AN AUTOMATED APPROACH TO CREATE, MANAGE AND ANALYZE LARGE- SCALE EXPERIMENTS FOR ELASTIC N-TIER APPLICATION IN CLOUDS

A Dissertation
Presented to
The Academic Faculty

by

Indika Deepal Jayasinghe

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
August 2013

AN AUTOMATED APPROACH TO CREATE, MANAGE AND ANALYZE LARGE- SCALE EXPERIMENTS FOR ELASTIC N-TIER APPLICATION IN CLOUDS

Approved by:

Dr. Calton Pu, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Shamkant B. Navathe
School of Computer Science
Georgia Institute of Technology

Dr. Ling Liu
School of Computer Science
Georgia Institute of Technology

Dr. João E. Ferreira
Institute of Mathematics and Statistics
University of São Paulo

Dr. Ed Omiecinski
School of Computer Science
Georgia Institute of Technology

Date Approved: June 21st, 2013

To My Wife and Parents

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest thanks to my advisor Dr. Calton Pu who tirelessly guided me throughout my entire graduate school career as (academic) advisor. I could not imagine achieving so much when I first came to him five years ago. Calton always quickly grasped my thinking and proposed interesting ideas even if I myself was still unclear and hesitant. Most importantly, Calton never failed to provide the resources and opportunities for me to succeed.

I would like to thank all my fellow members of ELBA for their continuous comments and support on my work. Especially, Simon J. Malkowski, Qingyang Wang, Junhee Park, Jack Li, Pengcheng Xiong, Danesh Irani, Yasuhiko Kanemasa, Tao Zhu, Chien An Lai, Josh Kimball and Chien-An Cho directly contributed to this dissertation by co-authoring my research papers. I feel so lucky to have them as my companions all the way.

I would like to give special thanks to, Dr. Ling Liu, Dr. Shamkant B. Navathe, Dr. Ed Omiecinski, and Dr. João E. Ferreira for serving on my dissertation committee and approving this work in its entirety.

My highest appreciation to my wife and parents. Thanks to my wife Krishani for supporting me by sharing happiness as well as my tears and motivating me all the way. My parents, too, were invaluable in their support; I thank them for their guidance and motivation to move down the path to the higher education.

Finally, in recognition of the funding without which this work could not have occurred: this research has been partially funded by National Science Foundation by IUCRC/FRP (1127904), CISE/CNS (1138666), RAPID (1138666), CISE/CRI (0855180), NetSE (0905493) programs, and gifts, grants, or contracts from DARPA/I2O, Singapore Government, Fujitsu Labs, Wipro Applied Research, and Georgia Tech Foundation through the John

P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

Thanks to all of the others whose stories are too long to be included in the limited space.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS OR ABBREVIATIONS

AJP	Apache JServ Protocol.
API	Application Programming Interface.
BLR	Business Logic Request.
CPU	Central Processing Unit.
DB	Database.
DBMS	Database Management System.
EJB	Enterprise Java Beans.
EM	Empirical Model.
FT	Fourier Transformation.
HSM	Horizontal Scale Model.
HTTP	Hypertext Transfer Protocol.
IaaS	Infrastructure as a Service.
IAE	Integrative Adaptation Engine.
IP	Internet Protocol.
IT	Information Technology.
JDBC	Java Database Connectivity.
JVM	Java Virtual Machine.
KPI	Key Performance Indicators.
LAMP	Linux, Apache, MySQL, and PHP.
ODS	Operational Data Store.
QoS	Quality of Service.
RMS	Resource Management System.
SLA	Service Level Agreement.
SLM	Service Level Management.
SLO	Service Level Objective.

SQL Structured Query Language.

TPC Transaction Processing Performance Council.

URL Uniform Resource Locator.

VM Virtual Machine.

WAL Write Ahead Log.

WFM Workload Forecast Model.

GLOSSARY

Apache An open-source HTTP server that provides extensible hypertext services according to the current HTTP standards.

C-JDBC Clustered Java Database Connectivity, an open source database cluster middleware that allows Java applications to transparently access a cluster of databases.

CloudXplor A research prototype of a novel tool for configuration planning in clouds based on an iterative and interactive refinement process of empirical data.

EC2 Amazon Elastic Compute Cloud, `Amazon.com`'s cloud computing offering that provides highly scalable computing capacity in the cloud and allows web-users to flexibly rent virtual hardware based on a commodity-pricing model.

Elba A research project at the Center for Experimental Research in Computer Systems (CERCS) at Georgia Tech, addressing the challenges in the automation of large application system management. In particular, this research encompasses design through deployment to production and capture of application monitoring, evaluation, and evolution.

Emulab A network testbed that provides a wide range of computing environments for development and evaluation of computer systems.

Linux A Unix-like computer operating system, distributed and developed as open source software.

MySQL An open source relational database management system that runs as a server providing multi-user access to a number of databases.

MySQL Cluster An open source real-time transactional relational database with high availability, "shared-nothing" distributed architecture with no single point of failure.

PostgreSQL An open source object-relational database management system targeting enterprise class database environments.

RUBBoS Rice University Bulletin Board System, an open source n-tier application benchmark modeled after the website `http://slashdot.org/`.

RUBiS Rice University Bidding System, an open source n-tier application benchmark modeled after the website `http://www.ebay.com/`.

SysViz A trace analysis tool, developed at the Fujitsu Laboratories, able to reconstruct entire traces of executed transactions in n-tier application systems.

Tomcat Apache Tomcat is an open source application server based on an implementation of Java Servlet and JavaServer Pages technologies.

SUMMARY

Cloud computing has revolutionized the computing landscape by providing on-demand, pay-as-you-go access to elastically scalable resources. Many applications are now being migrated from on-premises data centers to public clouds; yet, the transition to the cloud is not always straightforward and smooth. An application that performed well in an on-premise data center may not perform identically on public computing clouds because many variables, including virtualization impact the performance. A natural and presumably unbiased approach to reveal the cloud's complexity is to collect significant performance data by conducting more experimental studies. However, conducting large-scale system experiments is particularly challenging because of the practical difficulties that arise during experimental deployment, configuration, execution and data processing. In spite of these associated complexities, we argue that a promising approach for addressing these challenges is to use exhaustive measurements of large-scale experiments through automation.

Automation removes the error prone and cumbersome involvement of human testers, reduces the burden of configuring and running large-scale experiments associated with distributed applications, and accelerates the process of reliable applications testing. In our approach, we have automated three activities of the key experiment measurement processes: *create*, *manage* and *analyze*. During *create*, we prepare the platform, deploy, and configure the application. In *manage*, we start the application components in the correct order, execute workloads, collect resource monitoring and other performance data, and parse and upload the results to the data warehouse. In *analyze*, we process the collected data using various statistical and visualization techniques to understand and explain performance phenomena. In our approach, a user provides the experiment configuration file and framework does everything so that at the end user sees the results. We enable the automation through

code generation. From an architectural viewpoint, our code generator adopts the compiler approach of multiple serial transformation stages; the hallmarks of this approach are that stages typically operate on an XML document that is the intermediate representation, and XSLT performs the code generation. Our automated approach for large-scale experiments has enabled cloud experiments to scale well beyond what have been possible through manual experimentation, and it has enabled us to identify several non-trivial performance phenomena that would not have been possible otherwise.

