable.

6. Assessments of Virtual Machine Image Contents at Run-time

One essential aspect of compute services is the use of prepared VM images and virtual appliances. Both provide an operating system and an optional software stack that is available upon instantiation of a VM. While bare VM images typically provide only the operating system, virtual appliances come with a full configured software stack. The latter reduces the development time for the majority of system setups on VMs.

Meta-data that describes the contents of available VM images is however yet absent. The assessment of available VM images and virtual appliances requires an approach that can describe and document the contents as meta-data attached to VM images.

In this chapter, a method for assessing VM image meta-data is presented. The method is capable of automatically instantiating VMs of VM images to gain immediate access. Thereby, VMs can be introspected regarding installed software packages at runtime. Using the assessed data, a database of VM image meta-data can be build that serves as a basis for a range of applications, such as VM duplication or search.

The method presented in this chapter instantiates the abstract concepts of the compute service assessment framework introduced in chapter 4. An overview of the concepts reused from the framework is depicted in figure 6.1. The framework's meta-data model is extended to accommodate VM image meta-data. Moreover, the run-time assessment automation model is adapted to provide a procedure for automated VM image introspections.

A stopping rule similar to the rule presented in section 5 could be introduced for VM image assessments as well. However, a dynamically determined, premature stopping of the VM image assessment process appears unreasonable. Meta-data of VM images is typically immutable and thus can be permanently stored and retrieved when needed. Instead of that, the present method offers a parameter to adjust the number of VM images to inspect and thereby gives means for a cost budgeting.

First, a data model to describe VM image meta-data is introduced. Then, the procedure to automatically introspect VM images is presented and the involved components are explained in detail. Finally, means to compare VM images within and across compute service providers are provided.

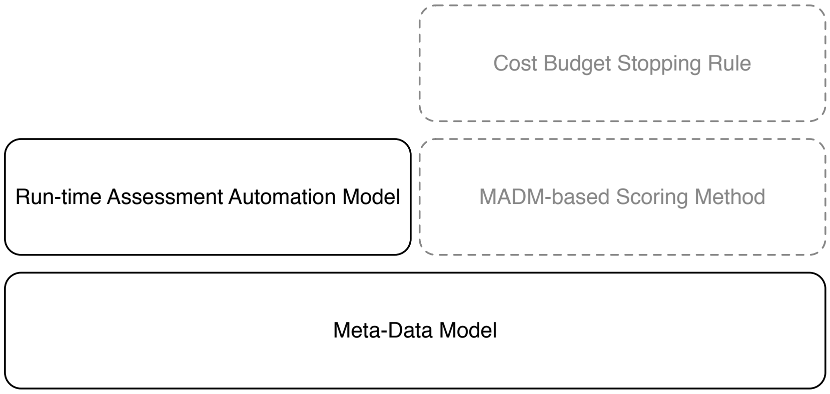

Cost Budget Stopping Rule
Run-time Assessment Automation Model
MADM-based Scoring Method
Meta-Data Model

Figure 6.1.: The Framework's Models & Methods used for VM Image Assessment

6.1. Virtual Machine Image Meta-Data Model

As a basis for future applications, the following data model shall provide a structured and queriable source of information about assessed VM image meta-data. The data model depicted in figure 6.2 extends the framework's basis meta-data model which considers a superset of meta-data available through compute cloud APIs (c.f. table 2.3). The present model adds a structure to accommodate meta-data gained from using the introspection method .

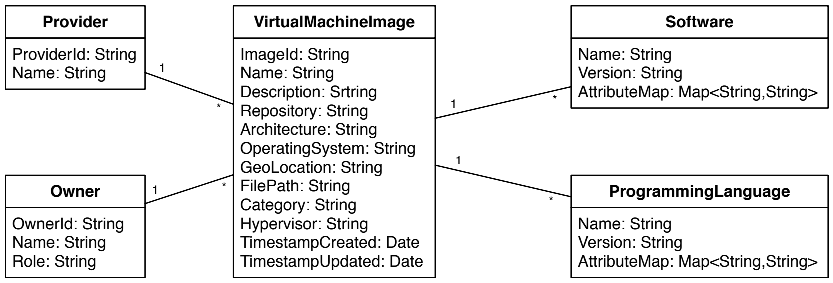


Figure 6.2.: The VM Image Meta-Data Model

The data model describes VM images as *VirtualMachineImages* which belong to a provider (*Provider*) and have an owner (*Owner*). Aside from directly associated attributes, *VirtualMachineImages* consist of software packages (*Software*) and support programming languages *ProgrammingLanguage*.

The *Software* and *ProgrammingLanguage* entities allow to hold information on the software stack and configuration retrieved via an introspection. Both, *Software* and *Pro-*

grammingLanguage, contain an *AttributeMap* attribute that allows to store arbitrary additional information as key-value pairs. Also, all data is typed as String to support all sorts of data formats as attributes like "Version" can be a number "3.0" or a non-numeric text "3.0-milestone".

6.2. Introspection of Virtual Machine Images

The following sections describe the components of the method for an automated meta-data assessment of compute services regarding available VM images and, in particular, virtual appliances with complex configurations. Particularly, a procedure for an automated introspection of VM image contents is introduced.

First, the procedure is presented to give an overview of the introduced components and their interactions. Then, the components to discover VM images in repositories and to introspect VM image contents are explained in detail.

6.2.1. Automated VM Image Introspection Procedure

Compute service providers maintain repositories of VM images. Depending on the provisioning model of the repository, the user community can publish VM images in the repository as well. The number of VM images in a repository can grow extensively large. Therefore, an assessment of VM image meta-data requires an automation. On of the goals of the procedure for automated VM image introspections is to minimize user interaction in order to execute in batches. In addition, the procedure must be capable of accessing diverse repositories from distinct compute service providers. Generally, the presented procedure is based on the blueprint automation model of the framework introduced in chapter 4.

To give means for a parameterization of a VM image introspection, the present procedure provides a user interface that allows to define the repositories and VM images to be considered. Besides, the meta-data to be assessed can be defined as a parameter, too.

The procedure incorporates multiple components that carry out different tasks in a VM image introspection. Figure 6.3 illustrates the components involved in the VM image introspection procedure.

An introspection of multiple VM images is coordinated by the main component *Automatic Introspection Coordinator*. It considers user preferences from the *User Preferences Store* that can be defined via the *User Interface* component. The user interface allows to define VM image repositories together with required credentials for remote access to a repository. Furthermore, lists of VM images (using an identifier) and filters can be specified that determine which images are considered in a meta-data assessment.

In the initial phase of a meta-data assessment, the *Discoverer* component accesses the VM image repository and and detects the set of included VM images. The set can then be matched with the user's preferences, such as blacklists, whitelists, or filters.

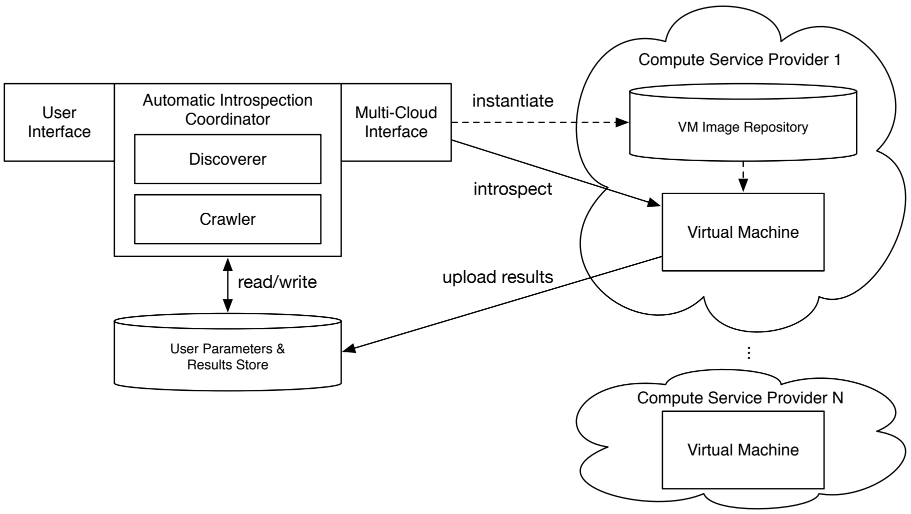


Figure 6.3.: Overview of the Automated VM Image Introspection Procedure

Based on the final set of VM images to be introspected, the *Crawler* component instantiates the images for an introspection using the *Multi-Cloud Interface*. Then, it accesses the running VMs and starts the introspection. Eventually, the results of an introspection are uploaded from within the VM to the *Results Store* which implements the VM image meta-data model.

6.2.2. Discoverer Component

The goal of the *Discoverer* component is to specify a list from available VM images in repositories and by adhering to requirements formulated by the user. Figure 6.4 depicts an overview of the process that leads to a final *discovery list*.

The *Discoverer* takes a list of *Repositories* and *Lists & Filters* as inputs from the *User Parameters Store*. A repository is a uniquely identified location where VM image meta-data of a compute service provider can be accessed. For example, Amazon EC2 offers an API that allows fetching a list of all AMIs from EC2 repositories in different regions. The Discoverer outputs a *Discovery list* for each repository, i.e. a list of identifiers of VM images that have been found in the repository and match the filter criteria.

Filters are provider-specific or even repository-specific, since different compute clouds likely provide differently named and detailed meta-data. There are two types of negative filters: (1) user-defined K.O. criteria, and (2) user-defined block-lists. K.O. criteria remove unwanted VM images from the discovery list, such as images with commercial

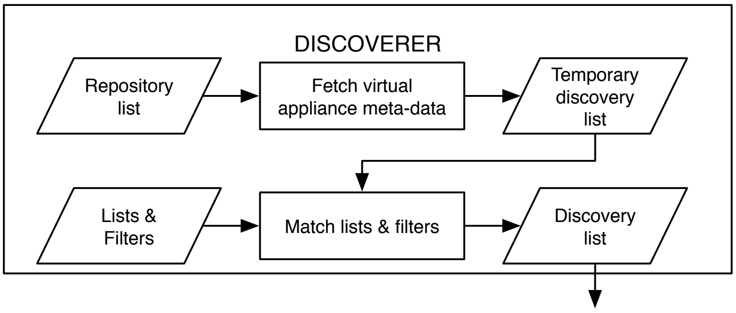


Figure 6.4.: Process of the Discoverer

software licenses. Then the identifiers in the block-list are matched with the discover list. Each match is removed from the discovery list, for example, identifiers of images that have already been introspected. Users can also define a positive filter as a requirement that needs to be fulfilled, for example, a specific operating system.

Since the discovery list can be long, the user might want to narrow down the list. Two approaches can be suggested to reduce the size of the *Discovery list*. The user either selects the desired images by their identifier (or description), or the user strips down the list to the desired coverage, either an absolute number of VM images or a fraction of the original *Discovery list*.

Additionally, a stated cost budget restrains the total number of introspections to a maximum amount of costs. The cost calculation needs to align with the actual costs generated at the Cloud provider with cost models differing in means of preciseness or billing subjects causing different costs per introspection, e.g., Amazon bills by the hour. While the cost budget would not be exceeded by an additional virtual machine instantiation the crawler proceeds with processing the discovery list.

6.2.3. Crawler Component

The *Crawler* component collects meta-data from the VM images in the *Discovery list* represented by repository-specific identifiers. Each specified VM image is introspected in the process and the results are uploaded to the *Results Store*. The name *Crawler* originates from the process of introspecting a large amount of VM images referred to as crawling. Besides, as VM image content is typically immutable, a crawling engine is able to eventually have collected all VM image meta-data in a database. However, the amount that is considered in a single assessment run can vary according to the user's requirements. Figure 6.5 gives an overview of the crawling process.

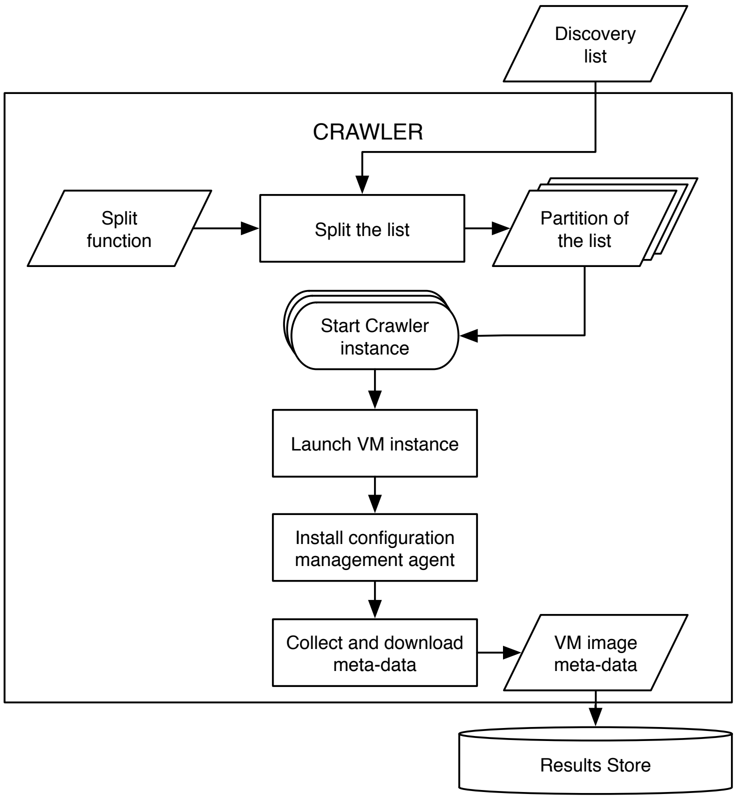


Figure 6.5.: Process of the Crawler

The *Discovery list* is processed in parallel by multiple *Crawler* instances. If the user provides a split function, the *Discovery list* can be segmented into multiple partitions. Each partition is processed by a separate *Crawler* instance. A Crawler instance can be a separate thread or process on the same server, or a separate (virtual) server.

