Design and Implementation of a System for Evaluating Distributed Application Deployments in Multi-Clouds

Antonis Papaioannou

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, Heraklion, GR-70013, Greece

Thesis Advisors: Prof. *Evangelos Markatos*

This work has been performed at the **University of Crete, School of Science and Engineering, Computer Science Department** and at the **Institute of Computer Science (ICS) - Foundation for Research and Technology - Hellas (FORTH), Heraklion, Crete, GREECE.**

The work is partially supported by the **PaaSage (FP7-317715) EU project**.

University of Crete
Computer Science Department

Design and Implementation of a System for Evaluating Distributed Application Deployments in Multi-Clouds

Thesis submitted by
Antonis Papaioannou
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Antonis Papaioannou

Committee approvals: _____
Evangelos Markatos
Professor, Thesis Supervisor

_____
Kostas Magoutis
Researcher, ICS-FORTH, Thesis Co-Supervisor

_____
Dimitris Plexousakis
Professor, Committee Member

Departmental approval: _____
Angelos Bilas
Professor, Director of Graduate Studies

Heraklion, October 2013

Abstract

Cloud computing has recently become an important model of infrastructural services that complements, and in some cases replaces, traditional data centers in providing distributed applications with the resources they need to serve Internet-scale workloads. The multitude of Cloud providers offering infrastructural services today raise the challenge of heterogeneity in the resources available to applications. An application developer is thus faced with the problem of choosing an appropriate set of Cloud providers and resources to support their application. One way to address this problem is to establish a Cloud-independent classification of Cloud resources and to systematically explore a large collection of past execution histories of real deployments of applications.

In this thesis we present an architecture for the modeling, collection, and evaluation of long-term histories of deployments of distributed multi-tier applications on federations of Clouds (Multi-Clouds). Our goal is to capture several aspects of application development and deployment lifecycle, including the evolving application structure, requirements, goals, and service-level objectives; application deployment descriptions; runtime monitoring, and quality control; Cloud provider characteristics; and to provide a Cloud-independent resource classification scheme that is a key to reasoning about Multi-Cloud deployments of complex large-scale applications. Since our target is to capture the continuous evolution of applications and their deployments over time, we ensure that our metadata model is designed to optimize space usage. Additionally, we demonstrate that using the model and data collections over varying deployments of an application (using the SPEC jEnterprise2010 distributed benchmark as a case study) one can answer important questions about which deployment options work best in terms of performance, reliability, cost, and combinations thereof. As an example, our analysis can pinpoint the most cost-effective among a dozen deployments of an application leading to savings of $2,100 per year without impacting its service-level objectives.

Περίληψη

Τα Υπολογιστικά Νέφη αποτελούν σήμερα ένα σημαντικό μοντέλο υπηρεσιών υποδομής που συμπληρώνουν, και σε κάποιες περιπτώσεις αντικαθιστούν, τα παραδοσιακά Κέντρα Δεδομένων στην παροχή πόρων σε κατανεμημένες εφαρμογές μεγάλης κλίμακας. Ωστόσο, ο μεγάλος αριθμός παρόχων υπηρεσιών Υπολογιστικού Νέφους και η ετερογένεια στους πόρους που παρέχουν συνιστούν μια πρόκληση για τις εφαρμογές. Ένας τρόπος να αντιμετωπιστεί αυτή η πρόκληση είναι η συστηματική μελέτη και κατηγοριοποίηση των πόρων Υπολογιστικού Νέφους σε κλάσεις απόδοσης ανεξάρτητες του παρόχου. Ταυτόχρονα, η ανάλυση του ιστορικού εγκατάστασης και εκτέλεσης κατανεμημένων εφαρμογών σε ετερογενή περιβάλλοντα Νέφους μπορεί να βοηθήσει σημαντικά τους μηχανικούς λογισμικού στο να κατανοήσουν τις σχέσεις κόστους - ωφέλειας σε αυτά τα περιβάλλοντα.

Στην εργασία αυτή παρουσιάζουμε μια αρχιτεκτονική για τη μοντελοποίηση, συλλογή και αξιολόγηση μακρόχρονων ιστορικών εκτέλεσης κατανεμημένων και πολυεπίπεδων (multi-tier) εφαρμογών σε πολλαπλά Υπολογιστικά Νέφη. Ο σκοπός μας είναι η καταγραφή της εγκατάστασης και εξέλιξης της εφαρμογής, συμπεριλαμβανομένων των απαιτήσεων και στόχων, της παρακολούθησης (monitoring) και του ελέγχου κατά την εκτέλεση, και τέλος των χαρακτηριστικών των παρόχων. Ταυτόχρονα, η κατηγοριοποίηση πόρων ανεξαρτήτως παρόχου βάσει συστηματικών μετρήσεων αποτελεί σημαντικό παράγοντα για την επιλογή της εγκατάστασης σύνθετων εφαρμογών μεγάλης κλίμακας σε πολλαπλά Υπολογιστικά Νέφη. Παρουσιάζουμε ένα σχεσιακό μοντέλο μεταδεδομένων που ενοποιεί την παραπάνω πληροφορία. Χρησιμοποιώντας αυτό το μοντέλο για τη συλλογή δεδομένων από διαφορετικές εγκαταστάσεις και εκτελέσεις της εφαρμογής (μελετώντας ως παράδειγμα την εφαρμογή-benchmark SPEC jEnterprise2010) υποδεικνύουμε πως μπορούν να απαντηθούν σημαντικά ερωτήματα που αφορούν την εύρεση του καλύτερου τρόπου εγκατάστασης της εφαρμογής με κριτήριο την απόδοση, το κόστος, και την επίτευξη των στόχων ή συνδυασμούς των προηγουμένων. Ως παράδειγμα της ανάλυσής μας, η συστηματική μελέτη διαφορετικών συνδιασμών εγκατάστασης του SPEC jEnterprise2010 μπορεί να υποδείξει τον πιο οικονομικό τρόπο εγκατάστασης της εφαρμογής, επιφέρωντας εξοικονόμηση $2.100 ετησίως χωρίς να επηρεάζονται οι στόχοι απόδοσης.

Acknowledgements

There are so many people that I would like to thank, each one helped me with their own special way. First of all, I would like to thank my advisor Professor Evangelos Markatos for supervising, guiding and having trust in this work. I would also like to thank my co-supervisor Kostas Magoutis. He has been a tireless source of inspiration, encouragement and ideas, but mainly I would like to thank him for his continuous enthusiasm and willingness to help me at every stage of this thesis. For all that and much more, I am grateful. During my studies in the Master's degree they introduced me to the notion of research, giving me the bases for the continuation of my studies and for my future career, so I am grateful to them.

I need to express my gratitude to the University of Crete and the Department of Computer Science for providing me with proper education; as well as the Institute of Computer Science of the Foundation for Research and Technology (ICS-FORTH) for supporting me.

I want to acknowledge Kuriakos, Giorgos, Chrysostomos, Panagiotis and Konstantina, members of PaaSage team for their valuable feedback on my work as well as Nikos and Giannis for the collaboration during the 2012 NOMS paper. I would also like to thank Antonis, Giorgos, Nikos, Panagiotis, Lazaros, Laertis, Ioasonas, Aris, Thanasis, Christos and Maria my colleagues at ICS-FORTH that made the countless working hours in the lab fun and interesting.

During my time in Heraklion, I have been fortunate to enjoy the friendship of a number of special people. I would like to give my appreciation to my friends, Tasos Papagiannis, Vassilis Papakonstantinou, Gogo Beletsioti, Zafi Tsibouli and Giannis Linardakis for the encouragement and the support during my undergraduate and graduate studies as wll as for the great moments we had together during these years. Also, I would like to thank Smaragda Asimaki who has always been there for me, supporting me in all possible ways.

I need to express my appreciation to my aunt Despoina and my uncle Giorgos for their support and care during all the years in Heraklion.

Last but not least I am grateful to my father Alexandros, my mother Maria as well as my brother Giorgos for the encouragement and the support in every single aspect of my life.

Antonis Papaioannou

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing [1] has become a dominant model of infrastructural services replacing or complementing traditional data centers and providing distributed applications with the resources that they need to serve Internet-scale workloads. The Cloud computing model is centered on the use of virtualization technologies (virtual machines (VMs), networks, and disks) to reap the benefits of statistical multiplexing on a shared infrastructure. However, these technologies also obstruct the details of the infrastructure from applications. Thus is very difficult to select the feasible or even the optimal resources for the application deployment. The problem of how to systematically explore a large collection of past execution histories of a multitude of real deployments of applications is less well studied. Such a facility has the potential of answering a variety of key questions about the runtime behavior of different deployments of complex applications, such as: which infrastructure (Cloud providers and/or types of resources) works best for certain applications? What are the most cost-effective options between a variety of configurations I (or my user community) have tried in the past? What rules or policies have been effective in achieving goals (or equivalently in addressing runtime issues) during past executions?

Integration of application modeling and deployment/operations environments has been gaining traction recently. For instance, the popular GitHub code repository supports adding a Cloud provider as a *remote* application repository, on which applications can be pushed for deployment. An interoperable Cloud provider (such as Heroku) will accept the push and receive the expected directories and files, deploying the application. Bringing together developer and operation teams is the goal of a new wave of *DevOps* platforms

such as Chef [2] and Puppet [3]. What is missing from such environments is a well-integrated feedback loop including monitoring information, SLA assessments, etc., support for storage and analysis of potentially vast histories of past executions, and awareness of Multi-Cloud deployments. Support for such an integrated loop is a key goal of the architecture presented in this thesis.

Handling heterogeneity in the infrastructure has been a challenging undertaking since the early days of data centers. A variety of hardware vintages and suppliers selected to improve cost-efficiency over time complicates the resource picture. The state of things in the Cloud space today is not much different: heterogeneity is ever present in the form of different types of resources offered from different Cloud providers. What is more, virtual resources carry nominal or indicative characterizations of their capabilities, making comparison inaccurate. To improve on the current state of affairs, we propose and implement a Cloud-independent classification scheme where virtual machine (VM) types are grouped with similarly-rated types across Cloud providers, according to a specific dimension such as CPU, memory, or I/O capability. The VM ratings are computed using a vector-driven approach taking into account individual micro-benchmarks on the VMs. Rating is sampled across regions and periodically repeated to capture variations and changes over time in the underlying hardware of the Cloud provider. Classification may also be repeated at different times to apply different criteria on the VM ratings (such as how many classes to group VMs into).

The architecture presented here combines a unique set of features not found in existing systems. It shares the principle of modeling application structure and deployed resources with systems such as SmartFrog [4], CloudML [5], TOSCA [6], models@runtime [7] and Cloudify [8]. It features detailed component-based monitoring of application performance typically found in Application Performance Management systems [9] and products such as IBM Tivoli Composite Application Manager. While addressing important problems in an application's lifecycle (such as determining problem root-causes) such systems do not offer a way to benefit from a potentially vast past experience in order to improve future deployments. Mining past histories to gain knowledge in the form of determining rules or detecting and ranking anomalies has been applied in the case of data center event collections [10, 11], in discovering configuration errors [12, 13], and in modeling performance characteristics of desktop applications [14]. Our approach extends to complex

distributed applications and covers deployment and lifecycle aspects such as elasticity and resource classification that are relevant to Multi-Cloud setups.

1.1 Thesis Contributions

- A Cloud-independent resource classification scheme that groups Cloud resources based on their benchmarked performance along three dimensions (CPU, memory, I/O)
- An information model capturing application development and deployment, requirements and goals, allocated infrastructure resources and runtime monitoring, along with characteristics of application executions over time
- An evaluation of data collected from executions a typical multi-tier distributed application (SPEC jEnterprise2010) over traditional (single Cloud) and Multi-Cloud setups, for a variety of use-case scenarios

1.2 Thesis Organization

The rest of this thesis is organized as follows: Section 2 provides background and related work that this work is based on. Chapter 3 presents the architecture, the design and implementation of the framework. Chapter 4 presents experimental analysis and evaluation of this work. Finally, Chapter 5 summarizes the work of this thesis and draws conclusions and future work.