CHAPTER I

INTRODUCTION

One of the most significant distributed computing models to revolutionize the computing landscape is cloud computing. Clouds enable service-oriented, on-demand, and pay-as-you-go network access to elastically scalable resources. These resources include services in the forms of infrastructure, platform, and software. In the cloud paradigm, clients do not own these resources but are given universal access to them through standard APIs and protocols.

As companies migrate applications from traditional data centers to private and public cloud infrastructures, companies must ensure that their applications are afforded a safe and smooth transition from their existing (and likely stable) environments to the cloud. An application that performs one way in the data center may not perform identically in computing clouds [174], which runs counter to companies needing to maintain a high-level of confidence that their applications will scale and deliver an expected level of performance, regardless of the current technological deployment strategy. Neglecting the performance impacts of cloud platforms can lead to unforeseen performance problems. Companies who are negligent (or simply ignorant) of this possibility could experience lower user satisfaction, missed SLAs, and even, lower operating income. For instance, a study by Amazon reported that an extra delay of just 100 milliseconds could result in roughly 1% loss in sales [207]. Similarly, Google found that a 500ms additional delay in returning search results could reduce revenues by up to 20% [194].

The current trend of running applications on top of cloud platforms poses significant challenges to anticipating and predicting an application's performance, because many

variables impact an application's performance in the cloud. These variables include user-controlled variables (e.g., software configuration, hardware layout/configuration) and non-user-controlled variables (e.g., virtualization parameters and server load). As an example, the performance of large-scale applications (e.g., e-commerce applications implemented as n-tier systems) becomes less predictable as node utilization increases. Because of these challenges, cloud users and service providers face a dilemma in which the low utilization of nodes leads to stable (read: more predictable) performance but higher costs. In contrast, utilizing a larger number of nodes reduces up-front costs but also leads to unpredictable performance (manifested as varying response times), which could ultimately result in penalties from missed SLAs and worse, lost business.

The effects of virtualization on response times, throughputs, and resource utilization, as well as the large number of resources provided, obfuscate predicting the performance of a virtualized service. Additional complexities come from the fact that cloud environments may combine multiple virtualization platforms that differ with regards to implementation and performance properties. The major challenges of predicting the performance of applications running in clouds are as follows:

- Applications deployed in cloud platforms lead to a complex technology stack that influences the performance of applications. To enable reasonable performance analyses, the key performance-relevant properties of virtualization environments have to be identified; for example a detailed understanding of virtualization techniques and their effects on software performance is necessary.

- Overhead associated with virtualization and the implications of resource sharing have to be taken into account. Furthermore, the resource demands and the inherent characteristics of accessing software in the cloud networks significantly differ from those in a "non-virtualized" service, leading to a difference in performance.

- Heterogeneity and the dynamic nature of applications and workload running in a cloud

introduce unpredictable resource usage and access patterns (e.g., network traffic, disk I/O).

- Cloud computing environments hide the servers and networking structure in which the application runs on. In addition, cloud providers assign virtual machines to different sets of servers even for two consecutive uses of the cloud. Even more, after deploying an application in the cloud, it may be migrated dynamically to a different server with different performance properties.

- Additional issues arise when taking the experimental measurements. For example, additional load might disturb the measurement data resulting from unpredictable noisy neighbors.

In general, performance prediction is one of the most active fields in systems' research. Consequently, many different performance prediction approaches have been proposed [68, 74, 214, 224, 225]. Those proposed approaches can be divided into two broad categorizes: analytical (model based) approaches and experimental approaches. The goal of analytical is to use modeling, simulation, or pre-existing data to predict the performance for a new configuration. In contrast, experimental approaches (e.g., Elba approach) focus on finding performance data and performance phenomena by conducting measurement studies with actual or representative applications on real hardware platforms. Furthermore, hybrid approaches have also been proposed to leverage the respective benefits of both analytical and experimental approaches [214].

For accurate performance predictions, the performance-relevant properties of cloud environments have to be reflected by the prediction models; however, current approaches provide limited or no support for the performance properties of cloud computing. For instance, properties like resource sharing (multi-tenancy), virtualization overhead (highly variable and load dependence), noise from neighbors, and fluctuations in resource utilization are largely unaccounted for in current prediction models. The prediction process needs

experts, who have detailed knowledge about virtualization techniques and their effects on software performance. In many cases, performance-relevant properties of virtualization platforms have to be retrieved by inspecting system documentation. Such documentation may not be available in all cases, may be outdated, or may not provide enough information to understand the effect specific implementation details have on performance.

The performance unpredictability problem is challenging, because many variables have a significant impact on performance based on different configurations. Practical experience and research both show that the performance of an n-tier application is a complex function of many variables, including software and hardware resource availability and tuning knobs. Since these variables are too complex for analytical models, determining the performance of an n-tier application in a cloud through large-scale experiment measurements is incredibly important. In fact, exhaustive measurements of large-scale experiments for elastic n-tier application configuration in computing clouds is motivated by three key observations. First, and perhaps least surprising, the performance of two identical software configurations can be vastly different in two different hardware platforms [174]. In fact, we have observed that a particular best practice (i.e., best performance) software configuration can become the worst-performing configuration by only changing the platform (e.g., migrating the RUBBoS application from Emulab [21] to Amazon EC2 [20]). Second, the same applications can exhibit nontrivial performance characteristics resulting from a combination of complicating factors even in a single platform [334]. Third, the performance of an application can vary unexpectedly with changes to a relatively small set of factors. Such factors may include garbage collection, network driver overhead, and context switching overhead that are often subtle and not directly controllable by users.

We believe that despite the challenges in performance unpredictability under both analytical approaches and experimental measurements, a utilization threshold exists below which the performance remains predictable. We already know that such a threshold would vary

depending on software and hardware configuration settings, so to determine this value for all the configuration settings of interest would require large-scale experimental measurements.

Running large-scale experiments to find performance and scalability characteristics in cloud environments introduces a set of new and challenging research issues. These come from following three key activities associated with the experiment measurement process:

- *Create:* preparing the experiment testbed (e.g., cloud) and deploying and configuring the application.
- *Mange:* starting the application components in the correct order, executing workloads, collecting resource monitoring and other performance data, and parsing and uploading the results to the data warehouse.
- *Analyze:* activities associated with the post-processing of measurement data under which the collected data is analyzed using various statistical and visualization techniques to understand and explain performance phenomena.