Each Crawler instance runs the following sequential process for each VM image in a *Discover list* partition:

1. Launch an instance of the VM image (if possible)
2. Upload and install a configuration management agent on the running instance
3. Execute the agent with parameters, collect the requested meta-data and download the meta-data to a central database

The initial step employs the *Multi-Cloud Interface* to a compute service in order to launch an instance of a VM image. Typically, VM images can only be instantiated using the corresponding provider's compute services. If the instantiation of the VM image fails, the process stops and meta-data for that image cannot be assessed.

The second step requires remote access to the running VM and a channel to upload agent software. For Linux operating system instances, the agent software bundle can be uploaded with ssh. An installer script installs the agent software according to the detected operating system (i.e., Linux distribution). Of course, existing agents of configuration management software packages can be used in this step, e.g., Puppet's facter [216] or Chef Ohai [107].

In the third step, the agent is executed and collects meta-data from package manager, by executing programs that return information about the system, and by inspecting known directories. The *User Parameters Store* is consulted to determine which meta-data to collect. All collected data is uploaded by the agent to the *Results Store*.

6.3. Search and Comparison of Virtual Machine Images

Being in possession of a database of VM images meta-data gives the chance for a range of applications, such as software usage statistics and analysis of software dependency and software complexity trends. One particular aspect is the search of a VM image with specific preconfigured content and the comparison of VM images.

Unlike performance meta-data of compute services, the contents of VM images are described non-numerically. Therefore, a comparison on a numerical scale that allows statements, such as "x is two times better than y" is not feasible. A text-based comparison can only apply a matchmaking algorithm that employs string-based tests.

In a search, the available requirement types are restricted to non-numerical text-oriented requirements, as listed in table 6.1. Using a conjunctive and disjunctive satisficing method [19], alternatives can be filtered by the given requirements.

Req. Type	Boolean Expression
Equals	$\gamma = s^r$
OneOf	$\gamma \in S = \{s_1^r, \ldots, s_n^r\}$

Table 6.1.: Non-Numerical Requirement Types

For VM image meta-data, the set of contained software packages and programming languages (c.f. section 6.1) can be used in requirement definitions. A suitable VM image can thereby be searched by the contents that have been assessed with the presented introspection method.

Moreover, comparisons of VM images can be based on assessed contents as well. A comparison can be conducted in a matchmaking of VM images. Therefore, the meta-data attributes of two VM images are matched, including the set of software packages and programming languages. Consequently, the number of meta-data matches and misses indicates the similarity of the images.

6.4. Software Prototype for Virtual Machine Image Introspection

The method to assess VM image contents has been implemented as a software prototype called Crawler. The software prototype constitutes from two components: (1) the Crawler script, and (2) the Crawler web application. While the script introspects VM images to assess their contents, the web application provides a graphical user interface to browse the contents meta-data of images.

6.4.1. Crawler Script

The *Discoverer* and *Crawler* components of the VM image assessment method (c.f.) are realized in a ruby script. The script is capable of accessesing Amazon machine images (AMIs) from AWS EC2 using Opscode Chef Ohai for the collection of meta-data. AWS credentials are expected to be given in a configuration file loaded upon initialization. Also, filters can be defined that set requirements on the region, owner, price (free or priced) of an AMI. A blacklist can be defined, too.

The Crawler script uses the AWS SDK for ruby to attain the list of available AMIs from EC2. After the list has been filtered and matched with the given blacklist, each AMI is instantiated. Once instantiated, a VM is connected to via SSH. The script then downloads Chef Ohai to the VM and collects information about installed software and libraries from Ohai. Finally, the collected information is exported in JSON format to a S3 bucket and the VM is stopped. Figure 6.6 depicts the components involved in the ruby Crawler script.

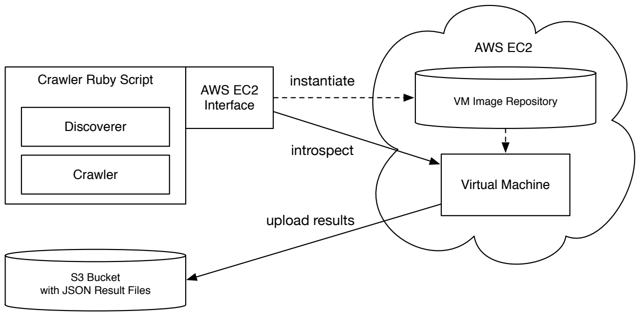


Figure 6.6.: Architecture of the Crawler Script

The source code of the Crawler script is published in a Github repository under `https://github.com/mugglmenzel/ami-crawler`.

6.4.2. Web Application

In addition to the ruby script, a web application providing a graphical user interface to browse collected meta-data was developed. The web application runs on the Google AppEngine [201] and graphical user interface was developed with GWT [202] and smartGWT [203].

Moreover, the web application is able to extract JSON files from an S3 bucket that have been generated by the Crawler script and import them into a database. An AppEngine cron triggers the import of existing JSON files from S3 once every 24h.

With the web application, the database is browsable in a sortable list and images become searchable by installed software libraries and keywords. A superset of software libraries is maintained by the application and presented for selection during a search. Figure 6.7 shows a screenshot of the web application.

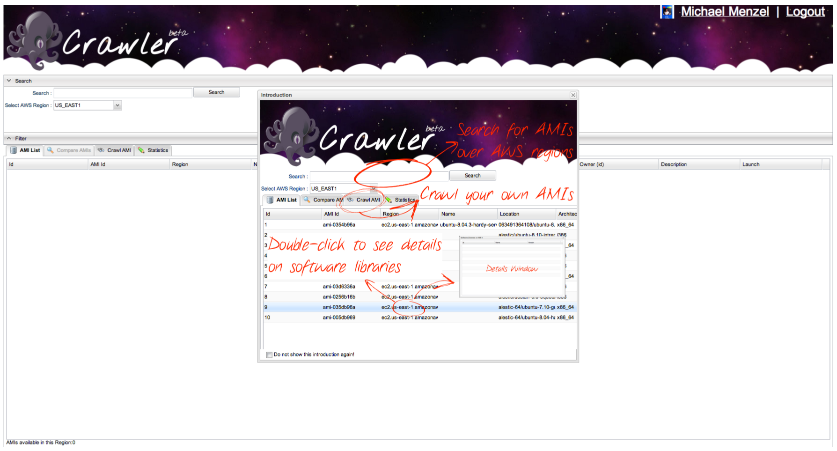

Crawler beta
Michael Menzel | Logout
Search for AMIs over AWS regions
Crawl your own AMIs
Double-click to see details on software libraries
Details Window
Search
Select AWS Region : US_EAST1
AMI List
Compare AMIs
Crawl AMI
Statistics
Do not show this introduction again!
AMIs available in this Region:0

Figure 6.7.: Screenshot of the Crawler Web Application

The source code of the Crawler web application is released to the public in a Github repository under `https://github.com/mugglmenzel/Crawler`.

6.5. Evaluation of the VM Image Introspection Method

The proposed VM image introspection method provides an automated approach to assess contents of images by introspecting a running VM. In respect of $RQ4$, an evaluation has to test the applicability of the proposed method:

RQ4. How can Virtual Machine image contents be assessed at run-time from multiple compute services in an automated manner?

The instantiation of the VM image introspection method in a software prototype (c.f. section 6.4) proofs the approach's feasibility. The method and the software prototype has been subject to discussions with peers in the research community [217]. This work presents the validation of the method conducted by employing the software prototype in an example with diverse VM images. Besides, the promptitude of automated VM image introspections is measured.

In the following, the first section explains how the validation with an introspection of various VM images takes place. Then, the findings of the promptitude experiments with the software prototype are presented. The final section provides statistics derived from meta-data about the AWS EC2 compute service collected with the software prototype.

6.5.1. Validation of the VM Image Introspection Approach

The validation aims at verifying the feasibility and applicability of the approach. In particular, an experiment which seeks to introspect the contents of diverse VM images in the repository of the compute service AWS EC2. The experiment is conducted to research the following hypotheses in the light of *RQ4*:

H_0 The introspection approach respectively the software prototype fails to collect meta-data about contents of a VM image.

H_1 The introspection approach respectively the software prototype collects meta-data about contents of a VM image.

While the null hypothesis H_0 holds true if the introspection approach fails, the hypothesis H_1 is proofed when meta-data describing the contents of a VM image were assessed successfully. To test the hypothesis, an experiment with the software implementation of the introspection approach is conducted. In the experiment an exemplary introspection seeks to proof the applicability of the software prototype (see section 6.4) by introspecting a range of VM images from the compute service AWS EC2.

In preparation of the experiment, the software prototype has been fed with a unknown AMIs list of four different EC2 Ubuntu Linux images from Bitnami [67]. Namely, the prepared VM images from Bitnami are a Wordpress blog software machine (ami-14c5277c), a LAMP stack machine (ami-f85fa990), a JBoss application server machine (ami-c79588ae), and a Gitlab versioning software machine (ami-1203fe7a). Using the AMIs list, the software prototype was executed in three repeated runs per image which is to assure repeatable results.

The software prototype uses the Chef Ohai client to collect system meta-data and installed libraries and software. Upon completion, the prototype exports the introspected VM image content meta-data into JSON files on the cloud storage service AWS S3.

To verify the correctness of the detected VM image contents, the exported JSON files were compared to the actually installed list of VM images retrieved with the following shell command:

```
sudo dpkg --get-selections
```

For all of the tested VM images, the comparison proofed the approach is capable of collecting VM image content meta-data. The exported JSON files were identical in every of the repeated runs. Therefore, hypothesis H_1 is proofed and a positive statement in the light of the research question *RQ4* can be made. Using the proposed approach implemented in the software prototype, the contents of diverse VM images could be assessed at run-time. Besides, an evidence of the feasibility of *RQ4* proofs also the existence of automated assessments in general. Thereby, *RQ1a* is addressed in parallel.

The experiment is based on the software prototype using a prior configuration with a list of AMIs. Given the configuration, the execution ofthe introspection task was executed in an automated manner. Therefore, the level of automation achieved by the method and software prototype is classified as *batch processing* according to Endsley [193].

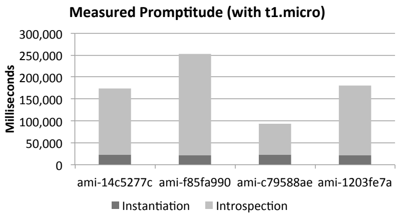
6.5.2. Results of the Promptitude Measurements

Apart from a validation of the approach, promptitude measurements give insights in the applicability in a practical context. In order to record the promptitude of an introspection, the validation experiments with four different VM images have been extended with a time measurement. Additionally, the introspection has been tested for two different compute service configurations, namely the AWS EC2 instance types *t1.micro* and *m1.small*.

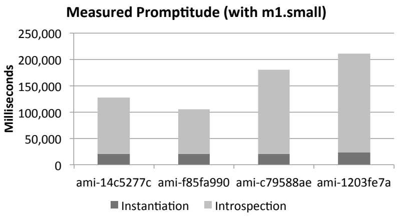
In the experiments, the duration of the whole introspection process was measured in two stages: (1) VM image instantiation, and (2) introspection. Thereby, the whole duration of the VM image contents assessement process results from the sum of the measured durations in both stages. As in the validation, the four EC2 AMIs from Bitnami [67] with Wordpress, LAMP, JBoss, and Gitlab are used. To ensure reliable results, the experiments were repeated three times.

Figures 6.8a and 6.8b depict the promptitude measurement results for the *t1.micro* and *m1.small* instance types, respectively. The graph shows that the instantiation is responsible for a small share of the total duration. In average, the instantiation required 22,443 ms for micro instances and 21,280 ms for small instances.

The introspection stage induces the major delay in the total promptitude ($\sim 87\%$ micro instances, $\sim 86\%$ small instances). The measurement results show that an introspection of micro instances took 152,767 ms and of small instances 134,133 ms in average. The available compute power has a direct influence on the promptitude of the introspection stage. Observations with system monitors discovered that system processes after the start of the system can induce a delay.



(a) Promptitude Results for Micro Instances



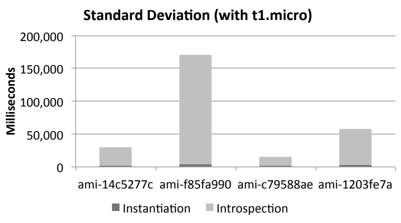
(b) Promptitude Results for Small Instances

From the recorded measurements, a variation in the promptitude was observed for different compute services. As depicted in figures 6.9a and 6.9b, the standard deviation is variable for micro and small instances. For micro instances, a maximum standard deviation of 167,036 ms ($\sim 72\%$) and minimum of 13,952 ms ($\sim 20\%$) for the introspection stage can be observed. For small instances, the maximum standard deviation of the introspection stage is 85,278 ms ($\sim 54\%$) and the minimum standard deviation is 15,980 ms ($\sim 19\%$).

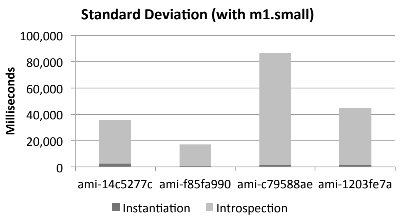
All in all, the promptitude show variations which differ in gravity between compute service. Overall, the measured promptitude for a VM image content assessment process never exceeded 401 seconds in total in all experiments.