Chapter 2

Background

2.1 Cloud computing model

Cloud Computing is a quite new term for a long-held dream of computing as a utility [15], which has recently emerged as a commercial reality. A cloud service has three distinct characteristics that differentiate it from traditional hosting. It is sold on demand, typically by the minute or the hour; it is elastic – providers supply their resources on-demand from their large pools installed in data center and users can use as much as they need at any given time; and the service is fully managed by the provider.

Cloud computing providers offer their services according to three fundamental models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). IaaS is the most basic cloud-service model and each higher model abstracts from the details of the lower models as shown is figure 2.1. According to this model providers offer access to resources such as virtual machines (VM) to their clients. PaaS is defined as a set of software and product development tools hosted on the provider's infrastructure. It also offers an execution runtime environment where users can deploy their applications without the need to choose the underlying operating system, the software and the hardware resources. SaaS model provides to users access to application software services. The infrastructure and the platforms that run the applications.

A cloud can be private or public. The main difference between these two models regard the targeted users who can use the offered resources rather than technical characteristics. A private cloud infrastructure operates solely within single organization. A public cloud is open to anyone and their ser-

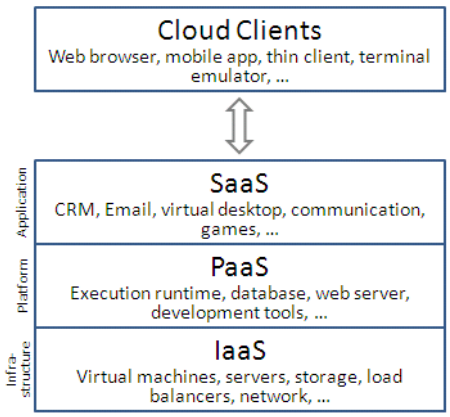

Figure 2.1: Cloud computing layers [16]

vices are rendered over a network. A third model is the hybrid cloud which includes the composition of at least one private and one public cloud that remain unique entities but are bound together. This model is useful when an application is deployed on a private cloud and "bursts" to a public cloud when the demand for computing capacity increases. Multi-Cloud deployment is a more general term and is used to describe that an application is deployed on many different clouds. Is is not necessary to include public and private clouds.

2.2 Heterogeneity of resources

Cloud computing is having a transformational effect on enterprise IT operations, software engineering, service-oriented computing, and society as a whole. Many Cloud computing providers today offer their services according to several fundamental models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). In addition there are approaches towards *Multi-Cloud* [17] which focus on the need to break the current lock-in experienced by application developers on the Cloud provider they design for and deploy on, and to allow them to simultaneously use (and seamlessly arbitrate between) several Cloud providers. As a result heterogeneity is ever present in the form of different types of resources offered from different Cloud providers. In addition the description of the resources usually include simple properties (such as the number of virtual cores) which doesn't provide an insight of the actual performance. A few Cloud providers have

defined their own metrics (Amazon EC2 units [18], Google Compute Engine units [19]); however these doesn't allow the comparison of resources across providers.

Ang Li et. al [20] propose a framework that tries to characterize the services offered by various cloud providers into a set of common service interfaces, and benchmarks the performance costs of these services. They focus on the common set of resources that provide different Cloud IaaS as well as PaaS cloud providers thus limiting their framework to use benchmarks written in Java.

Garfinkel studies the performance of Amazon Simple Storage Service (S3) and describes his experience in migrating an application from dedicated hardware into the cloud [21]. Walker investigates the performance of scientific applications on Amazon EC2 using both macro and micro benchmarks [22].

Compared to the work above, our approach is focused on the resources offered by IaaS Cloud providers. We identify a common set of resources that every IaaS provider offers. Thus our system could use a variety to build a multidimensional profile of the performance of a VM focusing on three major areas: CPU, disk, memory. As a result we set the common baseline for comparison and ranking of resources across providers. What is more we propose a Cloud-independent classification scheme in order to group VMs with similar performance. The classification clusters can be reconsidered once users want to apply different criteria on the measured ratings (such as the number of classes to group VMs) or combine more than one dimensions. Our comparative study include some of the most popular and representative providers but is generic enough and can be applied to every IaaS Cloud.

2.3 Infrastructure and application Modeling

There is a need of a cloud independent model to express the offered resources and their corresponding properties as well as the application structure, its Service Level Objectives (SLO) and the events regarding its lifecycle. There are a few efforts in this area described below.

CloudML [23] is a domain-specific language that supports expressing the deployment requirements and infrastructure descriptions. CloudML includes a runtime that performs provisioning actions on the Cloud provider deemed most suitable. As depicted in Figure 2.2, application requirements are mod-

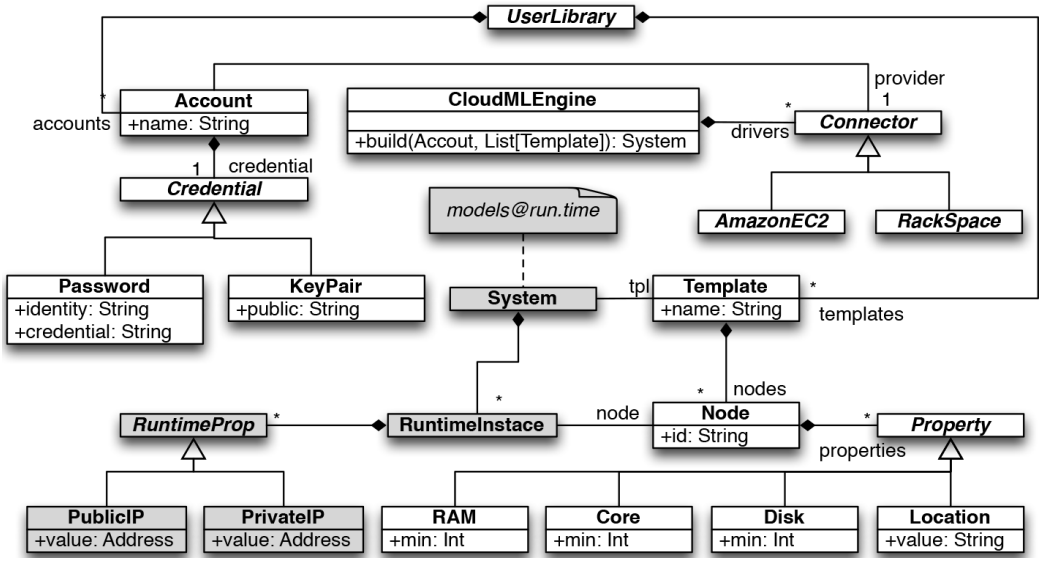

Figure 2.2: CloudML Metamodel [23].

eled as templates consisting of nodes that need to be provisioned in the Cloud. During the provisioning process, an engine creates instances for each node and attempts to find Cloud resources for each one of them. Runtime instances follow the models@run.time [7] approach, which allows the user to query their status or any other runtime-related information. Once provisioning is completed and all runtime instances have started, deployment can be performed. CloudML does not support Multi-Cloud provisioning: a single application cannot be spread across multiple providers. Finally, CloudML node models are characterized by simple textual descriptions of RAM, CPU, disk and location properties and thus cannot support advanced reasoning for Multi-Cloud cross-platform provisioning decisions.

PIM4Cloud [24] is another platform independent language devoted to the modeling of both private and public Cloud infrastructures through the description of the resources exposed by these infrastructures and required by a specific application. PIM4Cloud enables the expression of the intent of a service model without capturing its realization in a runtime framework. It can be exploited in discovering a Cloud provider that offers the resources required by a specific application. A limitation of PIM4Cloud however is that it ignores other phases of an application's lifecycle. Moreover, no deployment artifacts are associated with Cloud nodes, component dependencies are not modeled and the language does not support non-functional and QoS

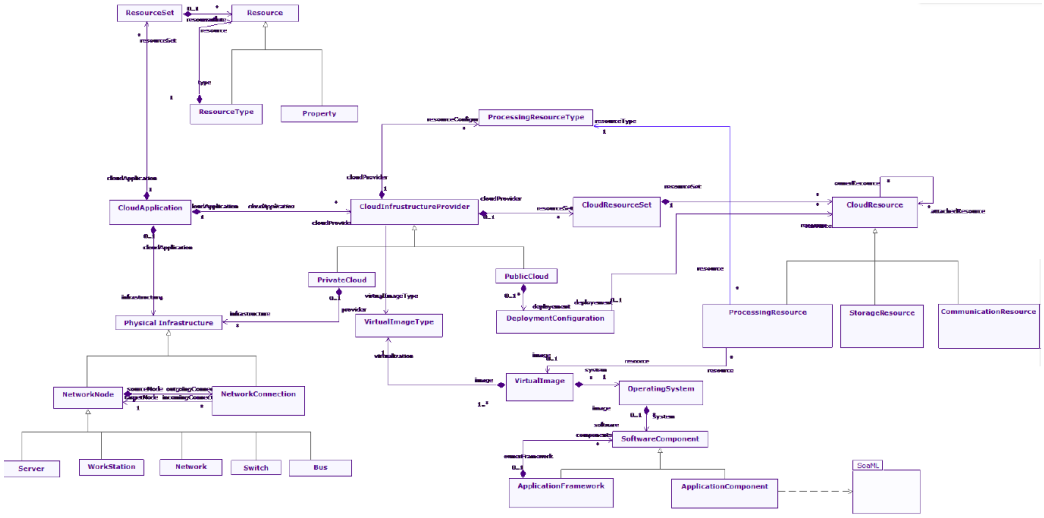

Figure 2.3: PIM4Cloud Metamodel [24].

requirements or capabilities. Additionally, its infrastructure descriptions are too low-level (e.g., CPU frequency).

A modeling approach must address the complete Cloud application lifecycle through appropriate processes and plans. TOSCA [6] is such a recent specification language, derived from a long line of work in software provisioning, deployment and management of distributed services. TOSCA is a middle-level language for the specification of the topology and orchestration of an IT service in the form of a service template. The language focuses on the semi-automatic creation and management of an IT service independent of any Cloud providers. The management of an IT service is achieved through the specification and execution of *process models* (BPMN, BPEL), which define an orchestration of services. Three types of plans are envisioned: build and termination plans associated to the deployment and termination of an IT service as well as modification plans associated to the IT service management. Each plan comprises tasks which refer to operations of either the interface of Node types or an external interface. The plans can be either placed inside the service specification or a reference to an external plan specification is used. Such plans can be specified by an existing process modeling language, such as BPMN or BPEL. In fact, the exploitation of BPMN plans is already supported in TOSCA. The topology of an IT service, which expresses the service structure, is described through the specification of nodes and their relationships. A node is described via various observed properties,

its requirements and capabilities, its management functions through its interface, while it also comprises deployment and implementation artifacts. It also declares the types of states it can expose and is associated to specific management policies. The implementation artifacts of a node are associated to the implementation of the node's interface operations, while the deployment artifacts are required during the node's instantiation. Relationships are used to denote node dependencies such that the requirements of one node are associated to the capabilities of a another node and there is a coupling of the nodes' interfaces. Their information includes the specification of properties and instance states. Figure 2.4 shows the main elements involved in the specification of a service template and their relationships.