In general, the above three activities could become trivial for experimental studies with fewer nodes; however, these types of experiments may not represent actual enterprise scale deployments. Hence, the correspondent findings will not be particularly germane. In contrast, applications with a large number of nodes provide good insight into realistic application deployments, and may produce interesting performance phenomena. Unfortunately, this also introduces significant challenges for each and every phase of the experiment cycle, as described above. Experiments at large-scale introduce the following challenges:

1. **Configuration Space:** Performance measurements for enterprise applications consist of multiple, closely related scenarios, which differ by a few (preferably one) configuration parameters. In fact, two different software or hardware configurations may produce the same performance (e.g., throughput) [174, 334], thus increasing the testable configuration space. Also, components of a distributed application can

interact in complex ways, yielding a virtually infinite number of possible states. The key challenge is to maximize the amount of configurations space that can be evaluated with limited time and human resource constraints.

2. **Deployment and Runtime Dependencies:** In distributed software testing, the applications being tested should start efficiently and in a provably correct order by simultaneously enforcing serialization constraints and leveraging the distributed system's inherent parallelism. This process may be trivial for applications with a small number of nodes; in contrast, for applications with many nodes and multiple application components (e.g., multi-tier applications), the process becomes challenging mainly due to complex dependencies among various system components (e.g., database URL, load balancers).

3. **Data Processing, Storing and Analyzing**: Performance measurement of enterprise applications often results in generating massive amounts of heterogeneous data. The amount and structure of the produced data vary based on the application, deployment platform (clouds), hardware configuration, and monitoring tools and strategies. The heterogeneous nature of the data introduces many challenges for data processing, storing, and analyzing since the structure of the data may change from one experiment to another. More specifically, storing this data becomes a challenge, because finding a universal schema is practically infeasible. The dynamic nature of the data (and by extension, the underlying schema) prevents using static tools to analyze the data, hence any successful approach must have inherently flexible and extensible.

4. **Cloud Heterogeneity:** Large-scale experiment measurements in modern cloud environments introduce many new challenges; first, selecting the most appropriate cloud from many cloud offerings is a non-trivial task; second, migrating an application between two clouds is a complex, time consuming, and error-prone task; third, there are

many challenges associated with communication, coordination, synchronization, monitoring, and complete management; lastly, the dynamic nature of the cloud introduces extra complexity.

In this thesis, I argue that a promising approach for addressing the non-trivial complexities of large-scale experiment measurement is the use of an automated approach despite the fact that manual and *ad hoc* (e.g., partially scripted) approaches are being heavily used for experimental measurements in traditional data centers. Automation removes the error-prone and cumbersome involvement of human testers, reduces the burden of configuring and running large-scale experiments for distributed applications, and accelerates the process of reliable performance measurement and data analysis. More concretely, my thesis statement can be formulated as follows.

Thesis Statement: *An **automated** infrastructure which enables cloud experiments at a scale that is beyond manual execution, by systematically reducing **deployment** and **data processing** complexity of **large-scale** performance measurement studies arising from their scale and heterogeneity.*

My proposed research is to facilitate large-scale performance measurement studies (application size and number of experiments) for n-tier applications (not limited to n-tier) in computing cloud infrastructures while minimizing human involvement. More precisely, this research enables cloud experiments in modern clouds (e.g., Amazon EC2, Open Cirrus, Emulab) at a scale that is well beyond the capacity of manual experimentation. This requires implementing the core of the Elba approach for large-scale experiment management, which is shown in Figure 1. The overall process consists of multiple building blocks. Some of which have already been addressed [178, 211, 345].

- **Experiment Design** is the process of creating a set of experiments that are necessary to evaluate a given application in given target clouds.

- **Expertus** is a code generator which converts experiment design specifications into deployment, configuration, execution, data collection, and parsing scripts that are essential ingredients for automating the experiment preparation and execution process.
- **Automation** is the process of using generated scripts to automate platform preparation, application deployment and configuration, to run the experiment, and to collect and process the generated.
- **Experiment** is executing a workload for a given hardware, software configuration while collecting monitoring and other data, in fact, most of the Elba publications to date belong to this category.
- **Experstore** is a flexible data warehouse that stores and analyzes resource monitoring and performance data collected through experimentation. These data are in fact heterogeneous and vary significantly, based on the experiments and monitoring strategy employed.
- **Performance Map** is a logical view of experiment results, for example, an application's performance across different cloud platforms.
- **Online/Offline Configuration** is the process of using performance data to make configuration decisions at runtime as well as finding appropriate software settings for new configurations.

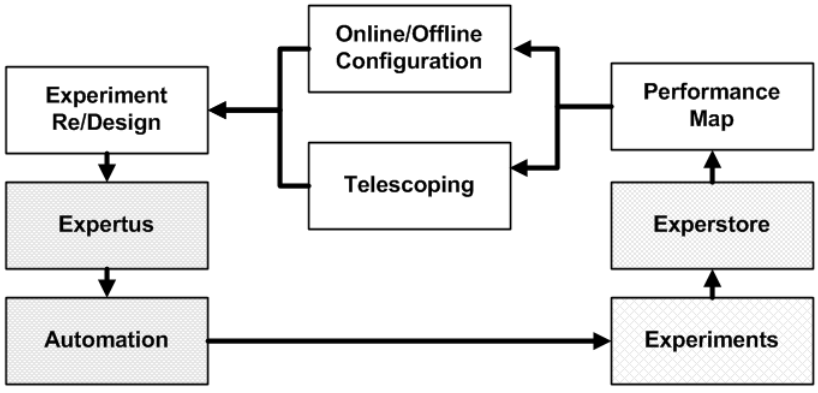


Figure 1: Our Approach to Large-scale Experiment Measurements

- **Telescoping** is the process of using collected data to drive more experiments to deeply understand observed phenomena.
- **Experiment Redesign** is the process of creating new experiments or modifying existing experiments either to validate online/offline configurations or to prove (or disprove) performance hypothesis.