6.5.3. Statistics for AWS EC2 Compute Services

Apart from assessing the contents of few selected VM images, the method can also be employed to gain statistics about whole repositories. As an extension to the hitherto evaluation of the introspection approach, a wide range of VM images provided in the



(a) Promptitude Standard Deviation for Micro Instances (in ms)



(b) Promptitude Standard Deviation for Small Instances (in ms)

Ubuntu package name	Installation count
perl	863
python	710
ruby1.8	399
python2.6	247
python2.7	263
openssh-server	869
vim-tiny	843
vim	502
subversion	23
git-core	8

Table 6.2.: Software Package Count (Canonical and Alestic.com)

EC2 repositories by Alestic.com and Canonical have been assessed. Both, Alestic.com and Canonical, prepare AMIs with varying Ubuntu Linux operating systems and offer them to the community of EC2 users for free.

To gain statistics, 869 of 981 AMIs from Alestic.com and Canonical were assessed with the software prototype implementing the introspection method. The other 112 AMIs could not be assessed due to problems over establishing a SSH connection (with users root, ubuntu, ec2-user).

Based on the assessed VM image contents, the frequency of certain software in Ubuntu Linux images on EC2 can be determined. Table 6.2 presents the frequency statistics for software packages in EC2 Ubuntu AMIs from Alestic.com and Canonical. The upper part of the table shows the installation count of the top five programming language packages, and the bottom part the installation count of the top five software packages.

Apart from observed software frequencies, further statistics could be generated from the meta-data assessment results. One particularity of the explored compute service EC2 is the separation of the global VM image repository into sub-repositories in diverse

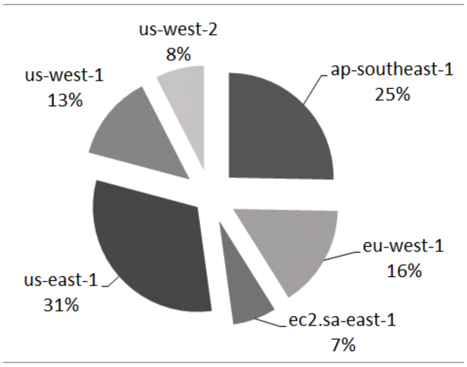
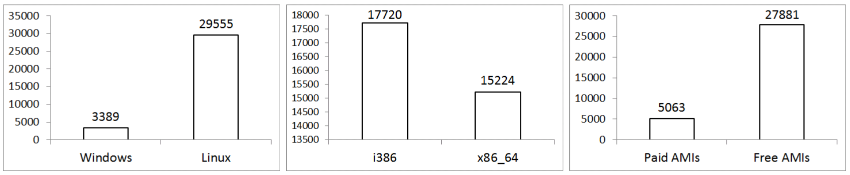


Figure 6.10.: Number of AMIs per Amazon EC2 Region

regions. To gain an understanding how the regions compare, the number of AMIs in the diverse regions has been determined via the EC2 API. The results are illustrated in a pie chart in figure 6.10.

Moreover, AMIs available in EC2 (aggregated over all regions) could be grouped by the meta-data attributes *operating system*, *processor architecture*, and *commercial license*. The results from the grouping by meta-data attribute are presented as bar charts in figure 6.11. The numbers show clearly that Linux is the predominant operating system in available AMIs. Also, most AMIs are based on a i386 32-bit processor architecture. Interestingly, the larger share of AMIs is provided for free. However, software in the AMIs may still require a license (referred to as *bring your own license*).



(a) Operating System (b) Processor Architecture (c) Commercial License

Figure 6.11.: Meta-Data Attributes of AMIs in All Regions

Part IV.

Finale

7. Conclusions, Discussion & Outlook

This final chapter completes the present work with a retrospection and final thoughts. First, the findings, research outcomes and contributions of this work are summarized and conclusions are drawn. Finally, the contributions of the present work are discussed and an outlook with open research questions is given, which leaves further explorations to future work.

7.1. Conclusions

This work started with the observation that compute service offerings lack comprehensive meta-data to support detailed comparisons and engineering decisions. The meta-data published by compute service providers is scarce and not necessarily reliable, particularly to describe performance characteristics and contents of VM image repositories. Both aspects have been of eminent interest in this work.

In regards of performance, the amount of meta-data is specifically limited when a cloud consumer needs to identify suitable compute services providing the necessary capacity for a software system. Also, tools are missing when a consumer seeks to monitor vicissitudes of the provided performance characteristics of a service.

The current state of the art lacks an automated approach to measure the performance of a compute service according to schedules and over time. Approaches which offer automation aspects and thereby are able to repeat and schedule benchmarking runs have not yet been adapted to target compute services. In contrast, approaches to benchmark compute services lack an automation, and need to be manually adapted to diverse compute services.

Moreover, in the realm of assessing VM image meta-data from compute services, existing approaches expect full image file access and do not gain meta-data about contents. The research community has already explored machine introspections for security purposes which, however, does not incorporate support for meta-data collection or access through remote interfaces of compute services.

The identification of a scarcity in reliable compute service meta-data lead to the following research question:

> *How can cloud compute services be assessed*
> *at run-time regarding meta-data?*

The principal research question is split into several subordinated questions which this work aims to answer and evaluate. In this regard, a framework of methods and models to assess compute service meta-data in an automated manner has been proposed. The methods of the framework leverage client-side remote access and collect meta-data from within VMs at run-time. In addition, the framework features support for comparisons of compute services based on a scoring. Thereby, compute cloud consumers are enabled to single-handedly assess and compare meta-data attributes of compute services.

The framework is furthermore instantiated to treat specifically two aspects of compute services, namely performance and VM image contents. Consequently, compute cloud consumers are provided with concrete methods to assess and compare compute services according to both aspects.

In the following, a summary of this work's contributions is given and conclusions are drawn.

Automated Compute Service Assessments
As part of the framework a blueprint model for the automation of assessments is proposed. The model describes necessary steps assess compute services at run-time and a scheme for the communication with compute services via remote protocols. Based on the automation model two instantiations are presented that enable consumers to assess compute services in an automated manner regarding two specific characteristics: (1) performance, (2) VM image contents.

To gain performance meta-data of compute services, the model is reused in a method to measure performance of compute services at run-time. A proposed performance measurement method supports a scheduling and repetition of measurement tasks to observe the performance behavior of compute services over time. Exploiting remote access offered by compute services, the approach executes benchmarking scripts from the client-side to conduct measurements. Collected performance benchmarking results are structured and attached to compute services as meta-data using the framework's meta-data model. An implementation of a software prototype shows the feasibility of the method and serves as the basis for further evaluations. The aspects of automation, scheduling and repetition of compute service performance benchmarking has been validated and compared to the state of the art. Besides, an examination of the promptitude of the software prototype showed that the induced overhead to prepare VMs for benchmarking at run-time never surpasses 220 seconds and, thereby, confirms the applicability in practice. The scheduling component of the prototype sustains an average impreciseness of 9 milliseconds in execution delay (with a maximum of 24 milliseconds).

The second instantiation of the automation model enables consumers to automatically assess the contents of VM images provided in repositories of compute services. Particularly, the approach introspects the software libraries and packages included in a VM image from a client at run-time. A VM image is instantiated as VM on the corresponding compute service and accessed via remote protocols. With an established remote connection in place, information from package managers and other operating system tools is extracted from the running VM. The list of detected software contents is attached to a VM image and stored as meta-data in a central database. An evaluation with a software prototype proofed the feasibility of the concept and revealed that an introspection of AWS EC2 VM images never surpasses 401 seconds according to a range of experiments.

Compute Service Scoring & Comparisons
Based on the meta-data gained from automated assessments, a further method of the framework introduces a scoring and, thereby, enables consumers to compare compute services according to a single value. The method combines multiple meta-data attributes into a single score value. The calculation is based on a weighted criteria hierarchy linked to the performance attributes and uses the Analytic Hierarchy Process. The resulting scores are normalized and mapped to a ratio scale for comparisons. An instantiation for performance assessments of compute services adapt the method for comparisons according to performance attributes. A software prototype implementing the performance assessment instantiation proofs the feasibility and served as basis for a case study with the industry. In addition, the computational complexity of the method has been examined for growing numbers of compute services and performance attributes to be considered. While doing so, the influence of the structure of criteria hierarchies on the computational complexity has been explored. The findings indicate that with a growing number of attributes, grouping should be considered to decrease the computational complexity. In realistic scenarios with no more than 30 performance attributes and compute services, the software prototype is able to calculate the weighted and normalized score values with 5-digit precision within 3.6 milliseconds.

Cost Budgeting Stopping Rule
To counter costs accruing from assessments of compute services, the framework incorporates a stopping rule approach. The stopping rule binds the maximum number of observable compute services to a given cost budget. In an instantiation for performance measurements, the framework's stopping rule is configured to support sequential observations of compute services with performance benchmarks. An implementation of the instantiations stopping rule asserts the feasibility of the approach. Moreover, experiments proof that the rule guarantees to stop the execution of performance benchmarking runs before a certain cost budget is spent. Apart from the budget, the stopping rule pursues minimizing costs when a prior order of the compute services regarding performance scores is provided. According to a comparison with other stopping rules in the course of the evaluation, the rule outperforms or equals the spendings of standard stopping rules when the set of compute services is in perfect or near perfect order.

Overall, the framework gives answers to the posed research questions and contributes a framework of evaluated models and methods to the research community. The introduced methods give means to assess and compare compute services regarding diverse characteristics. Instantiations apply the framework's methods and models to assess compute services in terms of performance and VM image contents.

An implementation of the methods in multiple software prototypes proofed the correctness and feasibility of the concepts and applicability in practice. The experiment results from the software prototypes served as the basis for an evaluation of the research questions. Existent limitations are subject to a discussion and might be addressed by future work – which is the matter of the next section.

7.2. Discussion & Outlook

The present work contributes a yet applicable framework of compute service assessment models and methods to the research community. Despite the fact that the findings of an evaluation indicate a validity and applicability of the methods, limitations are due to be discussed. Besides, there is certainly plenty of room for improvements, extensions and a dissemination of the results. In the following, the limitations and aspects of future work are discussed for each of the methods incorporated in the framework and particularly their instantiations.

Automated Compute Service Assessments
The automation model is strictly bound to compute services with accessible remote interfaces integrated into an implementation. The assumption made is that the remote interfaces support a completely automated management of VMs and the establishement of remote sessions to them. If any of the steps requires manual interaction on the client side, a full batch execution cannot be realized and the level of automation is lowered.

The instantiations of the automation model raise further discussable points. It must be mentioned that the measurement of performance attributes of a compute service using benchmarking scripts at run-time is tied to two aspects: (1) exploitation of remote access interfaces to VMs and (2) benchmarking based on synthetic workloads. As benchmarking only captures performance characteristics regarding synthetic workloads, other measurement approaches might be more suitable for different performance analysis scenarios. For example, an in-situ method can gain insights with the actual software in place and non-synthetic workloads.

However, an in-situ measurement requires a deployment of the software system and users to generate the workload. For an automation of the measurement, the deployment of the software system needs to be automated in the process, too. Besides, when actual users generate the in-situ workload, a measurement cannot be initiated ad-hoc, but is carried out over the long-term. Otherwise, a workload generator needs to be prepared for this purpose, such as Rain [218].

In case both issues are addressed, namely the introduction of an automated software deployment and a workload generator, in-situ measurements may well be integrated into the proposed performance measurement method. Nevertheless, such an integration is subject to future work.

In contrast, measurements with a simulation or emulation method are contrary to the proposed method. Both, simulation and emulation, void the access to the real system, but require to build an imitation of the system. Typically, the details of a system under test, being a compute service, are typically hidden from a consumer and treated as a business secret by the provider. Therefore, building an imitation of the compute service that resembles the behavior is a rather implausible undertaking.

Moreover, an examination of the software prototype's promptitude demonstrated that performance benchmarking requires minutes or hours of time per run. For popular use cases, such as capacity planning for a software system, first performance meta-data being available within hours is acceptable. In other use cases, a more responsive or even real time performance monitoring may be of interest. With the proposed run-time benchmarking approach, real time performance meta-data seems not achievable. Nonetheless, a reduction of the overhead and measures to increase the promptitude are subject to future work. Furthermore, a use of the approach in a server-side setting driven by compute service providers is a yet unexplored alternative.

A further aspect of compute services reveals a limitation of the approach. The computer hardware and resource allocation algorithms may change in quick iterations over time. The hardware purchased by cloud providers to expand and maintain data centers advances together with technological progress. A benchmarking approach from a consumer perspective (with client-side access) faces a black box and can only capture performance characteristics at a certain point in time. An exact and timely representation of a compute service's performance specification may rather be achieved by more costly monitoring approaches, used on the client or server-side.

Despite CloudHarmony [139] having built a first database of performance meta-data, one future effort is to build databases that host large amounts of compute service performance meta-data. With the methods and software prototypes presented in this work, the measurements can be conducted by many cloud consumers in a joint effort. We plan to establish the software prototype as a widely used web application which enables any compute cloud consumer to conduct performance measurements. The possibility to configure repetitions and schedule executions of performance benchmarks gives consumers further power and insights how the compute services' performances vary and evolve over time. The meta-data creates the need for applications that can aggregate and illustrate results from multiple repetitions and allow users to browse through performance meta-data histograms.

As with the automated performance benchmarking method, the automated VM image introspection method leverages the run-time and operates from a client-side. A certain overhead induces a delay which affects the promptitude of VM introspections. The aiming for real time results from introspections using the proposed method seems unlikely in this case as well.