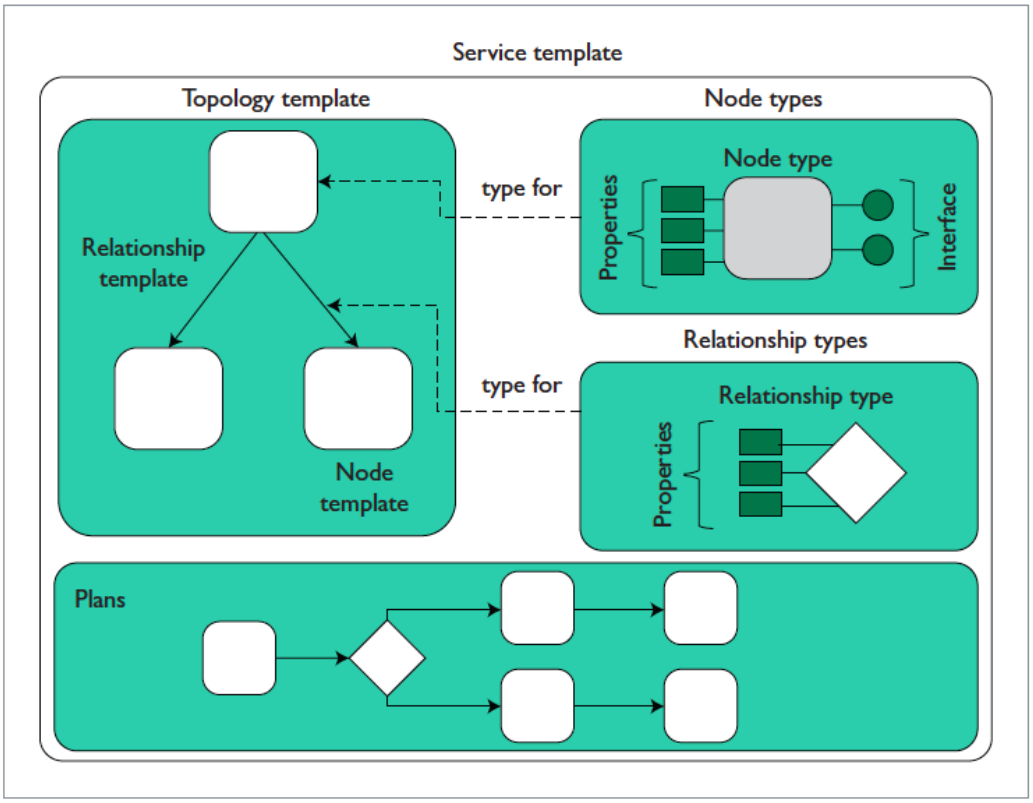

Figure 2.4: The TOSCA service template. Nodes represent the service's components, whereas relationships connect and structure nodes into the topology. Plans capture the operational aspects of the cloud service [6].

According to our approach we propose a metamodel that unifies properties of the previously described models and adds some extra features. Our

goal is to capture several aspects of application deployment and execution lifecycle. In more details it includes the application structure, requirements, goals and service-level objectives (SLO) as well as the Cloud provider characteristics and resource classification that are relevant to Multi-Cloud setups. In addition it features detailed component-based monitoring of application performance and lifecycle aspects such as elasticity.

2.4 Application management

The problem of evaluating a multitude of possible deployments of complex distributed applications on heterogeneous infrastructures with the aim of selecting feasible or even optimal ones dates from the early days of data centers [25]. Solutions to this problem often rely to either performance prediction models [26], simulations, or a limited set of experiments [27] to estimate performance of the underlying infrastructure under different load assignments to it.

There are a previous approaches to observing system executions over time trying to understand characteristics of the application. AppModel [14] aims at the construction of performance models to be useful for application developers to help expand application testing coverage and for IT administrators to assist with understanding the performance consequences of a software, hardware or configuration changes. The system monitors the behavior of an application on a machine and how the same application behaves under different configurations on different machines. PeerPressure [12] aims at troubleshooting a system using statistics from a set of sample machines in order to diagnose the cause of misconfigurations on a sick machine. The search can be performed over a "GeneBank" database that consists of a large number of machine configuration snapshots. Clarify [13] aims to improve error reporting of black-box applications by monitoring and generating a behavior profiles for applications. It uses statistics in order to classify these application's behavior histories in order to provide more precise error report than the output of the application itself. Clarify needs the binaries of the applications are already linked with Clarify runtime and the authors assume that Clarify-enabled binaries can be distributed along standard software distribution channels.

Our approach extends to complex distributed applications and the eval-

uation of their deployments in Multi-Clouds. At the core of the architecture we use an information metamodel that can model the application structure as well as the allocated infrastructure resources. The application does not require any modifications or instrumentation itself or its components as there is no need to export information regarding its state as most of the previous approaches. What is more our system supports detailed component-based monitoring application performance (e.g throughput) as well as Service Level Objectives (SLO). In addition it supports monitoring of the underlying resources used by the application. It doesn't make any assumptions of normal and abnormal behavior of the applications or any misconfiguration.

As a result it captures the history and evolution of the application lifecycle offering a way to benefit from experience in order to improve future deployments. This can be achieved by identifying which configuration settings and on which resources exhibit the best performance. Our system can be used in discovering cost-effective deployment plans, and in reasoning about elasticity policies under different assumptions

Chapter 3

Design

3.1 Architecture

The architecture of our system is depicted in Figure 3.6. It consists of three main components: the Classifier, the Explorer/Analyzer and the Metadata Database. The purpose of the Classifier component (described in detail in Section 3.2) is to assess and classify periodically the resources offered by different Cloud providers in a Cloud-independent manner. It subsequently updates the corresponding information in our metadata model described in Section 3.3. The Explorer/Analyzer component has two major goals. First, to explore the space of application configurations and deployment possibilities based on certain criteria. Second, to perform analysis of historical data from past executions in order to mine information about performance, cost, etc. The Explorer/Analyzer is aware of the structure of applications. The Explorer part decides on which VMs the application components should be deployed, based on certain requirements (e.g the J2EE application server component of the application should be deployed on a VM with CPU capability labeled -generically, and in a provider-independent manner– as LARGE). The Explorer stores these deployment plans as well as the SLA requirements and the elasticity rules of the application (if any) in the metadata model.

Clearly exploration can be a time-consuming task (since deployment and execution of each run can take hours or days, or even longer), so we assume that the exploration is a background activity that grows the historical database over time, rather than issued as a response to current time-sensitive queries. Optionally, part of the input to the historical database can be coming from a user community willing to share their deployment histories in

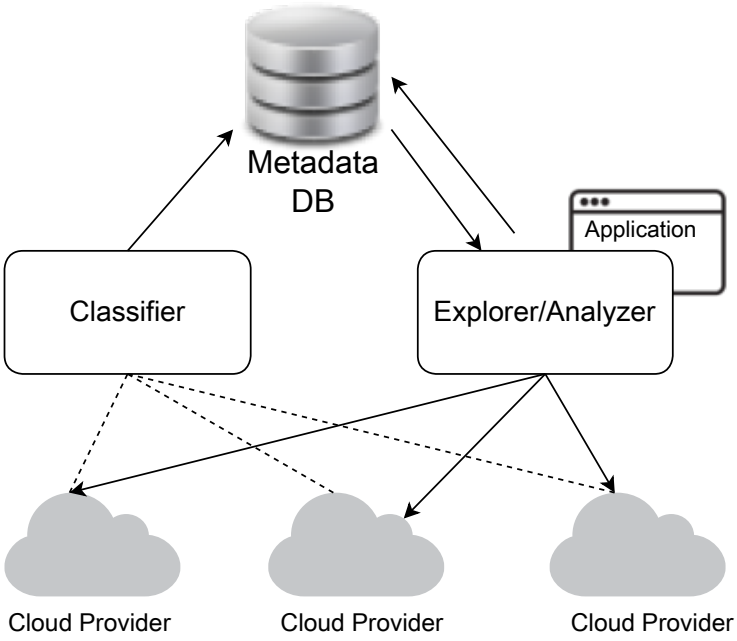

Figure 3.1: System architecture

open data repositories such as the Stanford Large Network Dataset Collection (SNAP) [28]. The Analyzer part can pose a variety of queries to the database to perform analytics, such as comparing the performance of an application across different deployments to determine the best performing or most cost-effective configuration settings.

3.2 Cloud-independent resource classification

To rank Cloud resources in a Cloud-independent manner we first create a profile for every VM type offered by various Cloud providers that describes the performance capabilities of the VM. In this way we can categorize resources to different classes of service, such as SMALL, MEDIUM, LARGE, etc. The VM profile is based on a vector of performance metrics focusing on three areas: CPU performance, memory size, I/O throughput. Our rationale in using a vector of performance metrics rather than a single benchmark was to simultaneously take into account multiple aspects of performance in ranking VMs. We use *k-means* cluster analysis [29] to classify resources based on benchmark results for each VM aspect. K-means takes as input the desired number of

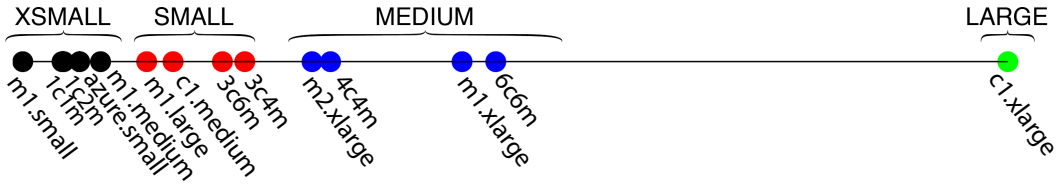

Figure 3.2: CPU performance classification of different VM types

clusters to group VMs in and performs the classification automatically.

Our experience showed that existing benchmark suites such as SPEC CPU2006 [30] satisfy our objectives, thus we rely on it for CPU performance characterization. To factor-in the benefits of core parallelism we use the *rate* metric of SPEC CPU2006 that measures aggregate throughput. Classification can be optionally refined by normalizing by the cost of resources. Figure 3.2 presents the *k-means* grouping of 14 VM types from three Cloud providers (Amazon EC2, Azure, Flexiant) based on CPU performance. We use the standard naming scheme for Amazon VM types. We developed our own naming scheme for Flexiant VMs based on two numbers: the first denotes number of cores and the second memory size (GB); e.g: 3c6m stands for 3 cores, 6GB memory. Our Azure academic account provided us with access to a single VM type, which we refer to as *azure.small* (1 core, 1.75GB). Our *k-means* analysis classifies the 14 VM types into four groups: XSMALL, SMALL, MEDIUM, LARGE. Amazon's *c1.xlarge* is alone in the LARGE class, whereas the other classes comprise 4-5 VM types each. A common theme in the XSMALL group is that all VMs in it have a single virtual core.

Classifying VM types based on memory size can be performed via the *k-means* algorithm, just as in the case of CPU performance. Figure 3.3 depicts three clusters SMALL, MEDIUM, and LARGE, created for the 14 VM types used in the previous case. Notice that the number of clusters created to classify VMs in each of the dimensions (CPU, memory, I/O) can vary.

Classifying VM types based on storage performance must ensure that all storage options offer the same consistency semantics. To reflect industry standard practices, we chose to evaluate VMs with remotely mounted storage (without optimizations) across all providers. We used the *hdparm* benchmark to measure average disk throughput and standard deviation over ten trials. We considered storage performance of the Amazon, Azure, and Flexiant FCO VM types shown in Table 3.1. I/O throughput to network storage seems to

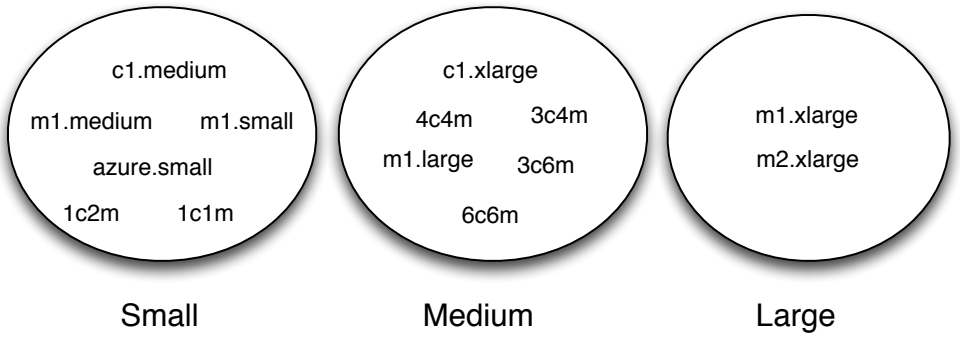

Figure 3.3: Memory size classification of different VM types

Provider	VM	T/put (MB/s)	Dev (%)	Avg (MB/s)	Class
Amazon	c1.medium	109.1	3.0	102	LARGE
	c1.xlarge	95.8	3.3		
	m1.small	78.7	5.9	73	MEDIUM
	m1.medium	70.9	10.7		
	m1.large	68.7	22.0		
Flexiant	1c1m	38.4	15.3	41	SMALL
	1c2m	40.5	13.5		
	3c4m	39.1	21.2		
	3c6m	41.1	16.7		
	4c4m	38.4	14.6		
	6c6m	45.0	6.3		
Azure	small	35.3	4.0	35	SMALL

Table 3.1: Disk throughput measurements and classification

fall into one or two narrow bands within each Cloud provider. It thus seems to depend more on the type of storage rather than type of VM (although smaller VMs spend a higher CPU share to do I/O at full speed). An exception is Amazon EC2 where *c1* instances seem to have access to higher-performing storage compared to *m1* instances. Table 3.1 shows the classification of the VMs into SMALL, MEDIUM, LARGE.