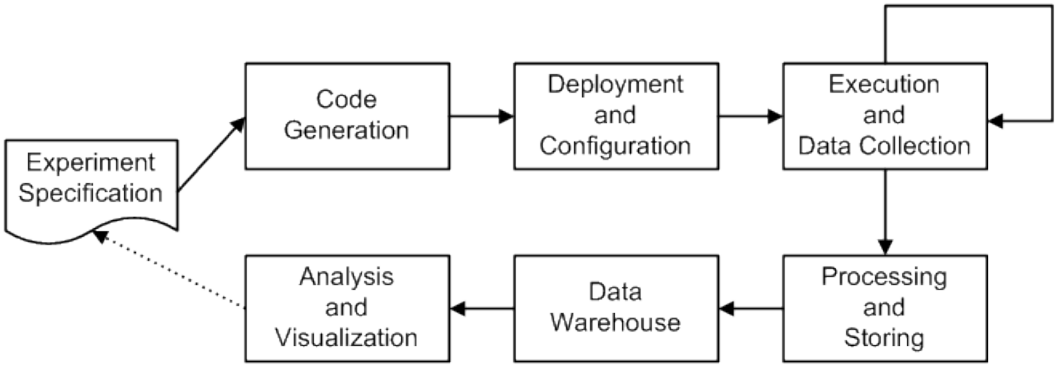


Figure 2: Typical Workflow for Experiments with Expertus

The proposed works automate the performance measurement process by systematically addressing the challenges associated with the highlighted areas in the Figure 1. We have automated the complete workflow (see Figure 2) from experiment specification to data analysis, and the framework generates necessary resources to automate complete process. The framework consists of a code generator, an experiment driver, an automated data parser, a flexible data warehouse, and a web portal for data analysis. The cornerstone of the automation is the code generator; yet, building a flexible and extensible code generator for distributed systems remains a significant research challenge. Environmental and design changes pressure the input language to change and evolve. Similarly, the generated code (output) often needs customization so it can be used for a range of software, cloud, and operating systems. This constant evolutionary pressure of input and output formats has so far limited the practical life span of code generators that have been developed for distributed system software.

My proposed thesis research can be arranged according to two main building blocks (extracted from Figure 1), which are shown in Figure 4. More specifically, my research in this area can be categorized in *an infrastructure for automating large-scale performance studies* and *demonstrating the success of the approach through large-scale experiments*.

Code generation and experiment automation [175] is the process of generating all the necessary resources to automatically manage the experiment execution process. This helps to reduce the difficulties of application configuration and dependency handling. More specifically, this enables researchers to conduct multiple different experiments in parallel with minimal involvement. My research in this area covers the process of building a flexible code generation framework that enables automation of large-scale experiment studies of n-tier applications in modern clouds. This includes generating code (scripts and other resources) for application deployment and configuration, experiment execution, data collection and parsing, storing data in the data warehouse, and data analysis. Our code generator capable of generating all the necessary resources to automate the complete process. We have a experiment driver that uses the generated resources to drive the complete experiment. We have created a adaptive data processor to support different monitoring formats, a flexible data warehouse to efficiently store and process measurements data, and a web portal to quickly and thoroughly analyze the data to find interesting performance phenomena. As unexpected failures in distributed systems (e.g., AWS outage on April 2011 [1] and December 2012 [2]) have become commonplace, we thought working on a fault-tolerant deployment at runtime [125] to make the deployment and execution failure resilient was another worthwhile exercise.

To demonstrate the success of our approach, we have used the automated infrastructure for large-scale experiment studies to explore and understand the complex system state space without rigid a priori assumptions. My research in this area includes: evaluating performance and scalability variations of different clouds, comparing the performance of different virtualization technologies, evaluating database scalability across different database

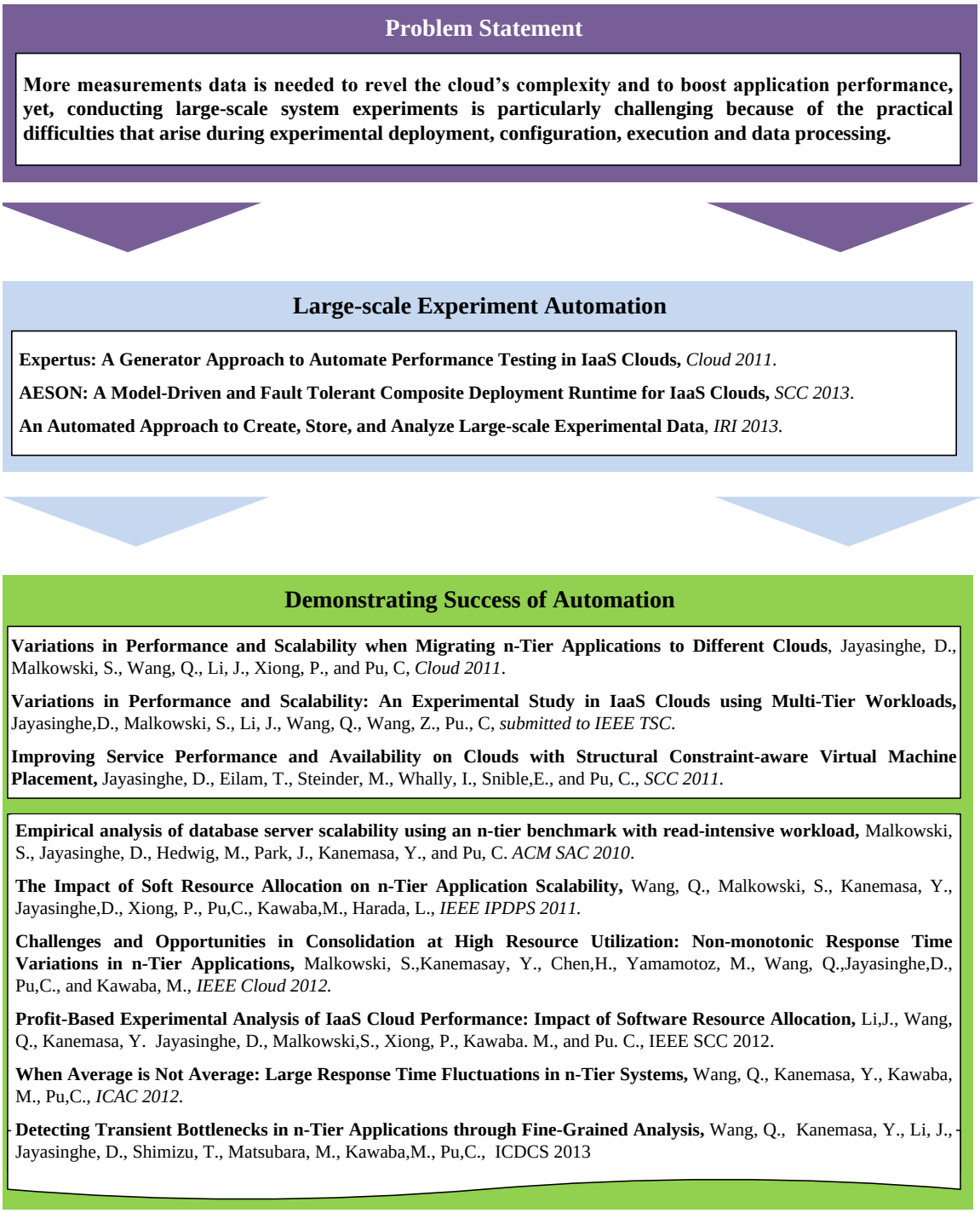


Figure 3: Overview of Core Dissertation Research in the Area of Automated Performance Management for Elastic n-tier Applications in Computing Clouds.