A comprehensive database of meta-data about VM image contents of diverse compute services is a future task that implicates extensive costs. To counter the costs, a joint effort from the community of cloud consumers is needed. Fortunately, each VM image is typically immutable and needs to be introspected only once. We plan to manifest the software prototype as a public web application that is used by many compute cloud consumer to introspect VM images.

The future of VM image repositories is yet blurry. The number of engaged institutions creating VM images for community-driven repositories, such as with AWS EC2, indicate an immense growth in the coming years. Aside from that, AWS has launched a marketplace for commercial VM images which could add to the growth rate in the future. The assessment of all VM images in a timely manner could become a Sisyphean task in the future. However, the proposed approach allows compute cloud consumers to focus on few VM images they have identified as particularly interesting. On the contrary, compute cloud providers can integrate the introspection method within their compute services on the server-side to take over the task for their consumers.

Compute Service Scoring & Comparisons
The scoring method proofed as applicable in practice according to a case study and interviews with the industry. As stated by users of the software prototype, MADM provides a suitable approach for comparisons of alternatives. A further validation of the prototype confirms that performance attributes of compute services can be mapped to a single score. The computational complexity, however, is growing exponentially. Nonetheless, computation times are acceptable with typical numbers of compute services and performance attributes and never exceeded 3.6 milliseconds in realistic scenarios. While the AHP allows for very precise comparisons on a ratio scale, simpler approaches, such as lexicographic methods that focus on one or few high priority criteria, may be sufficient as well.

A major task for the future is to establish standard scorings with fixed sets of performance attributes, provided hierarchies and predetermined weights. Thereby, inexperienced decision-makers and consumers can start with a common standard scoring. Nonetheless, an obvious merit of the AHP approach is that given the database of performance meta-data, every consumer can compare compute services with a custom configuration of selected attributes, weights and hierarchy structure.

Cost Budgeting Stopping Rule
The presented stopping rule is specifically adapted the situation that prior information is available. Only then the stopping rule is surely successful and can outperform standard stopping rules. Nevertheless, as the experiments with common scenarios indicate, the proposed stopping rule can outperform or equalize standard stopping rules in many cases.

Moreover, the rule ensures a cost budget is never exceeded within the sequential assessment process. However, the proposed stopping rule does not utilize the given budget to a full extent. Bin packing algorithms promise an approach to optimally use the given cost budget. Moreover, the presented stopping rule lacks a support for a time budget.

Apart from a time budget, a parallelization of the yet sequential assessment process is a promising approach to overcome time budgets. Observations, however, are then made in parallel and, thus, potentially in random order. A stopping rule would need to cope with randomly made observations of a pre-ordered compute service set, and still enforce a cost budget.

Final Remarks
Overall, this work provides a yet applicable and complete solution in regards of the initially observed problem and posed research questions. The methods of the proposed framework have been discussed with peers in the research community and throughout this document. Although the methods and prototypes arrived at a complete state in regards of the addressed problems, they leave room for enhancements, extensions, and competing approaches.

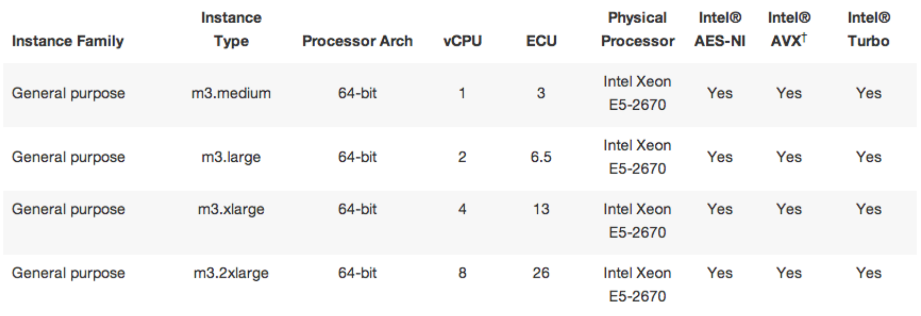
Specific extensions can be made in the realm of additional compute service aspects yet to be assessed. Further instantiations of the framework can provide consumers with means to assess and compare compute services regarding an extended range of attributes.

Hopefully, this work engages other researchers to devote energy into the development of meta-data-related approaches to further enrich and simplify the evaluation, choice, comparison, and consumption of cloud compute services for consumers.

Appendix

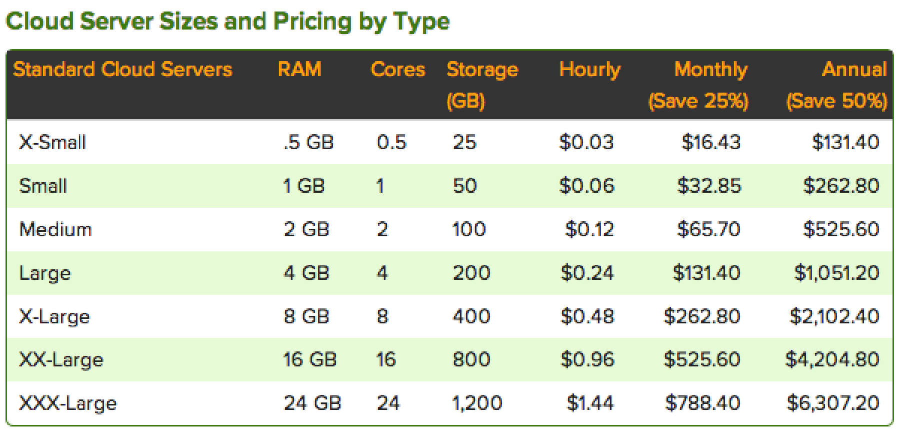
A. Performance Meta-Data on Compute Service Websites

The websites have been accessed on March 23, 2014.



Instance Family	Instance Type	Processor Arch	vCPU	ECU	Physical Processor	Intel® AES-NI	Intel® AVX†	Intel® Turbo
General purpose	m3.medium	64-bit	1	3	Intel Xeon E5-2670	Yes	Yes	Yes
General purpose	m3.large	64-bit	2	6.5	Intel Xeon E5-2670	Yes	Yes	Yes
General purpose	m3.xlarge	64-bit	4	13	Intel Xeon E5-2670	Yes	Yes	Yes
General purpose	m3.2xlarge	64-bit	8	26	Intel Xeon E5-2670	Yes	Yes	Yes

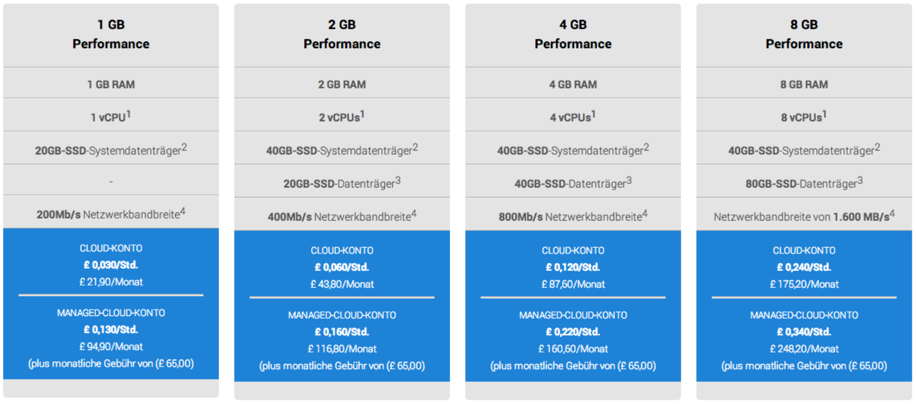
Figure A.1.: AWS EC2 Performance Meta-Data



Cloud Server Sizes and Pricing by Type

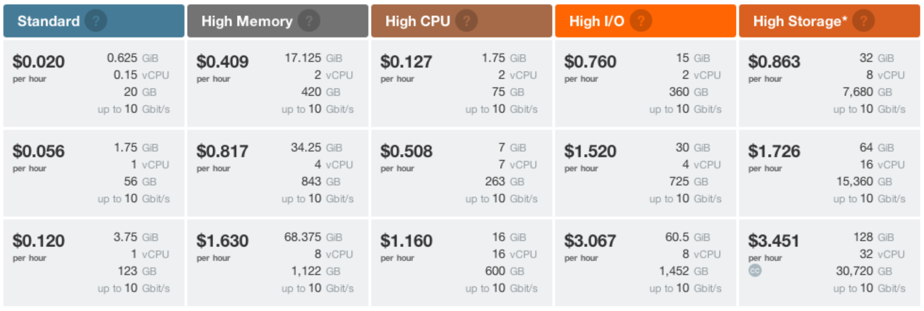
Standard Cloud Servers	RAM	Cores	Storage (GB)	Hourly	Monthly (Save 25%)	Annual (Save 50%)
X-Small	.5 GB	0.5	25	$0.03	$16.43	$131.40
Small	1 GB	1	50	$0.06	$32.85	$262.80
Medium	2 GB	2	100	$0.12	$65.70	$525.60
Large	4 GB	4	200	$0.24	$131.40	$1,051.20
X-Large	8 GB	8	400	$0.48	$262.80	$2,102.40
XX-Large	16 GB	16	800	$0.96	$525.60	$4,204.80
XXX-Large	24 GB	24	1,200	$1.44	$788.40	$6,307.20

Figure A.2.: GoGrid Performance Meta-Data



1 GB Performance	2 GB Performance	4 GB Performance	8 GB Performance
1 GB RAM	2 GB RAM	4 GB RAM	8 GB RAM
1 vCPU[1]	2 vCPUs[1]	4 vCPUs[1]	8 vCPUs[1]
20GB-SSD-Systemdatenträger[2]	**40GB-SSD**-Systemdatenträger[2]	**40GB-SSD**-Systemdatenträger[2]	**40GB-SSD**-Systemdatenträger[2]
-	**20GB-SSD**-Datenträger[3]	**40GB-SSD**-Datenträger[3]	**80GB-SSD**-Datenträger[3]
200Mb/s Netzwerkbandbreite[4]	**400Mb/s** Netzwerkbandbreite[4]	**800Mb/s** Netzwerkbandbreite[4]	Netzwerkbandbreite von **1.600 MB/s**[4]
CLOUD-KONTO **£ 0,030/Std.** £ 21,90/Monat	CLOUD-KONTO **£ 0,060/Std.** £ 43,80/Monat	CLOUD-KONTO **£ 0,120/Std.** £ 87,60/Monat	CLOUD-KONTO **£ 0,240/Std.** £ 175,20/Monat
MANAGED-CLOUD-KONTO **£ 0,130/Std.** £ 94,90/Monat (plus monatliche Gebühr von (£ 65,00)	MANAGED-CLOUD-KONTO **£ 0,160/Std.** £ 116,80/Monat (plus monatliche Gebühr von (£ 65,00)	MANAGED-CLOUD-KONTO **£ 0,220/Std.** £ 160,60/Monat (plus monatliche Gebühr von (£ 65,00)	MANAGED-CLOUD-KONTO **£ 0,340/Std.** £ 248,20/Monat (plus monatliche Gebühr von (£ 65,00)

Figure A.3.: Rackspace Performance Meta-Data



Standard ?	High Memory ?	High CPU ?	High I/O ?	High Storage* ?
$0.020 per hour; 0.625 GiB, 0.15 vCPU, 20 GB, up to 10 Gbit/s	**$0.409** per hour; 17.125 GiB, 2 vCPU, 420 GB, up to 10 Gbit/s	**$0.127** per hour; 1.75 GiB, 2 vCPU, 75 GB, up to 10 Gbit/s	**$0.760** per hour; 15 GiB, 2 vCPU, 360 GB, up to 10 Gbit/s	**$0.863** per hour; 32 GiB, 8 vCPU, 7,680 GB, up to 10 Gbit/s
$0.056 per hour; 1.75 GiB, 1 vCPU, 56 GB, up to 10 Gbit/s	**$0.817** per hour; 34.25 GiB, 4 vCPU, 843 GB, up to 10 Gbit/s	**$0.508** per hour; 7 GiB, 7 vCPU, 263 GB, up to 10 Gbit/s	**$1.520** per hour; 30 GiB, 4 vCPU, 725 GB, up to 10 Gbit/s	**$1.726** per hour; 64 GiB, 16 vCPU, 15,360 GB, up to 10 Gbit/s
$0.120 per hour; 3.75 GiB, 1 vCPU, 123 GB, up to 10 Gbit/s	**$1.630** per hour; 68.375 GiB, 8 vCPU, 1,122 GB, up to 10 Gbit/s	**$1.160** per hour; 16 GiB, 16 vCPU, 600 GB, up to 10 Gbit/s	**$3.067** per hour; 60.5 GiB, 8 vCPU, 1,452 GB, up to 10 Gbit/s	**$3.451** per hour; 128 GiB, 32 vCPU, 30,720 GB, up to 10 Gbit/s