Finally we note that we consider network performance primarily a characteristic of the Cloud provider and to a lower extent a characteristic of a VM type, we thus treat it separately from the other three dimensions (CPU, memory, storage). We use netperf [31] as measurement tool to report network bandwidth according to previous work [32]. Table 3.2 summarizes the

Platform	Throughput (Mbps)
Flexiscale	750
Amazon	550
Azure	555
Rackspace	29.8

Table 3.2: Inter-VM bandwidth measurements

results of the evaluation of Cloud providers based on their network performance. Rackspace seems to limit -through explicit control at the network level or at the hypervisor- the VM network bandwidth to a fraction of the peak bandwidth supported by the network technology.

3.3 Metadata model

In this section we describe the metadata model at the core of our architecture (Figure 3.6). The model (whose schema in standard E/R notation is shown in Figure 3.4) is meant to capture the description of an application, its requirements and goals, rules and policies, and its provisioned resources, as well as runtime aspects of its execution histories such as monitoring information at different levels, invocations of rules and policies, and quality of service assessments. The model also captures Cloud provider characteristics, platforms, as well as users, roles, and organizations.

The system described in this paper is meant for long-term preservation for information. It is designed to associate mutations with a wall-clock timestamp and to trace the identity of the sources of mutations. It thus shares principles with archival systems [33], temporal databases [34], and provenance systems [35]. The information schema describes the applications and their deployment using principles from specifications such as CloudML [5], PIM4Cloud [24], and TOSCA [6]. The exposition of the metamodel provided here is necessarily concise and lacking detail, as an exhaustive presentation would require far more space than we have available in this paper.

3.3.1 Cloud provider properties

Cloud providers are described in CLOUD_PROVIDER object where we store information regarding the name, the geographic location of the data center

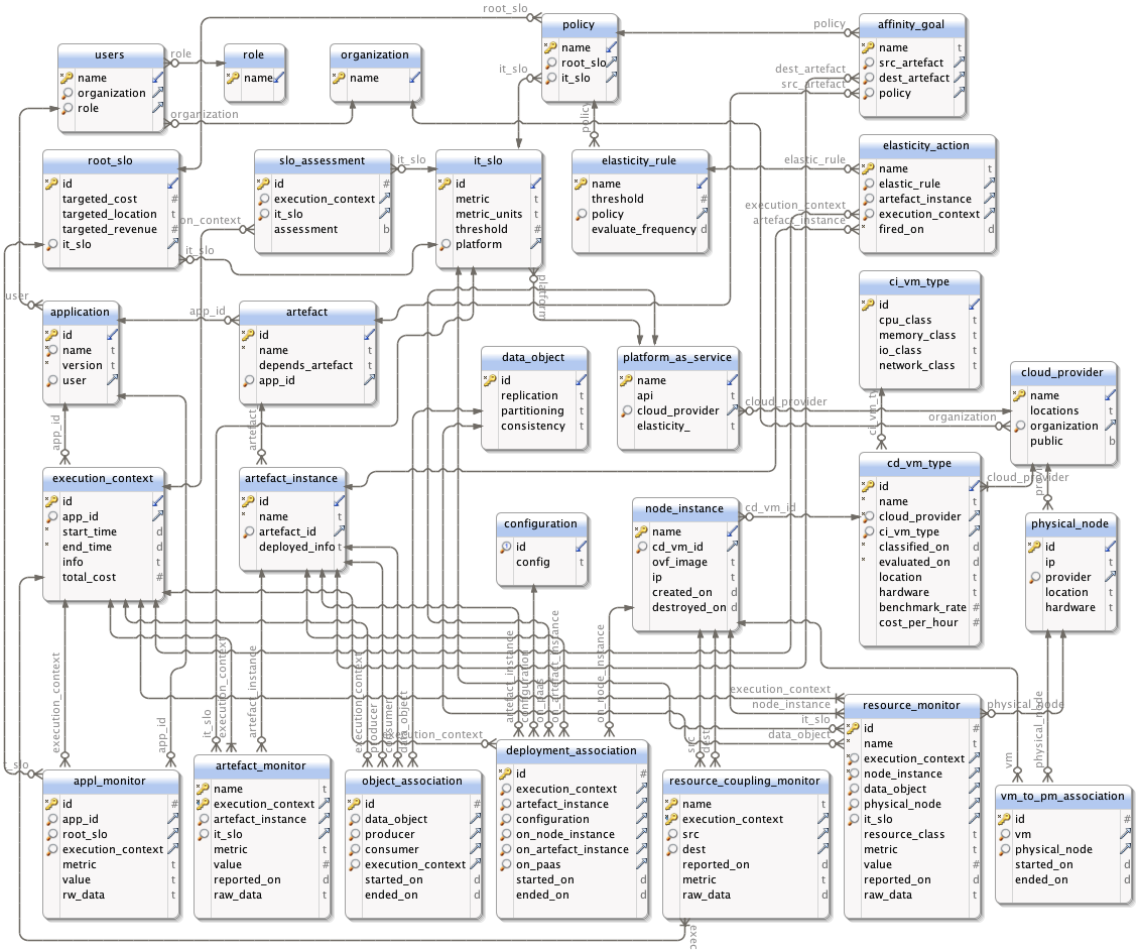

Figure 3.4: Metadata model

and whether it is a public or private provider. Information about a virtual machine instance such as the name, IP address, OS as well as its lifetime period is stored in ARTEFACT_INSTANCE object. Each NODE INSTANCE is of a particular CD_VM_TYPE and CI_CM_TYPE where CD stands for *cloud dependent* and CI for *cloud independent*. A CD_VM_TYPE describes a real-world VM type offered by a Cloud provider. In addition to the textual description of the properties of the VM (name, hardware, location, cost) this object keeps the score of the performance evaluation based on the benchmarking assessment described in Section 3.2. CI_VM_TYPEs are the result of (periodic) classifications described in Section 3.2. The platform services offered by a Cloud provider described in PLATFORM_AS_SERVICE object. In addition information about the physical infrastructure and the topology of the Cloud platform is modeled with the PHYSICAL_NODE and VM_TO_PM_ASSOCIATION objects.

3.3.2 Application

A version of an application is rooted at an APPLICATION object where we store the application name, its name and the responsible user. An application is comprised by software ARTEFACT and ARTEFACT_INSTANCE objects. An artefact type represents a generic component of the application (e.g. Java EE application server or a relational database). The artefact instance refers to a specific instance of an artefact type. For example the corresponding instance of the artefact Java EE application server and relational database could be JBoss AS 7 and MySQL 5.5 respectively. In addition applications may use DATA_OBJECTs (such as JavaBeans). These DATA_OBJECTs are exported (produced) and consumed between ARTEFACT_INSTANCEs. This association is represented by the OBJECT_ASSOCIATION object. Finally, users, roles, and organizations connected to the modeled entities are described in USERS, ORGANIZATION, and ROLES objects [36].

3.3.3 Requirements, Service Level Objectives & Elasticity

ROOT_SLO is used to express non-IT constraints such as overall cost, location, etc. IT_SLO objects can express a requirement on an IT metric [37]. This object can be used to express the objectives of the application or objec-

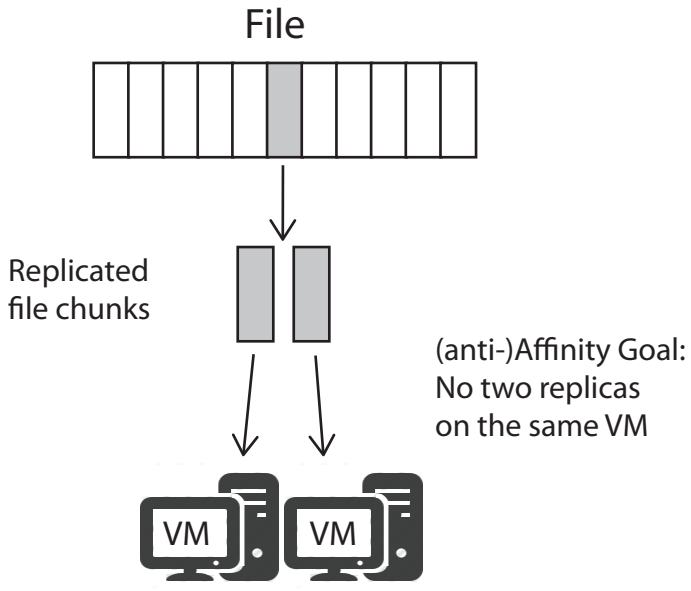

Figure 3.5: Affinity goal of a distributed file system

tives per ARTEFACT_INSTANCE. SLO_ASSESSMENT object can be used to express whether the SLO is violated or not. If the application has elasticity features the metadata model provides ELASTICITY_RULES and ELASTICITY_ACTION objects. In addition AFFINITY_GOAL expresses dependencies between artifacts. For example suppose that the application is a distributed file system. When a user wants to store a file, the application usually splits it to chunks. For each chunk the application stores two replicas concerning availability and durability reasons. The application policy requires each replica to be stored on a different VM. This kind of goal is modeled as an AFFINITY_GOAL of the application (figure 3.5).

3.3.4 Execution History

Information about each execution of an application is rooted at an EXECUTION_CONTEXT with a start and end time. Part of the execution history is the mapping between the software components of the applications (artefacts) and the corresponding resources they use. This correlation is represented with DEPLOYMENT_ASSOCIATION and OBJECT_ASSOCIATION objects. In addition an execution of the application may include monitoring information. APPL_MONITOR object is used to monitor the overall appli-

cation behavior while ARTEFACT_MONITOR concerns monitoring data for a particular artefact instance. On the other hand monitoring information regarding the resources can be modeled using RESOURCE_MONITOR and RESOURCE_COUPLING_MONITOR objects. The latter is used to monitor the communication per VM pairs.

3.3.5 Physical Resources

The Cloud computing model is based on the virtualization technologies where the physical resources are shared among many virtual machines exploiting the benefits of the statistical multiplexing. VMs can end up being collocated because either the allocation policy may in some cases favor this (e.g., when using large core-count systems such as Intel's Single-Chip Cloud (SCC) [38]) or because of limited choices available to the allocation algorithm at different times. While collocation may be desirable for the high data throughput available between the VMs, it is in general undesirable when an application wants to decouple VMs with regards to physical machine or network failure for the purpose of achieving high availability.

Several distributed applications such as data-intensive applications [39] [40] [41] which replicate data onto different system nodes for availability require the nodes holding replicas of the same data block to have independent failures. To decide which nodes are thus suitable to hold a given set of replicas, an application must have knowledge about data center topology, which is often hidden from applications or provided only in coarse form.