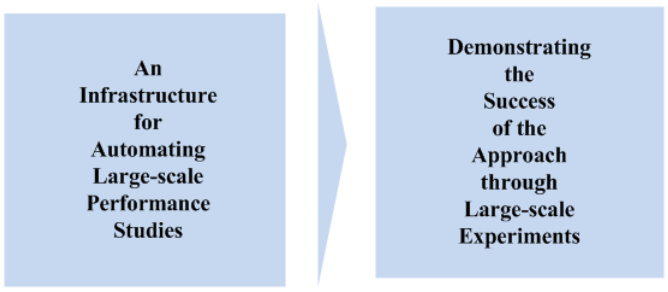


Figure 4: Research Overview: Building Blocks

technologies [174, 218], and studying the challenges and opportunities associated with consolidation at high resource utilization levels [220]. I have also investigated related topics such as comparing the similarities and differences between software and hardware resource allocation [334] and analyzing the performance and scalability of n-tier systems when they are migrated to different clouds [174]. In addition, I have also focused on ways in which we can use to improve application performance in cloud environments through performance and application semantic aware virtual machine placement [173].

The description of my core thesis research follows the aforementioned building blocks in Figure 4. Concretely, in this thesis proposal document, I have divided my work to focus on large-scale experiment automation, presented in Chapter 3, and the fault fault-tolerant approach to recovery from failure during deployment and runtime, presented in Chapter 4. The success of this approach is demonstrated in Chapter 5 (comparison of different clouds), Chapter 6 (comparison of modern virtualized technologies), and in Chapter 7, which evaluates performance and virtual machine placement. In Chapter 8, we conclude the proposal with a summary, an outlook and immediate future works.

CHAPTER II

RELATED WORK

This chapter presents an initial overview of related work. Section 2.1 discusses related work from the area of experiment automation and code generation. Section 2.2 provides related approaches for detection of non-trivial performance phenomena in elastic systems, including clouds and virtualization technologies. While these sections serve as a good starting point for contextualizing the contributions made in this dissertation, all subsequent chapters (that detail technical contributions) feature comprehensive related work sections that are dedicated to each specific technical topic.

2.1 Automating Large-Scale Experiment Measurements

Experimentation is an essential approach used in both academia and industry to gain an understanding of system behavior, formation and testing of hypotheses, system configuration and tuning, solution information formation, and performance bottleneck resolution. There are many open source and commercial tools for configuration management and small scale application deployment [7, 14, 16, 17]. Yet, researchers have undertaken relatively few efforts to build software tools to reduce complexity associated with large-scale testing of distributed applications [65, 170, 185, 259, 260, 335].

Steafan et al. [92] introduced Cloud9 –a platform for automated software testing that uses scalable parallelization of symbolic execution on clusters of commodity hardware – to help cope with path explosion. Some authors extended their work and introduced the concept of testing as a service (TaaS) [98]. The authors claimed that the combination of recent advances in test automation and the availability of compute clouds offer unprecedented levels of testing quality. DART [114], an automated regression testing framework built on top of Emulab, provided a set of primitives for writing tests for distributed systems and a

run-time mechanism to execute the tests in a fast and efficient manner. However, extending these tools to other platforms has remained a challenging problem. In our approach, we tried to develop a generic framework to facilitate large-scale experiment through automation. Compared to other existing approaches, our tool has proven to be extensible, flexible and modular by smartly combining the most commonly used technologies.

Our project parallels several others using XML and XSLT for code generation. For example, the SoftArch/MTE and Argo/MTE teams have also had favorable experiences using XML + XSLT generators to ”glue” off-the-shelf applications together [95, 154]. Apache Axis2 [296] uses an XSLT-based code generator for generating multi-language stubs and skeletons from WSDL. One of the most similar approaches to ours is Weevil [336], which also focuses on workload generation and script creation. In fact, later on, the Weevil team observed some of the limitations with their approach, and they proposed four enhancements to explore richer scenarios to obtain results with greater confidence [335]. To our knowledge, these efforts have not explored the issues of extensibility, flexibility, and modularity that are presented here, in this document.

2.2 Experimental Analysis for System Scalability and Performance in Clouds

A central problem with enterprise system studies is that scalability evaluations traditionally address only scaled load scenarios to determine the best achievable performance (e.g., [104]). Concretely, experimentation methodologies, which feature parallel scaling of load and resources, mask system overhead in realistic production system conditions, which are typically over-provisioned. Consequently, a reliable body of knowledge on the aspects of management, capacity planning, and availability is one of the limiting factors for estimating the impact of potential real world solutions [102]. Moreover, the research community broadly understands that transactional replication can exhibit unstable scale-up behavior [152]; therefore, peak performance cannot be sufficiently represented by peak throughput [102].

While the increasing popularity of cloud computing has spawned very interesting research on private and public clouds, to the best of our knowledge, no previous work has evaluated IaaS clouds using complex n-tier (also known as multi-tier) systems to find performance and scalability issues across all possible tiers. Jim et al. presented a method for achieving optimization in clouds by using performance models in the development, deployment, and operation of applications that run in the cloud [202]. They illustrated the architecture of the cloud, the services offered by the cloud to support optimization, and the methodology used by developers to enable runtime optimization of the clouds. Thomas et al. analyzed the cloud for fundamental risks that might arise from sharing physical infrastructure among mutually distrustful users, even when their actions are isolated through machine virtualization [267]. Ward et al. discussed the issues and risk associated with migrating workloads to clouds, and the authors also proposed an automated framework for facilitating a "smooth" migration to cloud [338]. In this thesis, we instead have provided experimental results related to potential scalability and performance issues after migration.

Mayur et al. evaluated Amazon S3 as a black box and formulated recommendations for integrating S3 with science applications and for designing future storage utilities targeting this class of applications [249]. They discussed how costs can be reduced by exploiting data usage and application characteristics to improve performance and more importantly, by introducing user managed collaborative caching in the system. Despite the apparent differences, this work has some important similarities to our work: first, both approaches evaluated commercial clouds; second, both papers used real applications and not simulations; third, both approaches derived recommendations to address the uncovered issues. Guohui et al. have deeply analyzed the network overhead on EC2, and they presented a quantitative study on end-to-end network performance across different EC2 instances [332]. Furthermore, Walker has presented a performance analysis and shown that a performance gap exists between performing HPC computations on a traditional scientific cluster and on an EC2 provisioned scientific clusters [331]. Similar to our findings, his experiments also revealed

the network as one of the key sources of system overhead.