Figure A.4.: Joyent Cloud Performance Meta-Data

B. Implementation Details & Code Examples

B.1. Example of AotearoaLib Programming Interface

```
Decision<Alternative> decision =
            new Decision<Alternative>();
decision.setName("Compute␣Service␣Provider␣Selection");

Goal g1 = new Goal("Costs");
g1.setGoalType(GoalType.NEGATIVE);
Goal g2 = new Goal("Performance");

// Costs
Criterion c11 = new Criterion("Hourly␣Costs");
Criterion c12 = new Criterion("Initial␣Costs");
g1.addChild(c11);
g1.addChild(c12);

// Performance
Criterion c21 = new Criterion("LINPACK");
Criterion c22 = new Criterion("WHETSTONE");
g2.addChild(c21);
g2.addChild(c22);

// defining Alternatives
Alternative a1 = new Alternative();
a1.setName("Amazon␣Web␣Services␣EC2");
Alternative a2 = new Alternative();
a2.setName("Rackspace␣Cloud");
Alternative a3 = new Alternative();
a3.setName("Terremark␣vCloud");

decision.addGoal(g1);
decision.addGoal(g2);

decision.addAlternative(a1);
decision.addAlternative(a2);
decision.addAlternative(a3);


AnalyticHierarchyProcess ahp =
            new AnalyticHierarchyProcess(decision);
```

```
        decision.getImportanceGoals(GoalType.NEGATIVE).add(
                new GoalImportance(0, 1, 3D, null));
        decision.getImportanceGoals(GoalType.POSITIVE).add(
                new GoalImportance(0, 2, 3D, null));

        double[][] crit1 = { { 1, 1 }, { 1, 1 } };
        Matrix ccrit1 = new Matrix(crit1);
        ahp.setChildrenCriteriaWeights(g1, ccrit1, 15);

        double[][] crit2 = { { 1, 2 }, { 0.5, 1 } };
        Matrix ccrit2 = new Matrix(crit2);
        ahp.setChildrenCriteriaWeights(g2, ccrit2, 15);


        List<Evaluation> evals = new ArrayList<Evaluation>();
        // Costs Goal Evaluation
        Evaluation ev = new Evaluation();
        // Hourly Costs
        double crit11[][] = {
                { 1, 2D / 3D, 4D / 5D },
                { 3D / 2D, 1, 6D / 5D },
                { 5D / 4D, 5D / 6D, 1 } };
        Matrix crit11M = new Matrix(crit11);
        ev.getEvaluations().add(crit11M);

        // Initial Costs
        double crit12[][] = {
                { 1, 6, 1D / 3D },
                { 1D / 6D, 1, 1D / 8D },
                { 3, 8, 1 } };
        Matrix crit12M = new Matrix(crit12);
        ev.getEvaluations().add(crit12M);

        evals.add(ev);

        // Performance Goal Evaluation
        ev = new Evaluation();

        // LINPACK
        double crit21[][] = {
                { 1, 4, 8 },
                { 1D / 4D, 1, 1D / 2D },
                { 1D / 8D, 1D / 2D, 1 } };
        Matrix crit21M = new Matrix(crit21);
        ev.getEvaluations().add(crit21M);

        // WHETSTONE
        double crit22[][] = {
                { 1, 1, 1 },
                { 1, 1, 1 },
                { 1, 1, 1 } };
        Matrix crit22M = new Matrix(crit22);
        ev.getEvaluations().add(crit22M);

        evals.add(ev);

        EvaluationResult results = ahp.evaluateFull(evals);


        System.out.println("Multiplicative:␣"
                + results.getResultMultiplicativeIndexMap());
        System.out.println("Additive:␣"
                + results.getResultAdditiveIndexMap());
        System.out.println("Positive:␣"
                                + results.getResultPositiveGoalsMap());
                        System.out.println("Negative:␣"
                                + results.getResultNegativeGoalsMap());
```

B.2. Excluded Micro-benchmarks in LibIntelliCloudBench

The LibIntelliCloudBench library excludes all micro-benchmarks of the category "Graphics" from the Phoronix test suite. Additionally, the blacklist comprises the following benchmarks named by their Phoronix internal identifier:

- aio-stress-1.1.0
- battery-power-usage-1.0.0
- bork-1.0.0
- bullet-1.1.0
- compilebench-1.0.0
- encode-ogg-1.4.0
- encode-wavpack-1.2.0
- espeak-1.3.0
- etqw-demo-iqc-1.0.0
- ffte-1.0.1
- gcrypt-1.0.0
- gnupg-1.3.1
- hdparm-read-1.0.0
- idle-1.1.0
- idle-power-usage-1.0.0
- interbench-1.0.1
- java-scimark2-1.1.1
- jgfxbat-1.1.0

- juliagpu-1.2.0
- luxmark-1.0.0
- mandelbulbgpu-1.2.0
- mandelgpu-1.2.0
- mencoder-1.3.0
- nexuiz-iqc-1.0.0
- npb-1.1.1
- pgbench-1.4.0
- pyopencl-1.0.0
- smallpt-gpu-1.2.0
- scimark2-1.1.1
- sqlite-1.8.0
- stresscpu2-1.0.1
- sunflow-1.1.0
- systester-1.0.0
- tachyon-1.1.0
- tscp-1.0.0
- ttsiod-renderer-1.3.0
- xplane9-iqc-1.0.0

C. Original Interview Form for a MADM Software Prototype

On the following pages, the structured interview originally developed for the decision-makers of the industry partner T-Systems. Due to time restrictions, the original feedback form was not filled out by the decision-makers. Instead an unstructured interview was conducted.

„Aotearoa" Software Prototype Feedback Form

Please fill out this form to give feedback regarding your personal process of defining a Decision Template. The form uses the method of semi-structured interviews and consists of several questions that allow to record all issues you faced during your definition process (of the Decision Template). In particular, two types of questions are employed to capture your personal experiences: rating questions that ask for a rating from 1 to 9, and free text questions that offer to write down all experiences (no limitations).

Role of participant in the decision-making process:

____

Transparency

How is your experienced level of transparency in the decision-making process by using the prototype (1: intransparent – 9: transparent): ___ [1-9]

Experiences and issues regarding the transparency in the decision-making process (How is the perceived transparency in the decision-making process when using the prototype?):

____

____

Speed

Please estimate the time needed to finish the decision-making process (without re-evaluation): ___ minutes

Experiences and issues regarding the speed of the decision-making process (How is the perceived time spent in the decision-making process when using the prototype?):

____

____

Trust and Correctness

How much do you trust in the correctness of the results of the decision-making process when using the prototype (1: untrusting – 9: trusting): ___ [1-9]

Experiences and issues regarding the correctness of the decision-making process (How is the perceived correctness in the decision-making process when using the prototype?):

____

____

How would you compare the Aotearoa decision-making process to the regular, original process in your organization?

Which decision-making process is more transparent?: ________ [original or Aotearoa]

How much more transparent would you rate the more transparent decision-making process (1: equally fast – 9: much more transparent)?: ___ [1-9]

Which decision-making process is faster?: ________ [original or Aotearoa]

How much faster would you rate the faster decision-making process (1: equally fast – 9: much faster)?: ___ [1-9]

Which decision-making process results do you trust more regarding correctness?: ________ [original or Aotearoa]

How much more trustful would you rate the more trustful decision-making process (1: equally trustful – 9: much more trustful)?: ___ [1-9]

What is your personal level of experiences with the original decision-making process (1: beginner – 9: expert)?: ___ [1-9]
(Comment: You can use indicators such as frequency of involvement in the original process, or last time being involved)

Usability

Usability of Aotearoa prototype's user inteface (1: bad – 9: good): ___ [1-9]

Comments about the prototype's usability (improvements, missing support?):

____

____

Reuse

Did you have to modify the template you created with the Decision template form in the first place?: ___ [yes, no]

How would you rate the effort to modify the template to your actual needs (1: no effort – 9: immense effort)?: ___ [1-9]

Were you able to reuse a template for multiple decisions?: ___ [yes, no]

How would you rate the effort to modify the template to new needs in subsequent decisions (1: no effort – 9: immense effort)?: ___ [1-9]

Experiences and issues regarding the reusability of decision templates:

____

____

Any additional comments related to the whole decision-making support tool prototype Aotearoa:

____

____

List of Figures

1.1. A Consumer Measuring Compute Service Performance Meta-Data . . . 5
1.2. Overview of Research Process 12
1.3. Overview of Research Paths 13
1.4. Published Material Overview 18

2.1. Service Models and Layers in the Cloud Stack [8] 26
2.2. Example Structure of a Xen Hypervisor [50] 28
2.3. Software on VM Images and Appliances 30
2.4. Overview of VM Image Repository Contents 30
2.5. Exemplary Access Channels to a Compute Service 32
2.6. Structure of Compute Service Hardware Configuration Meta-Data [10] . 33
2.7. Classification of Performance Evaluations [77] 35
2.8. Typical OSCM System Architecture 41
2.9. Taxonomy of Multi-Attribute Decision-Making Methods [19] 45
2.10. Example of a AHP Hierarchy 46
2.11. Pairwise Comparison Matrix [112] 46

3.1. Screenshot of CloudHarmony Performance Meta-Data Service 55
3.2. Performance Variations on Google's AppEngine (Python) [13] 62
3.3. Architecture of the BEEN system [172] 63
3.4. Screenshot of the Phoromatic Online Service 64
3.5. Screenshot of Schedule Definition in Phoromatic 65

4.1. Overview of the Conceptual Framework 72
4.2. Basic Meta-Data Model 75
4.3. Overview of the Model for Automated Assessments at Run-Time 76
4.4. Levels of Automation [191] 77
4.5. The Process within the $(MC^2)^2$ 79
4.6. Typical Criteria Hierarchies 84
4.7. Schema of a Resulting Evaluation Function 85
4.8. Example of Normalized vs. Idealized Scale 87
4.9. Distribution of Expected Scores Among Ordered Compute Services . . 91
4.10. Random Distribution of Expected Scores Among Unordered Compute Services 92
4.11. The AotearoaLib Data Model 99

4.12. The Hoopla Architecture 100
4.13. Screenshot of the Hoopla Web Application 101
4.14. Computational Complexity with 100 Criteria and Growing Number of Services (Row/Sum Method) 107
4.15. Computational Complexity with 100 Criteria and Growing Number of Services (Saaty Method) 108
4.16. Computational Complexity with 100 Compute Services and Growing Number of Criteria (Row/Sum Method) 108
4.17. Computational Complexity with 100 Compute Services and Growing Number of Criteria (Saaty Method) 109
4.18. Computational Complexity with 100 Compute Services, 1000 Criteria and Growing Group Size 110

5.1. The Framework's Models & Methods used for Performance Assessment 115
5.2. Overview of the Automated Performance Benchmarking Method 117
5.3. Overview of the Automated Performance Measurement Procedure . . . 119
5.4. Available Levels to Induce Repetitions 121
5.5. Data Model of the Automated Performance Benchmarking Method . . . 125
5.6. User Involvement in an Automated Benchmarking Process 126
5.8. Architecture of LibIntelliCloudBench 139
5.9. Data Model LibIntelliCloudBench 139
5.10. Architecture of the Ballon Race Web Application 140
5.11. Screenshot of the Ballon Race Web Application 141
5.12. Measured Durations in Experiments EXP-A to EXP-C 147
5.13. Assessed Performance Meta-Data for EC2 Compute Services (LiB) . . 148
5.14. Assessed Performance Meta-Data for EC2 Compute Services (HiB) . . 149
5.15. Observed Cost Budget Consumption (in Percent) 152
5.16. Perfect Order 154
5.17. Perfect Order with One Switched Pair 155
5.18. Perfect Order with Two Switched Pairs 156
5.19. Reversed Perfect Order 157

6.1. The Framework's Models & Methods used for VM Image Assessment . 160
6.2. The VM Image Meta-Data Model 160
6.3. Overview of the Automated VM Image Introspection Procedure 163
6.4. Process of the Discoverer 164
6.5. Process of the Crawler 165
6.6. Architecture of the Crawler Script 168
6.7. Screenshot of the Crawler Web Application 169
6.10. Number of AMIs per Amazon EC2 Region 174
6.11. Meta-Data Attributes of AMIs in All Regions 174

A.1. AWS EC2 Performance Meta-Data 187

A.2. GoGrid Performance Meta-Data 187
A.3. Rackspace Performance Meta-Data 188
A.4. Joyent Cloud Performance Meta-Data 188

List of Tables

2.1. Meta-data Attributes available on Provider Websites 33
2.2. Sample of Common Performance Benchmarks 38
2.3. Meta-data Attributes accessible via Compute Service APIs 40

3.1. Literature Survey of Cloud-Related Multi-Criteria Selection Methods [40] 58

4.1. Example Criteria for Performance Characteristics 81
4.2. Common Requirement Types 82
4.3. List of Symbols in the Evaluation Function $f(\cdot)$ 86
4.4. Decision Models used for Experiments 106
4.5. Computation Times in 4 Realistic Scenarios (Row/Sum Method) 111
4.6. Computation Times in 4 Realistic Scenarios (Saaty Method) 111

5.1. Decision Models used for Experiments 145
5.2. Frequency of Processor Types per EC2 Compute Service 149
5.3. Results for Scenario 1 155
5.4. Results for Scenario 2 156
5.5. Results for Scenario 3 157
5.6. Results for Scenario 3 158

6.1. Non-Numerical Requirement Types 167
6.2. Software Package Count (Canonical and Alestic.com) 173

Acronyms

AHP	Analytic Hierarchy Process
ANP	Analytic Network Process
API	Application Programming Interface
AWS	Amazon Web Services
CCS	CloudHarmony Compute Scoring
CV	Coefficient of Variation
IaaS	Infrastructure-as-a-Service
IT	Information Technology
MADM	Multi-Attribute Decision-Making
$(MC^2)^2$	Multi-Criteria Comparison Method for Cloud Computing
MCDM	Multi-Criteria Decision-Making
OSCM	Operating System Configuration Management
PaaS	Platform-as-a-Service
QoS	Quality of Service
SaaS	Software-as-a-Service
SSH	Secure Shell
UML	Unified Modeling Language
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPN	Virtual Private Network

Literature

[1] J. Y. Thong. “An integrated model of information systems adoption in small businesses”. In: *Journal of management information systems* 15.4 (1999), pp. 187–214.