Apart from the virtual resources offered by the Cloud providers our schema can model the underlying physical nodes and their properties (PHYSICAL_NODE object) as well as the as the association between them and VMs that are hosted on (VM_TO_PM_ASSOCIATION object). Many applications that require information about the topology of the infrastructure in order to satisfy their goals. For example assume that the application described in section 3.3.3 requires to store identical replicas to VMs that do not share the same physical machine. In this case the application could achieve its availability and durability objectives exploiting the information about the infrastructure topology (figure ???). In section 4.3 we present a case study of such an application and propose a method which can extract information about the topology in public providers. In addition this kind of information is very useful for administrators of the Cloud infrastructure as they can identify

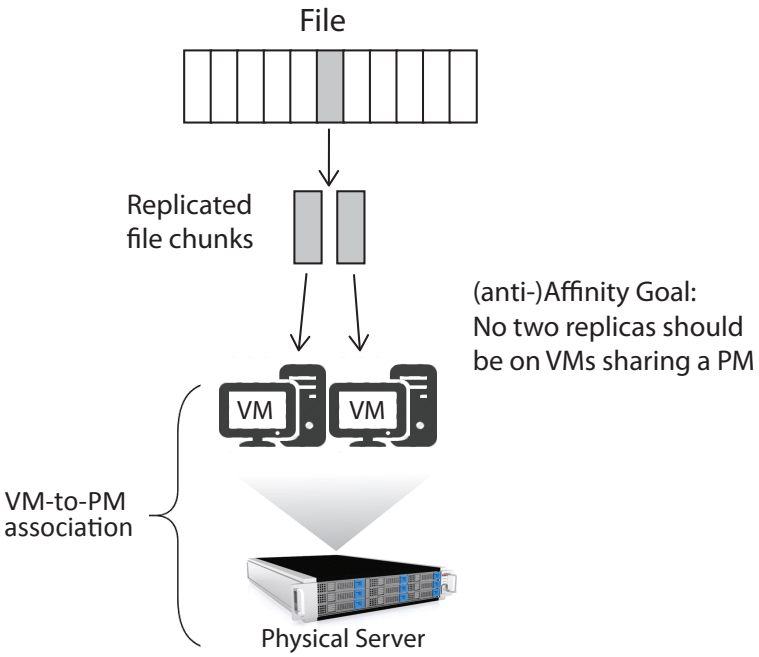

Figure 3.6: System architecture

which VMs hosted on the physical nodes. As a result it is straightforward to find out the applications affected by bad performance of a specific physical node or ever schedule VM migrations due to hardware maintenance.

3.3.6 Temporal Aspect

Objects that are related with the lifecycle of the deployed application (such as execution context, node instances associations objects) have a temporal aspect indicating that they are related with specific fact, an execution of that application. These objects contain data that vary over time as they model the history of reality. They are represented by *valid-time state tables*. The records of these tables are specified by two timestamp columns, one specifying when the row became valid and one specifying when the row stopped being valid. The intervening time is termed the *period of validity* of records. On the other hand monitoring objects and elasticity action object use a timestamp to capture the point in time when the data are reported. It doesn't meant to define a period of validity rather a snapshot in time.

The classifier component of our system updates the performance evaluation and its corresponding classification data of CD_VM_TYPE objects in

our metadata model periodically. Each update is marked with a timestamp. There is a *sequenced* constraint which applied at any point in time. The constraint requires there is only one performance rating score of each VM at any point in time. As a result we use only one timestamp to define periods of validity. In particular the timestamp defines the start of a new epoch of performance capabilities of the VM and at the same time marks the end time of its previous epoch.

Chapter 4

Evaluation

We evaluate our architecture using the distributed SPEC jEnterprise2010 benchmark [42] as a case study. To create a rich history of executions we deploy SPEC jEnterprise2010 to different Cloud providers under different deployment plans. SPEC jEnterprise2010 is a full system benchmark that allows performance measurement and characterization of Java EE 5.0 servers and supporting infrastructure. The benchmark models supply a chain consisting of an automobile manufacturer (referred to as the Manufacturing Domain) and automobile dealers (referred to as the Dealer Domain). The Web-based interface between the manufacturer and dealers supports browsing a catalog of automobiles, placing orders, and indicating when inventories have been sold. The SPEC jEnterprise2010 application requires a Java EE 5.0 application server and a relational database management system (RDBMS), which comprise the *system under test* (SUT). A typical configuration of the application is depicted in Figure 4.1.

The primary metric of the SPECjEnterprise2010 benchmark is throughput measured as jEnterprise operations per second (EjOPS). EjOPS are calculated by adding the metrics of the Dealership Management Application (Dealer Domain) and the Manufacturing Application (Manufacturing Domain) as:

$$\text{SPEC jEnterprise2010 EjOPS} = \text{Dealer transactions / sec} + \text{Work orders / sec}$$

A load generator (referred to as a Driver) produces a mix of browse, manage, and purchase business transactions at the targeted injection rate (or

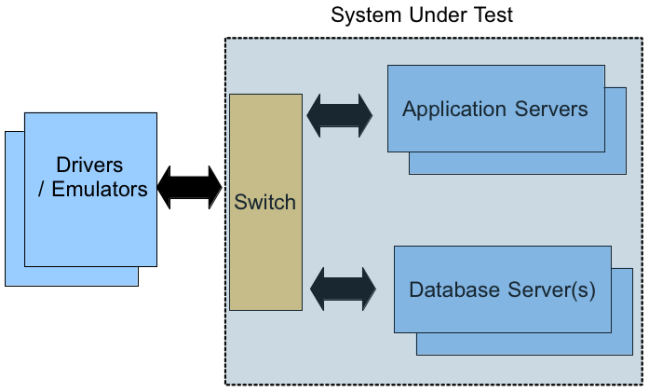

Figure 4.1: Typical configuration of SPEC jEnterprise2010 [42]

txRate, equal to the number of simulated clients divided by 10), which aims to produce the targeted EjOPS. The Driver measures and records the response time (RT) of the different types of business transactions. The RT refers to the time elapsed between the first byte the Driver sent in order to request a business transaction and the last byte received by the Driver to complete that business transaction. Failed transactions in the measurement interval are not included in the reported results. At least 90% of the business transactions of each type must have a RT of less than the constraint threshold (set to 2 seconds for each transaction type). The average RT of each transaction type must not exceed the recorded 90^{th} percentile RT by more than 0.1 seconds. This requirement ensures that all users see reasonable response times. For example, if the 90^{th} percentile RT of purchase transactions is 1 second, then the average cannot be greater than 1.1 seconds. The Driver checks and reports on whether the response time requirements are met during a run.

We model the SPEC jEnterprise2010 application using three artefacts (Section 3.3) corresponding to the business logic of the application, the application server, and the RDBMS. These artefacts are instantiated as *specj.ear* and *emulator.war* files, a JBoss 6.0 application server, and a MySQL 5.5 and their corresponding node instance are associated via the DEPLOYMENT_INSTANCES table, as shown in Figure 4.2. The IT_SLO on response time is set to 2 sec. Our deployment plans consider the deployment of the application and database artefact instances on the same node as well as on separate node instances (i.e., same VM vs. different VMs) on one or more providers. The back-end database was populated for a targeted dataset of up

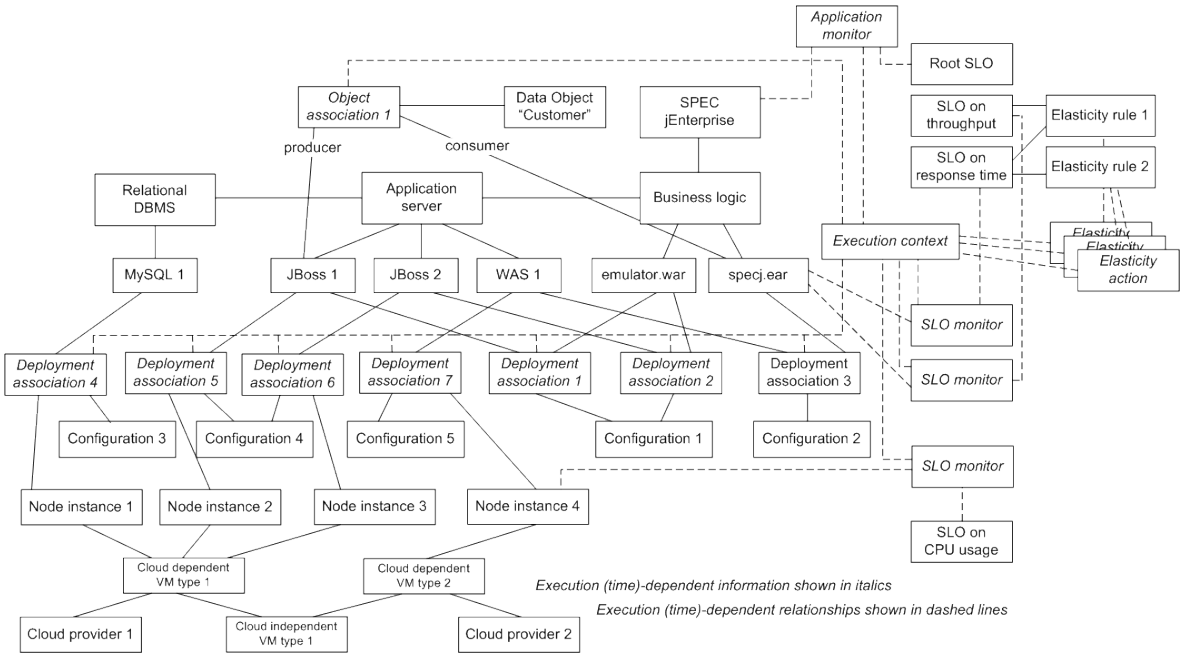

Figure 4.2: Deployment and execution model of SPEC jEnterprise2010

to 800 clients. For every execution of the benchmark we use its EjOPS metric and the reported RTs to assess the SLO requirements of the application during the run. Simultaneously, we monitor the resource usage (CPU and disk I/O) of the node instances used. We store the raw data in a time-series database and periodic averages into the metadata database. The latter is implemented on MySQL 5.1 RDBMS hosted on a Flexiant *2c4m*-type VM.

4.1 Typical use cases

Table 4.1 describes selected deployments of SPEC jEnterprise2010 on different VM types from Amazon, Flexiant and Azure chosen by the Explorer and/or contributed by the user community. All deployment scenarios considered here use an injection rate of 200 clients (TxRate 20). A specific use-case of our system is to mine past executions over a user-defined time interval, select those that satisfied their SLOs, and between those select the most cost-effective configuration options. The set of those configurations can be determined by executing the following SQL query to our metadata database.

Deployment Configuration	Artefact	Config Settings	VM type	Provider
1	App Server DB	Heap Size 2048MB Buffer Size 128MB	m1.large	Amazon
2	App Server DB	Heap Size 2048MB Buffer Size 2000MB	m1.large	Amazon
3	App Server DB	Heap Size 128MB Buffer Size 768MB	c1.medium	Amazon
4	App Server DB	Heap Size 768MB Buffer Size 128MB	m1.small	Amazon
5	App Server DB	Heap Size 2048MB Buffer Size 1100MB	3c4m	Flexiant
6	App Server DB	Heap Size 768MB Buffer Size 128MB	1c2m	Flexiant
7	App Server DB	Heap Size 768MB Buffer Size 128MB	small	Azure
8	App Server DB	Heap Size 6000MB Buffer Size 2000MB	m1.large m1.large	Amazon Amazon
9	App Server DB	Heap Size 768MB Buffer Size 2000MB	c1.medium m1.large	Amazon Amazon
10	App Server DB	Heap Size 768MB Buffer Size 2000MB	m1.small m1.large	Amazon Amazon
11	App Server DB	Heap Size 2048 MB Buffer Size 1100	3c4m m1.large	Flexiant Amazon

Table 4.1: Selected SPEC jEnterprise2010 deployment plans

Deployment Configuration	EjOPS	Cost ($)	Ratio
3	20.272	0.145	139
5	20.228	0.176	114
1	20.135	0.24	83.89
2	20.123	0.24	83.84
9	20.273	0.385	52.65
8	20.239	0.48	42.16

Table 4.2: Cost-effectiveness for successful executions (Ratio is price-normalized performance)

Listing 4.1: Identify the most cost-effective configuration options over the successful past executions

```
SELECT arm.execution_context, arm.value, ec2.total_cost, (arm.
    value/ec2.total_cost) AS ratio
FROM artefact_monitor arm, execution_context ec2
WHERE arm.name='EjOPS' AND ec2.id=arm.execution_context AND arm.
    execution_context IN
    (SELECT ec.id
     FROM execution_context ec
     WHERE app_id=1 AND ec.id NOT IN
        (SELECT DISTINCT(asses.execution_context)
         FROM slo_assessment asses
         WHERE asses.assessment=FALSE)
    )
   ORDER BY ratio DESC
```

The results of this query are summarized in Table 4.2. The execution offering the best absolute performance costs $0.385/hour whereas the most cost-effective deployment costs $0.145/hour (VM time is the only billable metric in our experiments). By selecting the most cost-effective rather than the best-performing selection leads to saving $0.24/hour or about $2,100 per year while still achieving SLOs.