As a way of showing the success of automated experiment measurement, we have extended performance analysis to show the significance of virtualization and associated network overhead. Padma et al. have also looked at both the transmission and recipient aspects of this workload. Concretely, they analyzed virtualization overhead and found that in both cases (i.e., transmit and receive) the limitations of VM scaling are due to *dom0* bottlenecks for servicing interrupts [58]. Koh et al. also analyzed the I/O overhead caused by virtualization and proposed an outsourcing approach to improve the performance [193].

Additionally, Barham et al. [70] benchmark Xen against VMware Workstation and User-Mode Linux. They show that Xen outperforms these alternatives based on a range of micro-benchmarks and system-wide tests. Clark et al. [115] repeat this performance analysis of Xen in [70] and confirm the high performance of Xen claimed in [70]. Also, they compare Xen on x86 with IBM zServer and find that the former has better performance than the latter. In an another study, Padala et al. [48] concentrate on comparing Xen's and OpenVZ's performance when multi-tiered applications are consolidated. Their experimental results show that Xen, as compared to OpenVZ, incurs higher overhead and a higher average response time increase by over 400% as the number of application instances grows from one to four. This can be explained by looking at L2 cache misses; Xen has more L2 cache misses than OpenVZ.

In another study, Deshane et al. [96] focus on three aspects Xen and KVM benchmarking: overall performance, performance isolation, and scalability. They illustrate that Xen has excellent scalability while KVM encounters substantial problems as guests crash when their hosts have more than 4 guests. KVM outperforms Xen in isolation. Using a kernel compile test , Xen has better performance than KVM while KVM outperforms Xen based on I/O-intensive tests. Camargos et al. [197] analyze the performance and scalability of six virtualization technologies (KQEMU, KVM, Linux-VServer, OpenVZ, VirtualBox and Xen) for Linux. They find Linux-Server delivers little to no overhead according to all of their

tests. In all these experiments, Xen, KVM and VirtualBox perform well. Their OpenVZ experimental result is disappointing, and they suggest that KQEUM should be avoided in production systems. The model in [197] evaluates a queuing network with fair share scheduling using trace-driven simulation. Yet, our experiences with performing large-scale studies suggest that the challenges associated with performance prediction occur; because, these existing approaches have been unable to accommodate complex system characteristics (e.g., garbage collection, context switching).

CHAPTER III

LARGE-SCALE EXPERIMENT AUTOMATION

The flexibility and scalability of computing clouds make them an attractive application migration target; yet, cloud is still a black-box for the most part. This opacity impedes running new applications in the cloud efficiently. The natural and presumably unbiased approach to reveal the cloud's complexity is to conduct more experimental studies. However, conducting large-scale system experiments is particularly challenging because of the practical difficulties that arise during experimental deployment, configuration, execution and data processing. We address these challenges through Expertus — a flexible automation framework to `create`, `manage` and `analyze` large-scale experiments [175]. Expertus combines template-driven code generation techniques with aspect-oriented programming concepts to generate necessary resources to fully automate the experiment measurement process. The system is composed of a code generator, a data warehouse, a web portal for data analysis, and a set of command line and web-based tools to provide the complete workflow for automated measurements. In Expertus, a researcher provides only the high-level description about the experiment, the framework does everything else, and at the end, the researcher receives the results in the web portal.

To date, Expertus has been used with over 100,000 different experimental scenarios (500 different hardware configurations, seven IaaS clouds, and over 10 benchmarks); in fact, our experience shows that new clouds, benchmarks, and software packages can easily be incorporated.

3.1 Introduction

One of the most reliable approaches to better understand the cloud is to collect more data through experimental studies. By using the experimental measurement data, we can understand *what has happened*, explain *why it happened*, and more importantly *predict what will happen* in the future. Yet, conducting large-scale performance measurement studies introduce many challenges due to the associated complexity of application deployment, configuration, workload execution, monitoring, data collection, data processing, and data storage and analysis. In addition, due to the nature of the experiments, each experiment produces a huge amount of heterogeneous data, exacerbating the already formidable data processing challenge. Heterogeneity comes from the use of various benchmarking applications, software packages, test platform (cloud), logging strategies, and monitoring tools and strategies. Moreover, large-scale experiments often result in failures; hence, storing incomplete or faulty data in the database is a waste of resources and may impact the performance of data processing.

As mentioned, conducting large-scale experiments in distributed platforms such as cloud environments introduces a set of new and challenging research issues, and these can be categorized into three main types: **create**, **manage** and **analyze**. Create is the process of preparing the platform, deploying and configuring the application. Manage refers to starting the application components in the correct order, executing workloads, collecting resource monitoring and other performance data and parsing and uploading the results to the data warehouse. In Analyze, we mean the process of analyzing the collected data using various statistical and visualization techniques to understand and explain performance phenomena. Nevertheless, the most challenging step of an experiment is to deploy and configure the application being tested on the target platform. According to industry analysts, IT operating expenses account for 70-80% of IT spending, and half of this is attributed to application deployment and deployment-related tasks [5].

In Expertus, we have created tools to fully automate the experiment measurement process.

In our approach, a user provides a description about the experiment (i.e., application, cloud, configurations, workloads, and etc...) through a domain specific language (or through the web portal), and then Expertus generates all of the resources that are necessary to automate the measurement process (i.e., create, manage and analyze). Methodologically, we achieve this by combining multiple automation steps. First, we have created a web-based workflow management portal to create the experiments that involves mixing and matching different application components, clouds and configurations (Alternatively, a user can manually create the experiment configuration file). Second, the code generator takes the experiment configuration file (either produced through the web portal or by other means) as an input and generates automated resources (e.g., shell scripts, property files, and etc...). Third, the experiment driver deploys and configures the application, executes workloads and monitors and collects measurement data. Fourth, an automated data extractor processes the measurement data and uploads the data to a warehouse. In fact, these data extractors come with the web portal to help the user to filter data during data extraction. Fifth, for each experiment, a fresh set of tables is created on-the-fly, and our data extraction tools use information describing the data format to parse the files. Finally, we have automated the data analysis, which consists of a web-based graphical user interface as well as integration scripts for statistical packages like *R* and *Matlab*.

In large-scale experiments, activities associated with data introduce many challenges. To address the data processing and parsing challenges, we have used ETL (extract, transform, and load) tools and approaches [73, 327] to build a generic parser (*Expertract*) to process the performance measurement data. Our parser can process more than 98% of the most commonly used file formats in our experimental domain (e.g., dstat logs, sar logs, Apache logs, Tomcat logs). To address the storage challenge, we have designed a special data warehouse called *Experstore* to store the corresponding performance measurement data. The tables are created and populated on-the-fly based on the experimental data. This is possible, because our data warehouse is fully dynamic. More specifically, at the end of

each experiment, we create a set of tables to store the data, and the underlying schema is solely based on the structure of the data (e.g., how many columns and tables). Finally, to address the challenges associated with navigating and analyzing an enormous amount of performance measurement data, we have built a web portal, which helps users to: navigate the data warehouse, visualize the data, statistically analyze the data, and identify interesting performance phenomena from it.