[2] D. Amor. *The e-business (R) evolution: Living and Working in an Interconnected World*. Prentice Hall PTR Upper Saddle River, NJ., 2000.

[3] D. Menasce. “Scaling for e-business”. In: *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on*. IEEE, 2000, pp. 511–513.

[4] R. Kalakota and M. Robinson. *E-business 2.0: Roadmap for Success*. Addison-Wesley Professional, 2001.

[5] C. D. Patel and A. J. Shah. *Cost model for planning, development and operation of a data center*. Tech. rep. Internet Systems and Storage Laboratory, HP Laboratories Palo Alto, 2005.

[6] C. Baun, J. Nimis, M. Kunze, and S. Tai. *Cloud computing: Web-based dynamic IT services*. Springer, 2011.

[7] P. Mell and T. Grance. *The NIST Definition of Cloud Computing, Recommendations of the National Institute of Standards and Technology*. Tech. rep. National Institute of Standards and Technology, 2011.

[8] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. “What’s inside the Cloud? An architectural map of the Cloud landscape”. In: *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Computer Society. IEEE, 2009, pp. 23–31.

[9] The Apache Software Foundation. *jClouds Java Library*. 2013. URL: `http://www.jclouds.org` (visited on Sept. 8, 2013).

[10] R. Prodan and S. Ostermann. “A survey and taxonomy of infrastructure as a service and web hosting cloud providers”. In: *Grid Computing, 2009 10th IEEE/ACM International Conference on*. IEEE, 2009, pp. 17–25.

[11] M. A. Vouk. “Cloud computing–issues, research and implementations”. In: *Journal of Computing and Information Technology* 16.4 (2004), pp. 235–246.

[12] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero. “Service specification in cloud environments based on extensions to open standards”. In: *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*. COMSWARE ’09. Dublin, Ireland: ACM, 2009, 19:1–19:12.

[13] A. Iosup, N. Yigitbasi, and D. Epema. “On the Performance Variability of Production Cloud Services”. In: *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE, 2011, pp. 104–113.

[14] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens. “Quantifying the performance isolation properties of virtualization systems”. In: *Proceedings of the 2007 Workshop on Experimental Computer Science*. ACM. New York, NY, USA: ACM, 2007, p. 6.

[15] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell. “VM^3: Measuring, modeling and managing VM shared resources”. In: *Computer Networks* 53.17 (2009), pp. 2873–2887.

[16] H. Jin, W. Cao, P. Yuan, and X. Xie. “VSCBenchmark: Benchmark for Dynamic Server Performance of Virtualization Technology”. In: *Proceedings of the 1st International Forum on Next-generation Multicore/Manycore Technologies*. IFMT ’08. Cairo, Egypt: ACM, 2008, 5:1–5:8.

[17] G. W. Torrance, W. Furlong, D. Feeny, and M. Boyle. “Multi-attribute preference functions”. In: *Pharmacoeconomics* 7.6 (1995), pp. 503–520.

[18] T. L. Saaty and L. G. Vargas. “How to Make a Decision”. In: *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Springer, 2012, pp. 1–21.

[19] K Yoon and C. L. Hwang. *Multiple Attribute Decision Making: An Introduction*. Sage Publications, 1995.

[20] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning. “Managing Security of Virtual Machine Images in a Cloud Environment”. In: *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. CCSW ’09. Chicago, Illinois, USA: ACM, 2009, pp. 91–96.

[21] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. “The Eucalyptus Open-Source Cloud-Computing System”. In: *Cluster Computing and the Grid, 2009. CCGRID ’09. 9th IEEE/ACM International Symposium on*. IEEE, 2009, pp. 124–131.

[22] B. Sotomayor, R. S. Montero, I. Llorente, and I. Foster. “Virtual Infrastructure Management in Private and Hybrid Clouds”. In: *Internet Computing, IEEE* 13.5 (2009), pp. 14–22.

[23] T. Garfinkel, M. Rosenblum, et al. “A Virtual Machine Introspection Based Architecture for Intrusion Detection.” In: *Proceedings of the Network and Distributed System Security Symposium*. Vol. 3. The Internet Society, 2003, pp. 191–206.

[24] J. Pfoh, C. Schneider, and C. Eckert. “A formal model for virtual machine introspection”. In: *Proceedings of the 1st ACM workshop on Virtual machine security*. ACM. New York, NY, USA: ACM, 2009, pp. 1–10.

[25] H. Miller and J. Veiga. “Cloud Computing: Will Commodity Services Benefit Users Long Term?” In: *IT Professional* 11.6 (2009), pp. 57–59.

[26] D. A. Menasce and V. A. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall Upper Saddle River, 2002.

[27] D. A. Menasce, V. A. Almeida, L. W. Dowdy, and L. Dowdy. *Performance by design: computer capacity planning by example*. Prentice Hall Professional, 2004.

[28] T. Kalibera, L. Bulej, and P. Tuma. “Benchmark precision and random initial state”. In: *in Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunications Systems (SPECTS 2005*. SCS, 2005, pp. 853–862.

[29] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. “Runtime measurements in the cloud: observing, analyzing, and reducing variance”. In: *Proceedings of the VLDB Endowment* 3 (1-2 2010), pp. 460–471.

[30] N. Huber, M. von Quast, M. Hauck, and S. Kounev. “Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments”. In: *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011)*. Vol. 1. SciTePress, 2011, pp. 7–9.

[31] R. Jain. *The art of computer systems performance analysis*. Vol. 182. John Wiley & Sons Chichester, 1991.

[32] A. Li, X. Yang, S. Kandula, and M. Zhang. “CloudCmp: comparing public cloud providers”. In: *Proceedings of the 10th annual conference on Internet measurement*. ACM. ACM, 2010, pp. 1–14.

[33] L. Gillam, B. Li, J. OLoughlin, and A. Tomar. “Fair Benchmarking for Cloud Computing systems”. In: *Journal of Cloud Computing: Advances, Systems and Applications* 2.1 (2013), p. 6.

[34] E. Walker. “Benchmarking amazon EC2 for high-performance scientific computing”. In: *USENIX Login* 33.5 (2008), pp. 18–23.

[35] T. Kalibera, L. Bulej, and P. Tuma. “Generic Environment for Full Automation of Benchmarking.” In: *SOQUA/TECOS* 58 (2004), pp. 125–132.

[36] R. P. Weicker. “An overview of common benchmarks”. In: *Computer* 23.12 (1990), pp. 65–75.

[37] R. P. Weicker. “A detailed look at some popular benchmarks”. In: *Parallel Computing* 17.10 (1991), pp. 1153–1172.

[38] W. Zeng, Y. Zhao, and J. Zeng. “Cloud service and service selection algorithm research”. In: *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation - GEC ’09* (2009), p. 1045.

[39] P. Saripalli and G. Pingali. “MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds”. In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 316–323.

[40] Z. U. Rehman, F. K. Hussain, and O. K. Hussain. “Towards Multi-criteria Cloud Service Selection”. In: *Proceedings of the 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, June 2011, pp. 44–48.

[41] T. Saaty. “Making and validating complex decisions with the AHP/ANP”. In: *Journal of Systems Science and Systems Engineering* 14.1 (2005), pp. 1–36.

[42] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. “The Cost of Doing Science on the Cloud: The Montage Example”. In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. SC ’08. Austin, Texas: IEEE Press, 2008, 50:1–50:12.

[43] C. Binnig, D. Kossmann, T. Kraska, and S. 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. Providence, Rhode Island: ACM, 2009, 9:1–9:6.

[44] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema. “Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing”. In: *Parallel and Distributed Systems, IEEE Transactions on* 22.6 (2011), pp. 931–945.

[45] Y. Bakos. “The Emerging Role of Electronic Marketplaces on the Internet”. In: *Commun. ACM* 41.8 (Aug. 1998), pp. 35–42.

[46] J. Y. Bakos. “Reducing buyer search costs: implications for electronic marketplaces”. In: *Management science* 43.12 (1997), pp. 1676–1692.

[47] S. Moorthy, B. T. Ratchford, and D. Talukdar. “Consumer Information Search Revisited: Theory and Empirical Analysis”. English. In: *Journal of Consumer Research* 23.4 (1997), pp. 263–277.

[48] J. J. McCall. “The economics of information and optimal stopping rules”. In: *The Journal of Business* 38.3 (1965), pp. 300–317.

[49] G. J. Stigler. “The economics of information”. In: *The journal of political economy* 69.3 (1961), pp. 213–225.

[50] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. "Xen and the art of virtualization". In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 164–177.

[51] J. Estublier. "Software Configuration Management: A Roadmap". In: *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. Limerick, Ireland: ACM, 2000, pp. 279–289.

[52] W. F. Tichy. "Tools for software configuration management". In: *Proceedings of the International Workshop on Software Version and Configuration Control (SCM)*. Vol. 30. Berichte des German Chapter of the ACM. Teubner, 1988, pp. 1–20.

[53] M. Shaw. "What makes good research in software engineering?" In: *International Journal on Software Tools for Technology Transfer* 4.1 (2002), pp. 1–7.

[54] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. "A break in the clouds: towards a cloud definition". In: *ACM SIGCOMM Computer Communication Review* 39.1 (2008), pp. 50–55.

[55] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. "A View of Cloud Computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[56] L. Barroso, J. Dean, and U. Holzle. "Web search for a planet: The Google cluster architecture". In: *Micro, IEEE* 23.2 (2003), pp. 22–28.

[57] Amazon Web Services, Inc. *Elastic Compute Cloud (EC2)*. 2014. URL: `http://aws.amazon.com/ec2/` (visited on Jan. 17, 2014).

[58] Rackspace, Inc. *The Rackspace Public Cloud*. 2013-07-21. 2013. URL: `http://www.rackspace.com/cloud/` (visited on July 21, 2013).

[59] R. Woollen. "The Internal Design of Salesforce.Com's Multi-tenant Architecture". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA: ACM, 2010, pp. 161–161.

[60] Highscalability.com. *YouTube Architecture*. 2014. URL: `http://highscalability.com/youtube-architecture` (visited on Jan. 17, 2014).

[61] VMware, Inc. *VMware vSphere Hypervisor*. 2013. URL: `http://www.vmware.com/products/vsphere-hypervisor` (visited on Sept. 6, 2013).

[62] Red Hat, Inc. *Red Hat Enterprise Virtualization*. Accessed September-2013. 2013. URL: `http://www.redhat.com/products/cloud-computing/virtualization/` (visited on Sept. 8, 2013).

[63] M. Rosenblum and T. Garfinkel. "Virtual machine monitors: Current technology and future trends". In: *Computer* 38.5 (2005), pp. 39–47.

[64] GoGrid, LLC. *Cloud Servers*. 2014. URL: http://www.gogrid.com/products/cloud-servers (visited on Jan. 15, 2014).

[65] Amazon Web Services, Inc. *AWS Marketplace*. Sept. 2013. URL: https://aws.amazon.com/marketplace (visited on Sept. 8, 2013).

[66] The Cloud Market. *The Cloud Market*. 2013. URL: http://thecloudmarket.com/ (visited on Sept. 8, 2013).

[67] BitNami. *BitNami.com*. 2013. URL: http://bitnami.com/ (visited on Sept. 8, 2013).

[68] R. L. Grossman. "The case for cloud computing". In: *IT professional* 11.2 (2009), pp. 23–27.

[69] Amazon Web Services, Inc. *How AWS pricing works*. 2012. URL: http://media.amazonwebservices.com/AWS_Pricing_Overview.pdf (visited on Apr. 21, 2012).

[70] Rackspace, Inc. *Rackspace Cloud Server Pricing*. 2013. URL: http://www.rackspace.com/cloud/servers/pricing/ (visited on July 21, 2013).

[71] Microsoft. *Cloud Services Pricing Details*. 2013. URL: http://www.windowsazure.com/en-us/pricing/details/cloud-services/ (visited on Nov. 25, 2013).

[72] Amazon Web Services, Inc. *Amazon Web Services*. Sept. 2013. URL: http://aws.amazon.com (visited on Sept. 8, 2013).

[73] 1&1. *Dynamic Cloud Servers*. 2014. URL: http://hosting.1und1.com/dynamic-cloud-server (visited on Jan. 15, 2014).

[74] T. Metsch, A. Edmonds, R. Nyren, and A. Papaspyrou. *Open Cloud Computing Interface–Core*. Open Grid Forum, Open Cloud Computing Interface (OCCI) Working Group. 2011. URL: http://ogf.org/documents/GFD.183.pdf (visited on Nov. 21, 2013).

[75] T. Metsch and A. Edmonds. *Open Cloud Computing Interface–Infrastructure*. Open Grid Forum, Open Cloud Computing Interface (OCCI) Working Group. 2011. URL: http://ogf.org/documents/GFD.184.pdf (visited on Nov. 21, 2013).

[76] D. Davis and G. Pilz. *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol*. Tech. rep. Technical Report, Distributed Management Work Force (DMTF), 2012.

[77] F. Desprez, G. Fox, E. Jeannot, K. Keahey, et al. *Supporting Experimental Computer Science*. Anglais. Rapport de recherche RR-8035. INRIA, July 2012, p. 29.

[78] M. Hauck, J. Happe, and R. Reussner. “Towards Performance Prediction for Cloud Computing Environments based on Goal-oriented Measurements.” In: *CLOSER*. 2011, pp. 616–622.