Another use-case of our system is to determine typical causes for sub-optimal performance. For example, in executions whose SLOs are violated, a look into resource usage (e.g., CPU%) may point to the root cause. For example the results of the following query reveal that a saturated CPU (underprovisioning of some deployable artefact) is often an issue.

Listing 4.2: Identify suboptimal deployments (e.g. violate application's SLOs)

```
SELECT *
FROM resource_monitor rm
WHERE rm.name='cpu_idle' AND rm.execution_context IN
    (SELECT ec.id
    FROM execution_context ec
    WHERE app_id=1 AND ec.id IN (
        SELECT DISTINCT(asses.execution_context)
        FROM slo_assessment asses
        WHERE asses.assessment=FALSE)
    )
```

For example, in deployments where multiple VMs are used (e.g., Execution 9) we determine that the problem is using an Amazon *m1.small* VM for hosting the application server. The other VM, an *m1.large* hosting the database server, seems to be overprovisioned with its CPU always under 10% busy. A further look to the results of this query reveals that all deployments that involve any VM from the XSMALL cluster (regardless of Cloud provider) consistently lead to SLO violations. Therefore a deployment specialist would be advised to avoid VM types classified in that cluster in future deployments of the application. Another jEnterprise2010 configuration failing its SLO is #11. This is a case of a multi-Cloud deployment involving resources that, while adequate in a single-Cloud scenario, their tight coupling via frequent communication is expensive (highly latency, high cost) in a multi-Cloud scenario. The coupling can be determined through information in the RESOURCE_COUPLING_MONITOR object.

Another use of our system is in reasoning about variations in the service quality of Cloud providers over time. Such variations can be due to different levels of contention for hypervisor resources at different times or days of the week, month, or year. Another reason is the existence of different hardware infrastructures (such as CPU, network, and disk generations) in data centers (often within the same data center) of Cloud providers, as has been previously observed [43]. Figure 4.3 shows the results from collecting (in APPL_MONITOR and ARTIFACT_MONITOR objects) response-time measurements for SPEC jEnterprise purchase transactions over a 6-day period.

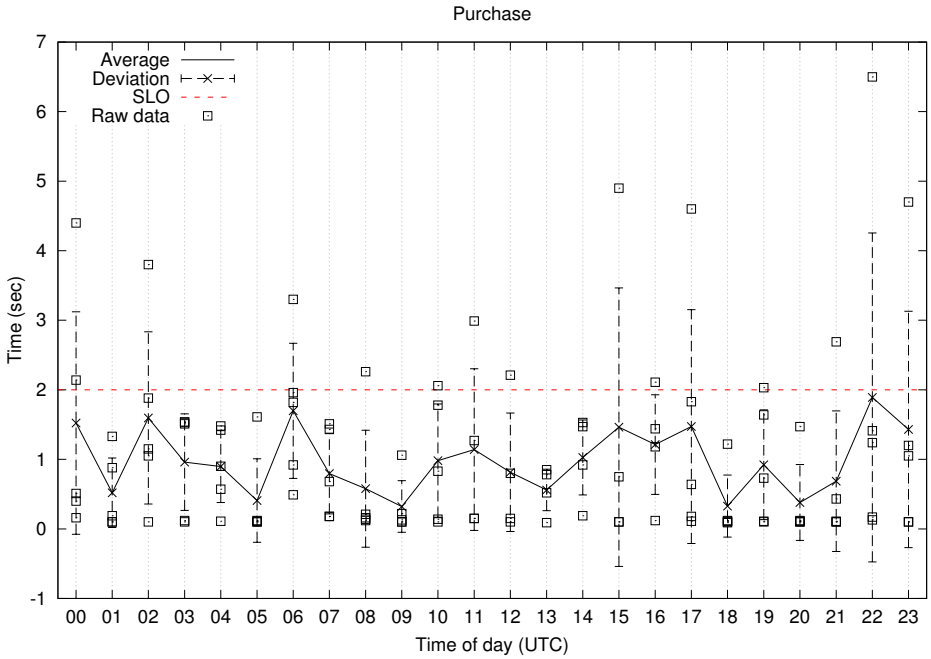

Figure 4.3: Response time of purchase transactions for hourly executions of jEnterprise2010 over a period of six days

4.2 Analysis of application elasticity actions

In this section we demonstrate how the Explorer can analyze the effectiveness of elasticity rules on application performance under different scenarios. Since SPEC jEnterprise2010 does not offer elasticity features in its standard setup, we modified it to place a load balancer (LB) frontend for distributing the load between several application servers as shown in Figure 4.4. Initially the deployment includes the LB, one VM hosting an application server, and another VM hosting the back-end database. The elasticity rule is triggered when the (periodically monitored) response time exceeds a SLO threshold (set by the benchmark at 2 sec); the action is to start a new application server VM and add it to the LB list.

During each run of our elastic SPEC jEnterprise2010 we collect all monitored information in our metadata database. The artifact and resource monitors report response time and CPU utilization every 20 seconds. Figure 4.5 shows the results of a run where the response-time SLO is initially violated due to an overloaded application server CPU. In this run, the load is set to 150 clients, the application server is hosted on an EC2 *m1.small* instance and

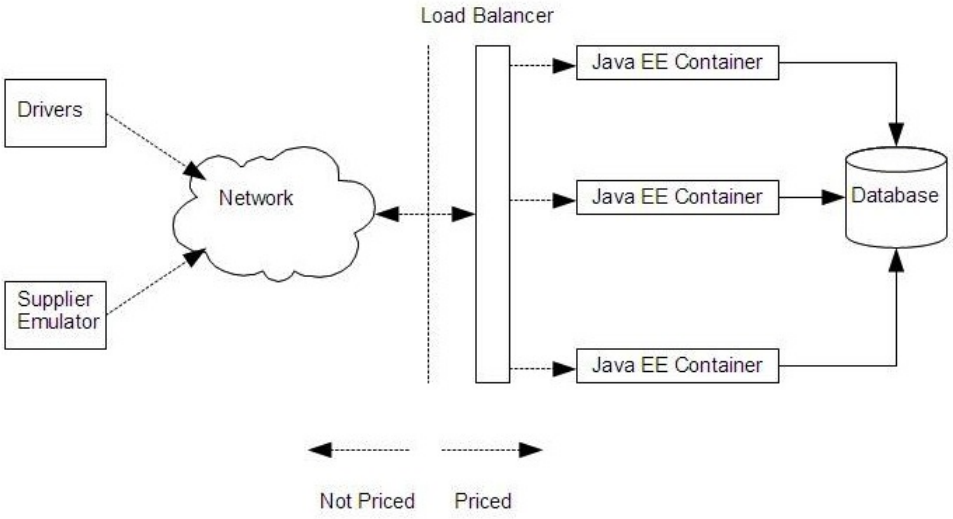

Figure 4.4: SPEC jEnterprise2010 with elasticity [42]

the back-end database servers is hosted on a *m1.medium* VM instance. After 60 sec, the elasticity rule triggers an elasticity action (adding a second application server on a new *m1.small* VM), eventually reducing response time to acceptable levels. Looking at CPU figures at the bottom of Figure 4.5 we can identify that CPU overload is indeed the cause of the performance bottleneck in this case.

In a different situation, such an elasticity action may not be effective. Figure 4.6 depicts performance of a deployment where an application server is deployed on a *m1.large* instance whereas its back-end database is hosted on a *m1.small* VM. The benchmark simulates the load of 350 clients. In this case, response time is unaffected by the elasticity action. A closer look at resource monitor data shows that CPU utilization was high but not a critical factor at neither the application servers or the database. In this case, network and I/O performance of the *m1.small* VM instance hosting the database back-end is the likely limitation.

4.3 Discover Physical Infrastructure Topology

In this section we describe how an application that requires information about the Cloud infrastructure topology could use our metamodel. Several distributed applications such as data-intensive applications [39] [40] [41]