Expertus makes significant strides towards realizing flexible and scalable benchmarking for today's complex cloud environments. For example, we have used Expertus in over 500 different hardware configurations (i.e., varying node number and type) and over 100,000 different software configurations (i.e., varying software and software settings). Our experiments have used over 100,000 computing nodes in various cloud environments such as Emulab [21], Amazon EC2 [20], Open Cirrus [24], and Wipro [29] using different benchmarks such as RUBBoS [25], RUBiS [26], and Cloudstone [294]. As more computing and storage cloud offerings become available, we expect the number and variety of experiments to increase by another order of magnitude with our current Expertus version.

Expertus is very efficient when one needs to carry out many different experiment scenarios. From one experiment to another, users can quickly change configuration settings, platform settings, software settings and cloud platform deployment settings with minimal effort. For example, by changing only one line (i.e., `<param name="platform" value="EC2"/>`) and changing the IP address of Emulab's experiment configuration, new scripts are automatically generated to run the same Emulab experiment on EC2. Similarly, automated scripts can be generated for a new RUBiS experiment from a preexisting RuBBoS experiment configuration with less than 20 line modifications.

Expertus is designed for flexibility and extensibility. For example, a developer can extend Expertus to support new clouds, new benchmarks and new software packages with only a few template changes. For example, by changing only 8.21% of the lines in a template, we were able to support Amazon EC2 once we had support for the Emulab testbed. This

8.21% caused a 25.35% change in the generated code for an application scenario with 18 nodes. Similarly, by changing 9.33% of a template, we were able to support the Open Cirrus cloud, and these changes resulted in 26.91% of the generated code changing. To switch from the RUBBoS to the RUBiS benchmark, the template only changed 5.66%.

Expertus automates and optimizes experiment scheduling in addition to collecting high quality and consistent experimental data across multiple trials of the same experiment (i.e. sequences of actions in exactly the same order). Scientific experimentation requires the ability to produce and reproduce high quality data. Repeatability requires resetting the experiment after each run to the same initial state and then re-deploying and re-configuring the application. For example, the deployment and configuration of the RUBBoS n-tier benchmark (with 24 nodes) on Amazon EC2 needs roughly 216 computing-node-hours when using manual approaches, because all of the nodes have to be up and running while the others are configured. In contrast, Expertus reduces the required deployment and configuration time to 48 computing-node-hours through increased parallelism, resulting in savings of about 80%.

The remainder of this chapter is structured as follows. In Section 3.2, we define some of the terms used in the chapter and discuss associated challenges in experimentation. We present the automated experiment management infrastructure in Section 3.3. The core of the automated infrastructure i.e., the code generator is discussed in Section 3.4, and automated performance measurement data generation is presented in Section 3.5. Section 3.6 presents the challenges of data and our solution to store the data. Processing and parsing the data is presented in Section 3.7, and the data visualization approach is discussed in Section 3.8. Finally, we provide a discussion of related state of the art approaches in Section 3.10, and we summarize the chapter in Section 3.11.

3.2 Background

The performance unpredictability problem is challenging, because many variables can significantly impact performance depending on specific configuration parameters. Practical experience and research both show that the performance of an n-tier application is a complex function of many variables, including software and hardware resource availability and tuning knobs. In this section, we first formally define experiment, hardware and software configuration (section 3.2.1), and detail some of the challenges in experimentation due to complex dependences, heterogeneity and scalability (section 3.2.2).

3.2.1 Terms and Definitions

Experiment is a comprehensive, controlled analysis of a system (e.g., RUBBoS on Amazon EC2) with single or multiple objectives. Objectives may include finding the resource bottlenecks, finding appropriate values for tuning knobs (e.g., size of a thread pool), or finding the maximum sustainable workload.

Hardware configuration is a set of connected nodes (either physical or virtual), where each node is represented with a set of resources $\sum R_i$, (e.g., CPU speed, memory, architecture, OS, virtualized, # cores).

We say that two hardware configurations are different if: 1) the number of nodes in the two configurations differs; 2) two configurations with the same number of nodes, but at least one node has a different R_i value.

Software configuration is the combination of application components, their types and configurable parameters (e.g., thread pool size, database pool size, connection timeout, # concurrent client requests).

We say two software configurations are different when: 1) at least one configuration parameter differs from one configuration to another; 2) use different software types (e.g., MySQL vs. Oracle); or 3) different application components (e.g., 2-tier vs. 3-tier).

3.2.2 Distributed Testing and Challenges

Measurement challenges: Performance testing for enterprise applications consists of multiple closely related scenarios by varying a few configurable (preferably one) parameters at a time. As an example, to find the maximum achievable throughput for an e-commerce application on EC2 may consist of multiple experiments where the difference between any two would be the number of concurrent client workloads (e.g., 1000 users vs. 2000 users). In fact, two different software or hardware configurations may produce the same throughput [174, 334], thus increasing the testable configuration space. Also, components of a distributed application can interact in complex ways, yielding a virtually infinite number of possible states. In reality, evaluating all possible configurations is practically infeasible for a vast majority of systems. Thus, the key challenge is to maximize the amount of configurations that can be tested with limited time and human resource constraints.

Application challenges: In distributed software testing the applications should start efficiently and in a provably correct order by simultaneously enforcing serialization constraints and leveraging the distributed system's inherent parallelism. This process may be trivial for applications with a few nodes; however, for applications with many nodes and multiple application components (e.g., multi-tier applications), the process becomes challenging mainly due to complex dependencies (e.g., database URL, load balancers). For example, a web application with Apache as the load balancer and Tomcat as the application server requires modification to the Apache configuration whenever application servers are added or removed. Similarly, changing the DBMS causes the application server configuration parameters (e.g., JDBC, URL) to change.

Data challenges: Measuring the performance of enterprise applications often results in massive amounts of heterogeneous data being generated. The amount and structure of the produced data varies based on the application, deployment platform (clouds), hardware configuration, and monitoring tools and strategies. The heterogeneous nature of the data introduces many challenges for data processing, storing, and analyzing, since its structure

may change from one experiment to another. More specifically, storing becomes a challenge, because finding a universal schema is practically infeasible. The dynamic nature of the database schema inhibits the use of static tools to analyze the data. Hence, a successful approach to storage must provide flexibility and extensibility to accommodate the dynamic nature of the data.