[79] R. Krebs, C. Momm, and S. Kounev. “Metrics and techniques for quantifying performance isolation in cloud environments”. In: *Science of Computer Programming* 90, Part B.0 (2014). Special Issue on Component-Based Software Engineering and Software Architecture, pp. 116 –134.

[80] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann. “What Are You Paying for? Performance Benchmarking for Infrastructure-as-a-Service Offerings”. In: *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*. IEEE. Washington, D.C., USA: IEEE, 2011, pp. 484–491.

[81] B. Randall and L. J. Russell. *Algol-60 Implementation*. Orlando, FL, USA: Academic Press, Inc., 1964.

[82] H. J. Curnow and B. A. Wichmann. “A synthetic benchmark”. In: *The Computer Journal* 19.1 (1976), pp. 43–49.

[83] J. Dongarra. “The LINPACK Benchmark: An explanation”. In: *Supercomputing*. Ed. by E. Houstis, T. Papatheodorou, and C. Polychronopoulos. Vol. 297. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1988, pp. 456–474.

[84] R. P. Weicker. “Dhrystone: A Synthetic Systems Programming Benchmark”. In: *Commun. ACM* 27.10 (Oct. 1984), pp. 1013–1030.

[85] J. L. Henning. “SPEC CPU2006 Benchmark Descriptions”. In: *SIGARCH Comput. Archit. News* 34.4 (Sept. 2006), pp. 1–17.

[86] K. M. Dixit. “The SPEC benchmarks”. In: *Parallel Computing* 17.10–11 (1991). <ce:title>Benchmarking of high performance supercomputers</ce:title>, pp. 1195 –1209.

[87] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. “MiBench: A free, commercially representative embedded benchmark suite”. In: *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[88] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. Jones, and B. Parady. “SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance”. English. In: *OpenMP Shared Memory Parallel Programming*. Ed. by R. Eigenmann and M. Voss. Vol. 2104. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 1–10.

[89] C. Bienia, S. Kumar, J. P. Singh, and K. Li. “The PARSEC Benchmark Suite: Characterization and Architectural Implications”. In: *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. PACT ’08. Toronto, Ontario, Canada: ACM, 2008, pp. 72–81.

[90] W. Gentzsch, D. Girou, A. Kennedy, H. Lederer, J. Reetz, M. Riedel, A. Schott, A. Vanni, M. Vazquez, and J. Wolfrat. “DEISA—Distributed European Infrastructure for Supercomputing Applications”. English. In: *Journal of Grid Computing* 9.2 (2011), pp. 259–277.

[91] P. Media. *Phoronix Test Suite*. 2014. URL: `http://www.phoronix-test-suite.com/` (visited on Jan. 15, 2014).

[92] I. Team. *Inqusitor*. 2014. URL: `http://www.inquisitor.ru/about/` (visited on Jan. 15, 2014).

[93] The OpenStack Foundation. *OpenStack Cloud Software*. 2014. URL: `https://www.openstack.org/` (visited on Mar. 20, 2014).

[94] The OpenStack Project. *OpenStack Manuals - Image metadata*. 2014. URL: `http://docs.openstack.org/image-guide/content/image-metadata.html` (visited on Mar. 20, 2014).

[95] The OpenStack Project. *OpenStack Manuals - Requesting Detailed Metadata on Public VM Images*. 2014. URL: `http://docs.openstack.org/api/openstack-image-service/1.1/content/requesting-detailed-metadata-on-public-vm-images.html` (visited on Mar. 20, 2014).

[96] B. Baker. *The Official InstallShield for Windows Installer Developer’s Guide*. M & T Books, 2001.

[97] I. Murdock. “Overview of the Debian GNU/Linux System”. In: *Linux Journal* 1994.6es (Oct. 1994).

[98] A. van der Hoek and A. L. Wolf. “Software release management for component-based software”. In: *Software: Practice and Experience* 33.1 (2003), pp. 77–98.

[99] Software in the Public Interest, Inc. *Advanced Package Tool (APT)*. 2014. URL: `https://wiki.debian.org/Apt` (visited on Jan. 15, 2014).

[100] Novell. *YaST - Yet another Setup Tool*. 2014. URL: `http://en.opensuse.org/YaST` (visited on Jan. 15, 2014).

[101] M. Ewing and E. Troan. “The RPM packaging system”. In: *Proceedings of the First Conference on Freely Redistributable Software, Cambridge, MA, USA*. 1996.

[102] M. Burgess. “Cfengine: a site configuration engine”. In: *USENIX Computing systems*. Vol. 8. USENIX Association, 1995, pp. 309–337.

[103] M. Burgess and R. Ralston. “Distributed resource administration using cfengine”. In: *Software: Practice and Experience* 27.9 (1997), pp. 1083–1101.

[104] D. Ressman and J. Valdés. “Use of CFengine for Automated, Multi-Platform Software and Patch Distribution”. In: *LISA*. USENIX Association, 2000, pp. 207–218.

[105] I. Opscode. *Chef Software*. 2014. URL: `http://www.getchef.com/chef/` (visited on Jan. 15, 2014).

[106] Puppet Labs, Inc. *Puppet*. 2014. URL: `http://puppetlabs.com/` (visited on Jan. 15, 2014).

[107] Opscode, Inc. *Chef Ohai Software*. 2014. URL: `http://docs.opscode.com/ohai.html` (visited on Jan. 15, 2014).

[108] C. L. Hwang, A. S. M. Masud, et al. *Multiple objective decision making-methods and applications*. Vol. 164. Springer, 1979.

[109] T. L. Saaty. *What is the analytic hierarchy process?* Springer, 1988.

[110] T. L. Saaty. "How to make a decision: The analytic hierarchy process". In: *European Journal of Operational Research* 48.1 (Sept. 1990), pp. 9–26.

[111] T. L. Saaty. "Fundamentals of the analytic network process—multiple networks with benefits, costs, opportunities and risks". In: *journal of systems science and systems engineering* 13.3 (2004), pp. 348–379.

[112] T. L. Saaty. "Relative measurement and its generalization in decision making why pairwise comparisons are central in mathematics for the measurement of intangible factors the analytic hierarchy/network process". English. In: *RACSAM - Revista de la Real Academia de Ciencias Exactas, Fisicas y Naturales. Serie A. Matematicas* 102.2 (2008), pp. 251–318.

[113] Y. Chow and H. Robbins. "On optimal stopping rules". In: *Probability Theory and Related Fields* 2.1 (1963), pp. 33–49.

[114] T. S. Ferguson. *Optimal Stopping and Applications*. 2012. URL: `http://www.math.ucla.edu/~tom/Stopping/Contents.html` (visited on Jan. 14, 2014).

[115] J. P. Gilbert and F. Mosteller. "Recognizing the maximum of a sequence". In: *Selected Papers of Frederick Mosteller*. Springer, 2006, pp. 355–398.

[116] T. S. Ferguson. "Who solved the secretary problem?" In: *Statistical science* 4.3 (1989), pp. 282–289.

[117] S. A. Lippman and J. McCall. "The economics of job search: A survey". In: *Economic inquiry* 14.2 (1976), pp. 155–189.

[118] W. E. Stein, D. A. Seale, and A. Rapoport. "Analysis of heuristic solutions to the best choice problem". In: *European Journal of Operational Research* 151.1 (2003), pp. 140 –152.

[119] F. T. Bruss. "Sum the Odds to One and Stop". English. In: *The Annals of Probability* 28.3 (2000), pp. 1384–1391.

[120] D. A. Seale and A. Rapoport. "Sequential Decision Making with Relative Ranks: An Experimental Investigation of the "Secretary Problem"". In: *Organizational Behavior and Human Decision Processes* 69.3 (1997), pp. 221 –236.

[121] J. J. McCall. “Economics of information and job search”. In: *The Quarterly Journal of Economics* (1970), pp. 113–126.

[122] A. Sen. “Metadata Management: Past, Present and Future”. In: *Decis. Support Syst.* 37.1 (Apr. 2004), pp. 151–173.

[123] A. Tannenbaum. *Metadata Solutions: Using Metamodels, Repositories, Xml, and Enterprise Portals to Generate Information on Demand*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[124] E. Madja, A. Hafid, R. Dssouli, G. v.Bochmann, and J. Gecsei. “Meta-data modelling for quality of service (QoS) management in the World Wide Web (WWW)”. In: *Multimedia Modeling, 1998. MMM ’98. Proceedings. 1998.* 1998, pp. 223–230.

[125] A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker. “UDDIe: an extended registry for Web services”. In: *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*. 2003, pp. 85–89.

[126] M. Papazoglou. *Web services: principles and technology*. Pearson Education, 2008.

[127] K. Sivashanmugam, K. Verma, A. P. Sheth, and J. A. Miller. “Adding Semantics to Web Services Standards.” In: *International Conference on Web Services*. 2003, pp. 395–401.

[128] S. A. McIlraith, T. C. Son, and H. Zeng. “Semantic Web Services”. In: *IEEE Intelligent Systems* 16.2 (2001), pp. 46–53.

[129] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. “Web services agreement specification (WS-Agreement)”. In: *Open Grid Forum*. Vol. 128. 2007.

[130] A. Keller and H. Ludwig. “The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services”. English. In: *Journal of Network and Systems Management* 11.1 (2003), pp. 57–81.

[131] P. Patel, A. H. Ranabahu, and A. P. Sheth. “Service level agreement in cloud computing”. In: *Cloud Workshops at OOPSLA’09* (2009).

[132] R. Buyya, R. Ranjan, and R. Calheiros. “InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services”. English. In: *Algorithms and Architectures for Parallel Processing*. Ed. by C.-H. Hsu, L. Yang, J. Park, and S.-S. Yeo. Vol. 6081. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 13–31.

[133] M. Wang, X. Wu, W. Zhang, F. Ding, J. Zhou, and G. Pei. “A Conceptual Platform of SLA in Cloud Computing”. In: *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*. 2011, pp. 1131–1135.

[134] S. Venticinque, R. Aversa, B. Di Martino, M. Rak, and D. Petcu. "A Cloud Agency for SLA Negotiation and Management". English. In: *Euro-Par 2010 Parallel Processing Workshops*. Ed. by M. Guarracino, F. Vivien, J. Träff, M. Cannatoro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. Di Martino, and M. Alexander. Vol. 6586. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 587–594.

[135] M. Alhamad, T. Dillon, and E. Chang. "Conceptual SLA framework for cloud computing". In: *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*. 2010, pp. 606–610.

[136] S. Sundareswaran, A. Squicciarini, and D. Lin. "A Brokerage-Based Approach for Cloud Service Selection". In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. 2012, pp. 558–565.

[137] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente. "From infrastructure delivery to service management in clouds". In: *Future Generation Computer Systems* 26.8 (2010), pp. 1226 –1240.

[138] A. Goscinski and M. Brock. "Toward dynamic and attribute based publication, discovery and selection for cloud computing". In: *Future Generation Computer Systems* 26.7 (2010), pp. 947 –970.

[139] CloudHarmony.com. *CloudHarmony Web Application*. 2011. URL: `http://cloudharmony.com/` (visited on Feb. 4, 2011).

[140] A Parasuraman, V. A. Zeithaml, and L. L. Berry. "Servqual". In: *Journal of retailing* 64.1 (1988), pp. 12–40.

[141] J. Santos. "EâĂŘservice quality: a model of virtual service quality dimensions". In: *Managing Service Quality: An International Journal* 13.3 (2003), pp. 233–246.

[142] P. Saripalli and B. Walters. "Quirc: A quantitative impact and risk assessment framework for cloud security". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE. 2010, pp. 280–288.

[143] E. Cristobal, C. Flavián, and M. Guinalíu. "Perceived eâĂŘservice quality (PeSQ)". In: *Managing Service Quality: An International Journal* 17.3 (2007), pp. 317–340.

[144] S. Habib, S. Ries, and M. Muhlhauser. "Towards a Trust Management System for Cloud Computing". In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. 2011, pp. 933–939.

[145] J. Abawajy. "Determining Service Trustworthiness in Intercloud Computing Environments". In: *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*. 2009, pp. 784–788.

[146] K. Khan and Q. Malluhi. “Establishing Trust in Cloud Computing”. In: *IT Professional* 12.5 (2010), pp. 20–27.

[147] S. Pearson, Y. Shen, and M. Mowbray. “A Privacy Manager for Cloud Computing”. English. In: *Cloud Computing*. Ed. by M. Jaatun, G. Zhao, and C. Rong. Vol. 5931. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 90–106.

[148] M. Klems, J. Nimis, and S. Tai. “Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing”. English. In: *Designing E-Business Systems. Markets, Services, and Networks*. Ed. by C. Weinhardt, S. Luckner, and J. Stößer. Vol. 22. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2009, pp. 110–123.

[149] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang. “The Method and Tool of Cost Analysis for Cloud Computing”. In: *Cloud Computing, 2009. CLOUD ’09. IEEE International Conference on*. 2009, pp. 93–100.

[150] Y. Chen and R. Sion. “To Cloud or Not to Cloud?: Musings on Costs and Viability”. In: *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. SOCC ’11. Cascais, Portugal: ACM, 2011, 29:1–29:7.

[151] S. Frey, W. Hasselbring, and B. Schnoor. “Automatic conformance checking for migrating software systems to cloud infrastructures and platforms”. In: *Journal of Software: Evolution and Process* 25.10 (2013), pp. 1089–1115.

[152] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. Teregowda. “Decision Support Tools for Cloud Migration in the Enterprise”. In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 2011, pp. 541–548.

[153] B. Johnson and Y. Qu. “A Holistic Model for Making Cloud Migration Decision: A Consideration of Security, Architecture and Business Economics”. In: *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. 2012, pp. 435–441.