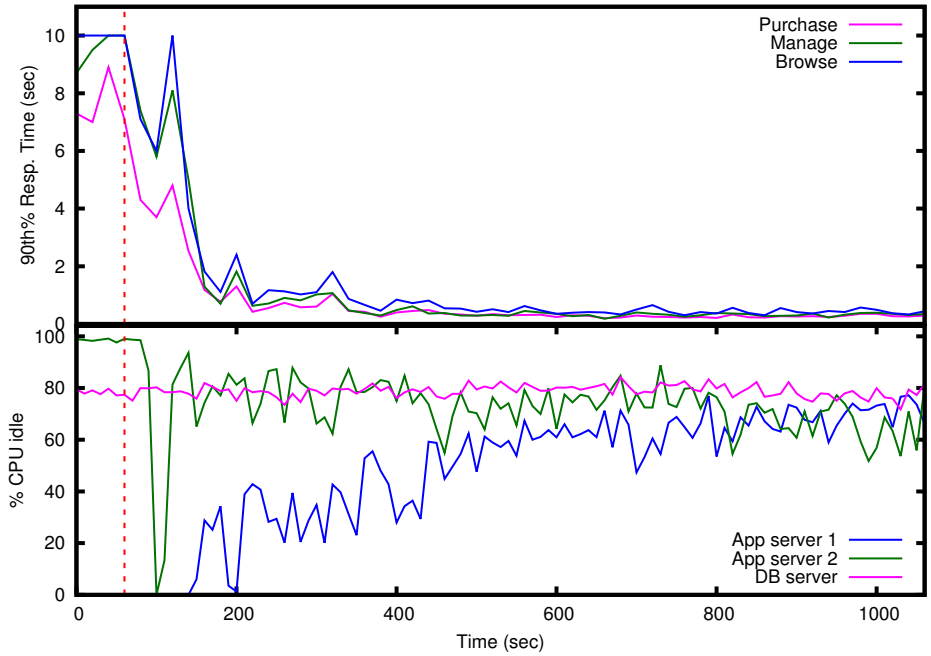

Figure 4.5: Successful invocation of elasticity rule

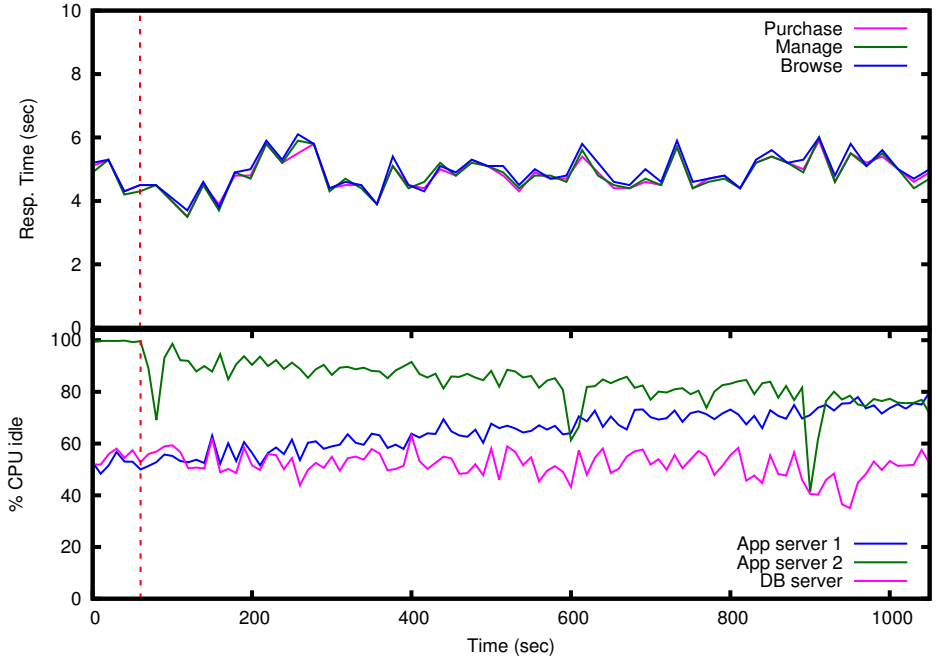

Figure 4.6: Unsuccessful invocation of elasticity rule

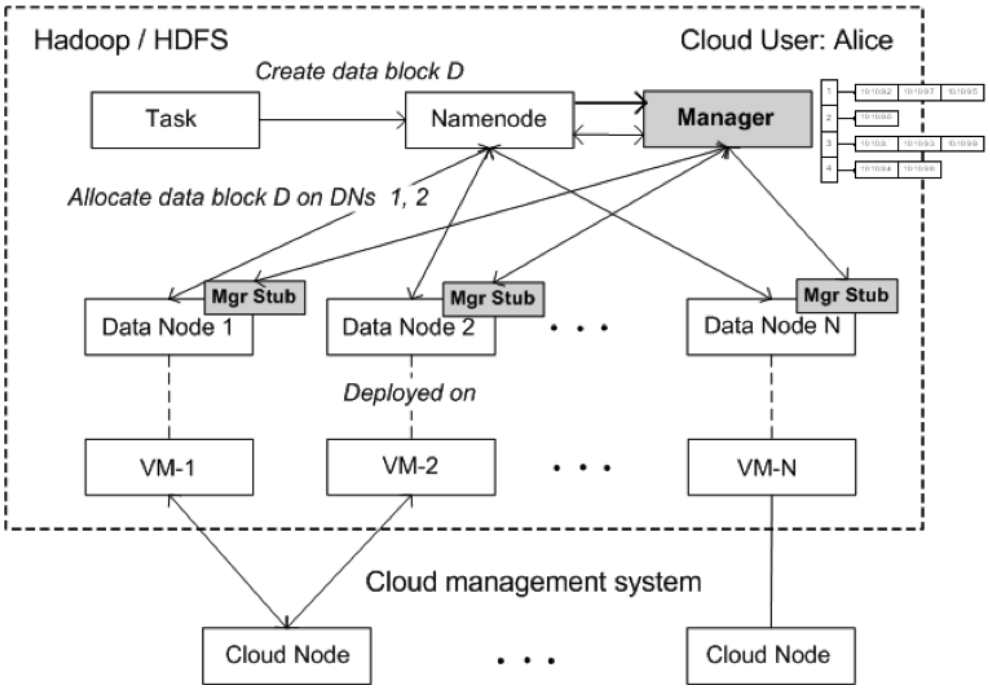

Figure 4.7: Integration of management system with HDFS

which replicate data onto different system nodes for availability require the nodes holding replicas of the same data block to have independent failures. To decide which nodes are thus suitable to hold a given set of replicas, an application must have knowledge about data center topology, which is often hidden from applications or provided only in coarse form. We choose the Hadoop file system (HDFS) as a case study, a core component of the Hadoop Map-Reduce [39] framework. As shown in Figure 4.7, HDFS consists of a name node (NN) and a number of data nodes(DNs). The NN manages the metadata (file and directory namespace and mapping of file blocks to DNs) and is responsible for the block placement and replication policy. The DNs are the nodes where file blocks replicas are stored. The NN and each of the DNs are hosted by a separate VM on a Cloud setup. We assume that DNs use the local storage resources of their underlying VMs-specifically locally-attached disks. Hadoop tasks perform some computation on blocks of data files. The overall principles in the design of Hadoop/HDFS are representative of a number of scalable data-intensive applications using a distributed replicated storage backend [44] [40] [45].

Cloud providers could export information about the VM collocations. We

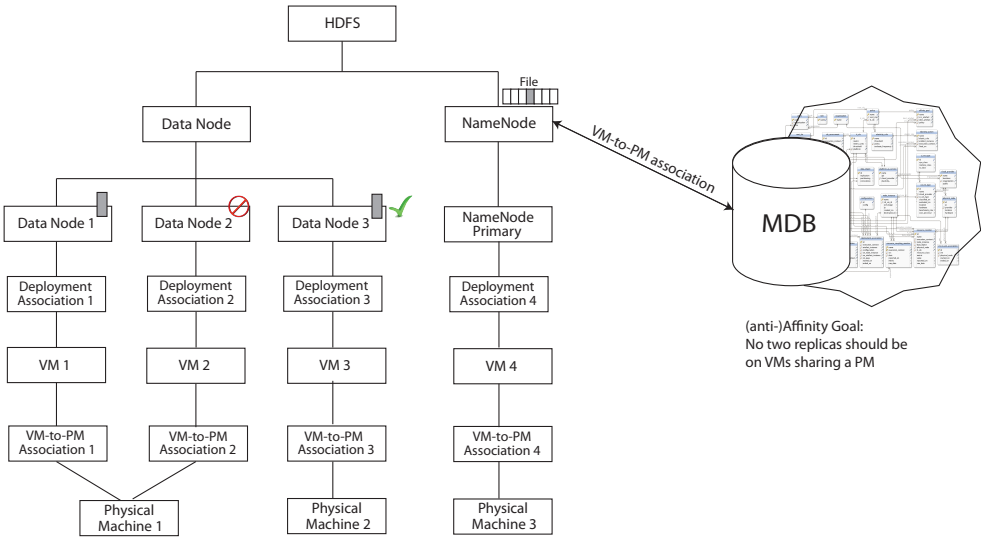

Figure 4.8: HDFS exploiting information about Cloud infrastructure topology

modify the HDFS namenode functionality taking into account this kind of information. In figure 4.8 we present the modeling of the basic components of the application according to our metamodel (for simplicity we omit details). In addition the application obtains information about the platform topology and thus can adapt its its behavior in order to achieve its affinity goals.

On the other hand most Cloud providers don't export information about their platform topology. As a result we extended the HDFS NN and DN source code to embed our manager and stub functionality (gray boxes of Figure 4.7). According to our approach the manager is a centralized component that orchestrates the operation of the system. We chose the term stub rather than agent or sensor (which would better fit their functionality) because they are embeddable into distributed components belonging to another middleware system (HDFS). We assume that each component (manager or stub) is deployed on a separate VM hosted by a specific physical machine (node) within the Cloud and that all VMs belonging to a Cloud user are hosting stubs.

The aim of the manager is to have up-to-date collocation information for all VMs and update the metadata schema. It learns about the existence of application VMs either through an explicit call or via starting a stub at that VM, which then registers with the manager. To maintain up-to-date col-

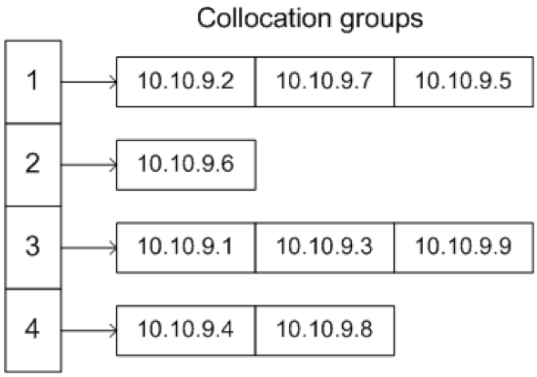

Figure 4.9: Representation of VM collocations of a Cloud user: 10.10.9.* are private IP addresses of VMs; 1-4 are identifiers of Cloud nodes hosting those VMs

location information the manager periodically instructs stubs to determine collocation relationships between themselves and other stubs according to a specific schedule. Each interaction between the manager and a stub for this purpose is called a discovery query (DQ). The manager subsystem responsible for scheduling DQs is the discovery query scheduler (DQS). A sequence of DQs that refreshes knowledge about all stubs in the system is called a scan and results in a representation of collocation relationships depicted in Figure 4.9. Boxes numbered 1-4 represent anonymous Cloud (physical) nodes. Sequences of 10.10.9.* boxes in each row represent VMs that are believed to be collocated within that Cloud node (we refer to such VMs as a collocation group). The typical usage of this management system is to help adaptive data-intensive applications drive policies such as task/data assignment and migration.

An important parameter in our system is how often to repeat a scan, which depends on the approach chosen. The manager uses a simple round-robin policy to assign DQs to stubs. Upon receiving a DQ a stub performs pair-wise measurements (in sequence) between itself and the set of stubs specified by the manager. When the measurements have been completed, the stub responds to the manager with the VM collocation relationships found. The manager (DQS) ensures that only a single discovery query is executing at a time to avoid interference with the discovery process and to reduce the impact on the executing application. However, external activity originating from other Cloud users is beyond the system's control and may impact measurements. Because of such interference the manager may dis-

cover that the information received from a DQ differs from the information it had previously discovered.

In performing a scan, the DQS chooses between two policies: I) trust collocation information found in a previous scan; or II) re-consider all collocation information. Assume that at the beginning of a scan there are N stubs in C collocation groups. In policy (I) the manager hands DQs to a representative stub from each collocation group and asks it to run measurements with representatives from each of the other collocation groups. The time to complete a scan is thus proportional to C^2. If we find a new collocation relationship between stubs x and y we merge their respective collocation groups. In policy (II) we consider any collocation information as potentially stale and thus schedule DQs with all stubs. For each stub we run measurements with representatives from each collocation group, including its own. The time to complete a scan in this case is thus proportional to N*C. We call the former an incremental scan and the latter a full scan. A special case of the full scan is the initial scan, where stubs are not known in advance but are registering with the manager as the scan progresses. In this case, DQS gives priority in scheduling DQs to stubs that have never previously been involved in a DQ.

In our experiments, VMs are reported as collocated if the average bandwidth between them exceeds 1Gbps. We used netperf [31] as the measurement tool to report bandwidth between nodes. We empirically determined that a 3-second run of netperf gives us a credible bandwidth estimate between any pair of nodes. We execute three such netperf runs and report the average bandwidth in table 4.3. This is a simple and accurate test based on the overwhelming dominance of 1Gbps Ethernet technology in the public Clouds we considered. However the more general assumption our system makes is not tied to a specific technology and is based on the observation that the memory bandwidth between collocated VMs is always greater than network bandwidth between non-collocated VMs. We believe this assumption holds true for the majority of commercial public or private Clouds. Even when considering Clouds using 10Gbit/s links at the node level, a VM will be capable of seeing only a fraction of that peak bandwidth, proportional to the share of system cores allocated to the VM. This is true even for 1Gbps links today, as previous studies [46] indicate and our results confirm. Additionally, several Cloud providers limit –through explicit control at the network level or at the hypervisor— the VM network bandwidth to a fraction of the peak bandwidth supported by the network technology. To the best of our knowledge, there

Platform	Non-collocated (Mbps)	Collocated (Gbps)
Flexiscale	750	4.0
Rackspace	29.8	5.2
Amazon	550	-

Table 4.3: Inter-VM bandwidth measurements

are no such limitations for the memory bandwidth between collocated VMs. Finally, it is highly unlikely that commercial Cloud providers would choose to maintain compute nodes whose memory system would be so dated as to violate the above assumption. In the rare case that the assumption is violated, use of alternative metrics (such as response time) can help in improving the accuracy of our results. It is important to note that even when our system is incorrect (labeling two VMs as collocated when in fact they are not), this does not hurt applications: Our HDFS prototype will allocate replicas of a block to VMs that are reported collocated if it has no other choice.

Interference with other Cloud activity may cause the miss of collocation relationships because of bandwidth or CPU sharing between VMs. The opposite however is not possible: A collocation relationship cannot be accidentally detected because of interference with other user workloads. Thus we believe that collocation relationships detected are generally accurate. Scans during periods of low Cloud workload will provide better quality results compared to scans over busy periods. Unfortunately prolonged periods of low load are rare in Cloud environments serving Internet-scale workloads around the clock. However brief periods of lower utilization are more common and as incremental scans coincide with them, we expect that the DQS will be correcting previous noise-induced errors over time (i.e., discover collocated VMs that were earlier determined to be non-collocated).

Incremental scans can achieve shorter DQ times by not re-considering past VM collocation information- this however may lead them to miss collocation changes that could have taken place via internal Cloud re-allocations such as VM migrations. Policy (II) can detect such changes but results into longer scan times. As Cloud re-allocations are rare and have typically long turnaround times we believe that the best approach is to use policy (I) in shorter time scales (e.g., hourly or daily) and policy (II) over longer time scales (e.g., weekly).

To conclude our metadata model is designed to hold information about

the mapping between VMs and its underlying physical nodes. This kind of correlation information can be obtained either by applications running in the VMs or coulb be exported by the Cloud provider via a Cloud management API as described in [32]. The metadata model could be updated periodically and applications could restore from it in case of a crash or in case of re-deployment using the same VM set. In addition applications' users can perform analytics concerning the VM collocation association and identify impact in its perfomance or functions (such as data migration). What is more the history of the correlation between VMs and physical nodes can witness VM migrations.

4.4 Managing database evolution over time

Our metadata schema (Figure 3.4) was designed to avoid redundancy when recording time-evolving state. They key to achieving this is to explicitly represent time-dependent associations, for example that of an application artefact with the resources used to deploy it in each execution of the application. If we recorded the time of deployment of every artefact within the ARTEFACT_INSTANCE table, we would need to create a new artefact instance for every execution of the application even though no other aspect of the artefact instance changed across executions. This design ensures that our metadata database grows at the reasonable pace. Similarly, we use the table ARTEFACT_CONFIG to correlate the configuration parameters of an artefact instance within an execution context.

To get a feeling of the rate of growth for the physical size of the metadata database in typical use cases, suppose the Explorer plans to deploy the SPEC jEnterprise application on 8 different node instances (VM types) across 3 Cloud providers. The Explorer further wants to try combinations of artefact configuration settings for each node instance. Examples of important settings for the JBoss application server and the MySQL RDBMS are JVM heap size and database buffer size. Trying 3 different values for each setting means the Explorer needs to execute the application at least 9 times on each VM type to cover the space of possible combinations. This would result into 72 executions of the application.

After the first execution the metadata database takes up 1.25 MB of storage, increasing to 1.31 MB after all 72 executions are in. Inserting data for another set of 72 runs of a second application (similar to the first) increases

the size of the database to 1.64 MB. Repeating for up to 100 applications (similar to the first application) increases the size to 29.3 MB. All applications are distinct and therefore they do not share artefacts, deployment instances (VMs), SLAs, monitors etc.

Taking into account the duration of each execution of SPEC jEnterprise2010 (one hour in our case) means that we will need at least 72 hours to test the application for the 72 runs we described above. When 100 similar applications simultaneously populate the metadata database, the storage growth would be about 30MB every three days. This equals 3.6 GB per year or 36 GB in 10 years, which is a modest requirement. Figure 4.10 depicts the storage growth trend as the number of applications increases. The execution time of queries in the metadata database increases with database size: The first query described previously takes on average 0.036, 0.384 and 3.55 seconds as the database size grows from 100, to 1,000 and 10,000 applications respectively.

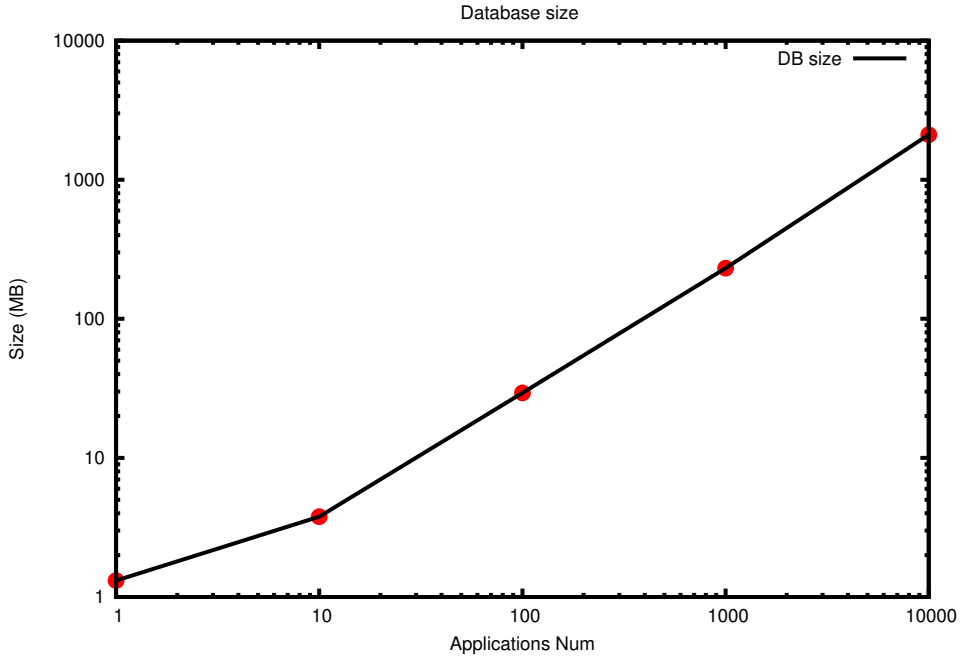

Figure 4.10: Growth of metadata database size with increasing number of applications

Note that this analysis does not take into account the raw monitor data that are typically stored in a time series database (TSDB). Only summaries

(e.g., average values) are periodically stored in the metadata database. In a typical setting, the raw data would be erased over time while the metadata database information would be preserved.

Chapter 5

Conclusions and Future Work

In this thesis outlined an architecture for evaluating distributed application deployments in Multi-Clouds. The information metamodel at the core of the architecture captures the history and evolution of distributed application deployments and can support a variety of interesting analytics. The proposed classification scheme is critical in enabling a cross-Cloud categorization of resources. Our evaluation using the SPEC jEnterprise2010 application benchmark exhibits the use of our system in discovering cost-effective deployment plans, and in reasoning about elasticity policies under different assumptions. We believe that the applicability of the architecture is broader and plan to exhibiting it further in our future work.

Appendix A

VM instance types

In this section we present the nominal characteristics of VM types as described by the corresponding providers.

Provider	VM	vCPU	Memory (GiB)
Amazon	m1.small	1	1.7
	m1.medium	1	3.75
	m1.large	2	7.5
	m1.xlarge	4	15
	m2.xlarge	2	17.1
	c1.medium	2	1.7
	c1.xlarge	8	7
Flexiant	1c1m	1	1
	1c2m	1	2
	3c4m	3	4
	3c6m	3	6
	4c4m	4	4
	6c6m	6	6
Azure	small	1	1.75

Table A.1: Instance Type Details

Bibliography

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[2] "Opscode Chef Automation Platform," http://www.opscode.com/chef, Accessed 6/2013.

[3] "Puppet Labs DevOps Platform," https://puppetlabs.com/solutions/devops/, Accessed 6/2013.

[4] P. Goldsack et al., "The smartfrog configuration management framework," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 16–25, Jan. 2009.

[5] N. Ferry et al., "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, ser. CLOUD '13. Washington, DC, USA: IEEE Computer Society, 2013.

[6] T. Binz, G. Breiter, F. Leymann, and T. Spatzier, "Portable Cloud Services Using TOSCA," *IEEE Internet Computing*, vol. 16, no. 3, pp. 80–85, 2012.

[7] U. Aßmann, N. Bencome, B. H. C. Cheng, and R. B. France, "Models@run.time (dagstuhl seminar 11481)," Dagstuhl Reports, Tech. Rep. 11, 2011.

[8] "Cloudify," http://www.cloudifysource.org/.

[9] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *Network Operations and Management Symposium, NOMS 2006. 10th IEEE/IFIP*, 2006.

[10] J. L. Hellerstein, S. Ma, and C.-S. Perng, "Discovering actionable patterns in event data," *IBM Systems Journal*, vol. 41, no. 3.

[11] K. Viswanathan et al., "Ranking anomalies in data centers," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 2012.

[12] H. J. Wang et al., "Automatic misconfiguration troubleshooting with peerpressure," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004.

[13] J. Ha et al., "Improved error reporting for software that uses black-box components," in *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '07. New York, NY, USA: ACM, 2007.

[14] Eno Thereska, Bjoern Doebel, Alice X. Zheng, Peter Nobel, "Practical Performance Models for Complex, Popular Applications," in *SIGMETRICS'10*, June 14-18, 2010.

[15] D. F. Parkhill, *The challenge of the computer utility.* USA: Addison-Wesley Professional, 1966.

[16] "Cloud computing, Wikipedia," http://en.wikipedia.org/wiki/Cloud_computing, Accessed 09/2013.

[17] G. Baryannis, P. Garefalakis, K. Kritikos, K. Magoutis, A. Papaioannou, D. Plexousakis, and C. Zeginis, "Lifecycle management of service-based applications on multi-clouds: A research roadmap," in *Proceedings of 2013 International Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud'13)*, April 2013.

[18] "Amazon web services ec2," http://aws.amazon.com/ec2/, accessed: 2013-08.

[19] "Google compute engine," https://cloud.google.com/products/compute-engine, accessed: 2013-08.

[20] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879143

[21] S. L. Garfinkel, "An evaluation of amazon's grid computing services: Ec2, s3 and sqs," Center for, Tech. Rep., 2007.

[22] E. Walker, "Benchmarking Amazon EC2 for high-performance scientific computing," *LOGIN*, vol. 33, no. 5, pp. 18–23, Oct. 2008.

[23] E. Brandtzæg, M. Parastoo, and S. Mosser, "Towards CloudML, a Model-based Approach to Provision Resources in the Clouds," in *8th European Conference on Modelling Foundations and Applications (ECMFA)*, H. Störrle, Ed., 2012, pp. 18–27.

[24] "REMICS Deliverable D4.1: PIM4Cloud," http://www.remics.eu/system/files/REMICS_D4.1_V2.0_LowResolution.pdf, 2012.

[25] J. Wolf, "The placement optimization program: a practical solution to the disk file assignment problem," in *Proceedings of the 1989 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 1989.

[26] R. R. Steffen Becker, Heiko Koziolek, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, pp. 3–22.

[27] J. Tordsson et al., "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, Feb. 2012.

[28] "Stanford Large Network Dataset Collection," http://snap.stanford.edu/data/, Accessed 6/2013.

[29] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observation," in *Proc. of 5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1967.

[30] "SPEC CPU2006 Benchmark," http://www.spec.org/cpu2006/, Accessed 6/2013.

[31] "Netperf," http://www.netperf.org/netperf/, Accessed 8/2013.

[32] I. Kitsos, A. Papaioannou, N. Tsikoudis, and K. Magoutis, "Adapting data-intensive workloads to generic allocation policies in cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 2012, pp. 25–33.

[33] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," in *Proceedings of the 1st USENIX conference on File and storage technologies*, ser. FAST'02. Berkeley, CA, USA: USENIX Association, 2002.

[34] R. T. Snodgrass, *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Series in Data Management Systems.

[35] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ser. ATEC '06. Berkeley, CA, USA: USENIX Association, 2006.

[36] "CERIF metadata model," Accessed 6/2013. [Online]. Available: http://www.eurocris.org/Index.php?page=featuresCERIF&t=1

[37] P. Bianco, G. Lewis, and P. Merson, "Service level agreements in service-oriented architecture environments," Software Engineering Institute, Tech. Rep. CMU/SEI-2008-TN-021, September 2008.

[38] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van der Wijngaart, and T. Mattson, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, 2010, pp. 108–109.

[39] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[40] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220.

[41] L. Koromilas and K. Magoutis, "Cassmail: a scalable, highly-available, and rapidly-prototyped e-mail service," in *Proceedings of the 11th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, ser. DAIS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 278–291.

[42] "SPEC jEnterprise2010 Benchmark," Accessed 6/2013. [Online]. Available: http://www.spec.org/jEnterprise2010/

[43] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings VLDB Endowment*, vol. 3, no. 1-2, pp. 460–471, Sep. 2010.

[44] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15.

[45] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[46] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 1163–1171.