[154] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. “Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud”. In: *Proceedings of the ACM SIGCOMM 2010 Conference*. SIGCOMM ’10. New Delhi, India: ACM, 2010, pp. 243–254.

[155] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. “Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds”. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS ’09. Chicago, Illinois, USA: ACM, 2009, pp. 199–212.

[156] Z. ur Rehman, O. Hussain, and F. Hussain. “Iaas Cloud Selection using MCDM Methods”. In: *e-Business Engineering (ICEBE), 2012 IEEE Ninth International Conference on*. IEEE, 2012, pp. 246–251.

[157] Jason Read, CloudHarmony.com. *What is an ECU? CPU Benchmarking in the Cloud*. 2014. URL: `http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html` (visited on Mar. 17, 2014).

[158] Jason Read, CloudHarmony.com. *Cloud Server Performance Benchmarking*. 2014. URL: `http://blog.cloudharmony.com/2010/03/cloud-server-performance-benchmarking.html` (visited on Mar. 17, 2014).

[159] A. Li, X. Yang, S. Kandula, and M. Zhang. "CloudCmp: shopping for a cloud made easy". In: *USENIX HotCloud* (2010).

[160] S.-M. Han, M. M. Hassan, C.-W. Yoon, and E.-N. Huh. "Efficient Service Recommendation System for Cloud Computing Market". In: *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ICIS '09. Seoul, Korea: ACM, 2009, pp. 839–845.

[161] M. Godse and S. Mulik. "An Approach for Selecting Software-as-a-Service (SaaS) Product". In: *Cloud Computing (CLOUD), 2009 IEEE International Conference on*. IEEE, 2009, pp. 155–158.

[162] S. Garg, S. Versteeg, and R. Buyya. "SMICloud: A Framework for Comparing and Ranking Cloud Services". In: *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. 2011, pp. 210–218.

[163] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright. "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud". In: *2nd IEEE International Conference on Cloud Computing Technology and Science*. IEEE. 2010, pp. 159–168.

[164] D. Menascé. "TPC-W: A benchmark for e-commerce". In: *IEEE Internet Computing* 6.3 (2002), pp. 83–87.

[165] Transaction Processing Performance Council (TPC). *TPC-W Benchmark*. 2011. URL: `http://www.tpc.org/tpcw/` (visited on Jan. 20, 2011).

[166] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, S. Patil, A. Fox, and D. Patterson. "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0". In: *Proc. of CCA*. Citeseer. 2008.

[167] V. Makhija, B. Herndon, P. Smith, L. Roderick, E. Zamost, and J. Anderson. "VMmark: A scalable benchmark for virtualized systems". In: *VMware Inc, CA, Tech. Rep. VMware-TR-2006-002, September* (2006).

[168] J. Casazza, M. Greenfield, and K. Shi. "Redefining server performance characterization for virtualization benchmarking". In: *Intel Technology Journal* 10.3 (2006), pp. 243–251.

[169] Y. Mei, L. Liu, X. Pu, and S. Sivathanu. “Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud”. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. i/o trhoughput for VMs on same physical host. IEEE, 2010, pp. 59 –66.

[170] T. Kalibera and R. Jones. “Rigorous Benchmarking in Reasonable Time”. In: *Proceedings of the 2013 International Symposium on Memory Management*. ISMM ’13. Seattle, Washington, USA: ACM, 2013, pp. 63–74.

[171] T. Kalibera and R. Jones. “Rigorous Benchmarking in Reasonable Time”. In: *SIGPLAN Not.* 48.11 (June 2013), pp. 63–74.

[172] T. Kalibera, J. Lehotsky, D. Majda, B. Repcek, M. Tomcanyi, A. Tomecek, P. Tuma, and J. Urban. “Automated Benchmarking and Analysis Tool”. In: *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*. valuetools ’06. Pisa, Italy: ACM, 2006, pp. 1–10.

[173] The Mono Project. *Mono Cross Platform Open-Source .NET Development Framework*. 2014. URL: `http://www.mono-project.com/` (visited on Mar. 20, 2014).

[174] T. Kalibera and P. Tuma. “Precise Regression Benchmarking with Random Effects: Improving Mono Benchmark Results”. In: *Formal Methods and Stochastic Models for Performance Evaluation*. Ed. by Horvóth, András and Telek, Miklós. Vol. 4054. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 63–77.

[175] T. Kalibera, L. Bulej, and P. Tuma. “Automated detection of performance regressions: the mono experience”. In: *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 13th IEEE International Symposium on*. IEEE, 2005, pp. 183–190.

[176] T. Kalibera, L. Bulej, and P. Tuma. “Quality Assurance in Performance: Evaluating Mono Benchmark Results”. In: *Quality of Software Architectures and Software Quality*. Ed. by R. Reussner, J. Mayer, J. Stafford, S. Overhage, S. Becker, and P. Schroeder. Vol. 3712. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 271–288.

[177] L. Bulej, T. Kalibera, and P. Tuma. “Repeated results analysis for middleware regression benchmarking”. In: *Performance Evaluation* 60.1âĂŽÄì4 (2005). Performance Modeling and Evaluation of High-Performance Parallel and Distributed Systems, pp. 345 –358.

[178] P. Media. *Phoromatic Web Application (Beta)*. 2014. URL: `http://www.phoromatic.com/` (visited on Jan. 15, 2014).

[179] B. Rochwerger, D. Breitgand, E. Levy, a. Galis, et al. “The Reservoir model and architecture for open federated cloud computing”. In: *IBM Journal of Research and Development* 53.4 (July 2009), 4:1–4:11.

[180] J. Matthews, T. Garfinkel, C. Hoff, and J. Wheeler. “Virtual Machine Contracts for Datacenter and Cloud Computing Environments”. In: *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*. ACDC ’09. Barcelona, Spain: ACM, 2009, pp. 25–30.

[181] A. Dastjerdi, S. Tabatabaei, and R. Buyya. “An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery”. In: *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 104–112.

[182] D. Reimer, A. Thomas, G. Ammons, T. Mummert, B. Alpern, and V. Bala. “Opening black boxes: using semantic information to combat virtual machine image sprawl”. In: *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. VEE ’08. Seattle, WA, USA: ACM, 2008, pp. 111–120.

[183] G. Ammons, V. Bala, T. Mummert, D. Reimer, and X. Zhang. “Virtual Machine Images As Structured Data: The Mirage Image Library”. In: *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud’11. Portland, OR: USENIX Association, 2011, pp. 22–22.

[184] R. Filepp, L. Shwartz, C. Ward, R. Kearney, K. Cheng, C. Young, and Y. Ghosheh. “Image selection as a service for cloud computing environments”. In: *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1 –8.

[185] K. Magoutis, M. Devarakonda, N. Joukov, and N. G. Vogl. “Galapagos: Model-driven discovery of end-to-end application-storage relationships in distributed systems”. In: *IBM Journal of Research and Development* 52.4.5 (2008), pp. 367 –377.

[186] IBM. *Tivoli Application Dependency Discovery Manager*. 2012. URL: `http://www-01.ibm.com/software/tivoli/products/taddm/` (visited on May 26, 2012).

[187] M. S. Wilson. “Constructing and managing appliances for cloud deployments from repositories of reusable components”. In: *Proceedings of the 2009 conference on Hot topics in cloud computing*. HotCloud’09. San Diego, California: USENIX Association, 2009, pp. 1–5.

[188] H. Liu. “Rapid application configuration in Amazon cloud using configurable virtual appliances”. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. SAC ’11. TaiChung, Taiwan: ACM, 2011, pp. 147–154.

[189] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. “Information and control in gray-box systems”. In: *ACM SIGOPS Operating Systems Review*. Vol. 35. 5. New York, NY, USA: ACM, 2001, pp. 43–56.

[190] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. “Sandpiper: Black-box and gray-box resource management for virtual machines”. In: *Computer Networks* 53.17 (2009), pp. 2923–2938.

[191] M. R. Endsley. “Level of automation effects on performance, situation awareness and workload in a dynamic control task”. In: *Ergonomics* 42.3 (1999). PMID: 10048306, pp. 462–492.

[192] R Parasuraman, T. B. Sheridan, and C. D. Wickens. “A model for types and levels of human interaction with automation.” In: *IEEE transactions on systems, man, and cybernetics. Part A, Systems and humans : a publication of the IEEE Systems, Man, and Cybernetics Society* 30.3 (May 2000), pp. 286–97.

[193] M. R. Endsley. “Level of Automation: Integrating Humans and Automated Systems”. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 41.1 (1997), pp. 200–204.

[194] R. L. Keeney and H Raiffa. *Decisions with multiple objectives*. Cambridge Books. Cambridge University Press, 1993.

[195] T. Saaty. *Theory and Applications of the Analytic Network Process - Decision Making with Benefits, Opportunities, Costs, and Risks*. Ed. by T. Saaty. RWS Publications, USA, 2005, p. 352.

[196] F. Pardee, T. Kirkwood, K. MacCrimmon, J. Miller, C. Phillips, J. Ranftl, K. Smith, and D. Whitcomb. “Measurement and evaluation of transportation system effectiveness”. In: *RAND Memorandum*. RAND Corporation, USA, 1969.

[197] D. Baker, D. Bridges, R. Hunter, G. Johnson, J. Krupa, J. Murphy, and K. Sorenson. “Guidebook to decision-making methods”. In: *Washington DC: US Department of Energy. WSRC-IM-2002-00002* (2001).

[198] T. Saaty. “Decision making—the analytic hierarchy and network processes (AHP/ANP)”. In: *Journal of systems science and systems engineering* 13.1 (2004), pp. 1–35.

[199] J. Karlsson, S. Olsson, and K. Ryan. “Improved practical support for large-scale requirements prioritising”. English. In: *Requirements Engineering* 2.1 (1997), pp. 51–60.

[200] J. Hicklin, C. Moler, P. Webb, R. F. Boisvert, B. Miller, R. Pozo, and K. Remington. *JAMA : A Java Matrix Package*. 2014. URL: `http://math.nist.gov/javanumerics/jama/` (visited on Mar. 27, 2014).

[201] Google, Inc. *Google AppEngine*. 2014. URL: `https://appengine.google.com/` (visited on Mar. 27, 2014).

[202] Google, Inc. *Google Web Toolkit*. 2014. URL: `http://www.gwtproject.org` (visited on Mar. 27, 2014).

[203] Isomorphic Software. *SmartGWT*. 2014. URL: `https://code.google.com/p/smartgwt/` (visited on Mar. 27, 2014).

[204] M. Menzel, M. Schönherr, and S. Tai. “$(MC^2)^2$: Criteria, Requirements and a Software Prototype for Cloud Infrastructure Decisions”. In: *Software: Practice and Experience* (2011).

[205] M. Menzel and R. Ranjan. “CloudGenius: Decision Support for Web Server Cloud Migration”. In: *Proceedings of the 21st International Conference on World Wide Web*. ACM. Lyon, France: ACM, 2012, pp. 979–988.

[206] S. Hove and B. Anda. “Experiences from conducting semi-structured interviews in empirical software engineering research”. In: *Software Metrics, 2005. 11th IEEE International Symposium*. IEEE, 2005, 10 pp.–23.

[207] P. Media. *OpenBenchmarking.org*. 2014. URL: `http://www.openbenchmarking.org` (visited on Mar. 27, 2014).

[208] Object Management Group. *Business Process Model and Notation (BPMN) Specification, Version 2.0*. 2011. URL: `http://www.omg.org/spec/BPMN/2.0/` (visited on Sept. 12, 2013).

[209] Vaadin Ltd. *Vaadin Java Framework*. 2014. URL: `https://vaadin.com` (visited on Mar. 27, 2014).

[210] The Apache Software Foundation. *Apache Tomcat*. 2014. URL: `http://tomcat.apache.org/` (visited on Mar. 27, 2014).

[211] Oracle Corporation. *MySQL Database System*. 2014. URL: `http://www.mysql.com/` (visited on Mar. 27, 2014).

[212] Sauron Software. *cron4j*. 2014. URL: `http://www.sauronsoftware.it/projects/cron4j/` (visited on Mar. 27, 2014).

[213] PassMark Software. *CPU List*. 2014. URL: `http://www.cpubenchmark.net/` (visited on Mar. 27, 2014).

[214] PassMark Software. *PassMark Software - PC Benchmark and Test Software*. 2011. URL: `http://www.passmark.com/` (visited on Feb. 4, 2011).

[215] S. Haak and M. Menzel. “Autonomic Benchmarking for Cloud Infrastructures: An Economic Optimization Model”. In: *Proceedings of the 1st ACM/IEEE workshop on Autonomic computing in economics*. ACM. Karlsruhe, Germany: ACM, 2011, pp. 27–32.

[216] Puppet Labs, Inc. *Puppet Facter*. 2014. URL: `http://projects.puppetlabs.com/projects/facter` (visited on Jan. 15, 2014).

[217] M. Menzel, M. Klems, H. A. Le, and S. Tai. “A Configuration Crawler for Virtual Appliances in Compute Clouds”. In: *Proceedings of the 1st International Conference on Cloud Engineering (IC2E)*. IEEE. San Francisco, USA: IEEE, 2013, pp. 201–209.

[218] A. Beitch, B. Liu, T. Yung, R. Griffith, A. Fox, and D. A. Patterson. “Rain: A workload generation toolkit for cloud computing applications”. In: *Electrical Engineering and Computer Sciences University of California at Berkeley, White paper UCB/EECS-2010-14* (2010).

ISBN 978-3-7375-5175-5