Cloud challenges: Testing software systems in today's cloud environments introduces many new challenges. First, selecting the most appropriate cloud from many cloud offerings is a non-trivial task. Second, migrating an application between two clouds is a complex, time consuming and error prone task. Third, communication, coordination, synchronization, and monitoring only add to testing's general management challenges. Lastly, the dynamic nature of the cloud introduces extra complexity. For example in a traditional data center, the IP address of a given server is assumed to be constant, indefinitely; therefore, once an application is deployed with pre-configured IP addresses then that configuration can be used for multiple test cases. Conversely, in computing clouds, whenever we rent new computing instances, they come with a fresh set of IP addresses. So, if we want to run multiple test cases, we would need to modify the configuration scripts after each test case to account for the new IPs, which is a non-trivial task for large applications.

3.3 Automated Experiment Management Infrastructure

Experiment measurement is a tedious process that consists of multiple activities; each of which can be accomplished by using different tools and approaches. A typical experiment consists of the following four activities: *design*, *create*, *manage* and *analyze*.

- *Design:* creating a set of experiments that are necessary to evaluate a given application in given target platform, which can be further classified into: initial design and redesign (change the design based on observed results).

- *Create:* preparing the experiment testbed (i.e., cloud) and deploying and configuring the application.

- *Manage:* starting the application components in the correct order, executing workloads, collecting resource monitoring and other performance data, and parsing and uploading the results into the data warehouse.

- *Analyze:* activities associated with analyzing the collected measurement data using various statistical and visualization techniques to understand and explain performance phenomena.

We have designed our automated experiment measurement infrastructure to fully automate the four activities listed above. In fact, some of these activities encapsulate multiple smaller activities. A high level view of our approach, which details these activities and some of the tools used in the process, appears in Figure 1. As shown in the figure, the process consists of eight activities, a brief description of these follows:

- *Experiment Design* is the process of creating a set of experiments that are necessary to evaluate a given application in given target clouds.

- *Expertus* is a code generator, which converts experiment design specifications into deployment, configuration, execution, data collection, and parsing scripts that are essential ingredients to automate the experiment preparation and execution processes.

- *Automation* is the process of using generated scripts to automate platform preparation, application deployment and configuration, experimental execution, data collection, and data processing.

- *Experiments* is the actual execution of workloads and data collection, in fact, most of the Elba publications belong to this category.

- *Experstore* is a flexible data warehouse that stores and analyzes resource monitoring and performance data collected through experiment measurements. These data are in fact heterogeneous and vary significantly depending on the experiments and monitoring strategy in use.

- *Performance Map* is a logical view of experiment results, for example, an application's performance across different clouds.

- *Online/Offline Configuration* is the process of using performance data to make configuration decisions at runtime as well as finding appropriate software settings for new configurations.

- *Telescoping* is the process of using collected data to drive more experiments to deeply understand observed phenomena.

- *Experiment Redesign* is the process of creating new experiments or modifying existing experiments either to validate online/offline configurations or to prove (or disprove) performance hypotheses.

We designed the automation framework by combining multiple modules and built-in flexibility to accommodate new modules. We employed modular architecture, because of its distinct advantage of enabling us to change one component without affecting other components. Our framework is designed to support both command line (CLI) users and graphical users. We accomplish this by providing a SOAP interface for each module. The different modules in the system are illustrated in Figure 5, and a brief description of each is given below:

1. *Code generator:* is the heart of the automation where it generates all the necessary resources to automate the experiment management process. In a nutshell, code generator takes experiment configuration files as the input and generates all the required files (e.g., scripts and other resources).

2. *Expertus Service:* Automation framework consists of multiple components; some of which are connected using SOAP and REST APIs. We created an Axis2 [296] based Web service that supports APIs for code generation, data extraction, status update, and information listing. We used the code generation API to create a command line

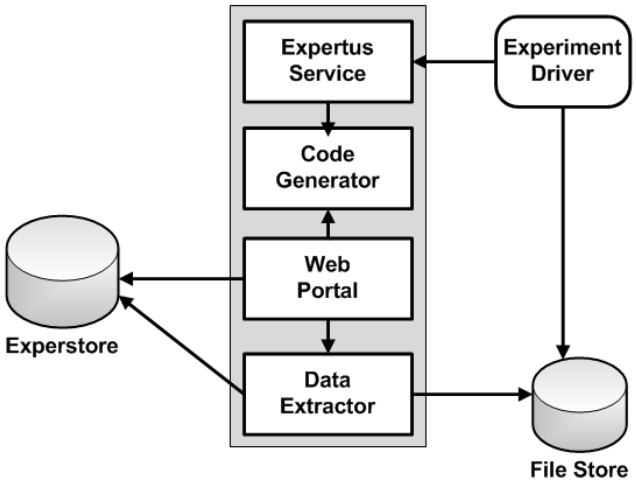


Figure 5: A High-level view of Automated Infrastructure

tool (CMI) for code generation. A user can pass an experiment configuration file and execute the code generation script, which calls the remote web service using the SOAP API to generate and download the code. Similarly, during experiment execution, the experiment driver reports a given experiment's status using the REST API. The Web service is designed to store all the information in the database. The REST API provides an easy way for any application to update the status without worrying about the database

3. *Experiment Driver:* We use a centralized approach for experiment execution, and the component called, the experiment driver, is responsible for this task. Code generator generates all the scripts, and a special script, called `run.sh`, maintains the sequence for script execution. Experiment driver uses `run.sh` to find the order of execution. It connects to all the nodes through SSH/SCP, and it executes the scripts on the corresponding nodes. In addition, experiment driver is configured to collect and report information about the user, time, workload start time and end time, and the platform to the Expertus service through the REST API.

4. *Data Extraction:* Each experiment produces gigabytes of heterogeneous data for resource monitors (e.g., CPU, Memory, thread pool usage, and etc...), response

time and throughput, and application logs. The structure and amount of collected data vary based on system architecture (`64-bits` vs `32-bit`, `2-core` vs. `4-core`), monitoring strategy and monitoring tools (e.g., `sar`, `iostat`, `dstat`, `oprofile`), logging strategy (e.g., Apache access logs), and number of nodes and workloads. Data extractor is written to help users easily export experiment data to the data warehouse.

5. *Filestore:* At the end of each experiment, experiment driver uploads experiment data to a file server to store the data in raw format. Some data analysis tools and approaches need to have access to the original data files, so these raw data files need to be retained. In addition, there are temporal files and error logs files, which we do not want to put into the database. Data extractor runs on the file store to export data from the file store into the data warehouse.

6. *Experstore:* Is the flexible, extensible and dynamic data warehouse we have created specifically to store heterogeneous experiment data collected through our experiments. However, our approach is general enough to use for other applications as well.

7. *Web Portal:* Is the user friendly web interface we have developed to aid in experiment creation, code generation, data extraction and data analysis. Our web portal is composed of multiple web applications to support a rich user experience and an efficient way to analyze the collected data.

3.3.1 Experiment Automation Process

Here we provide a brief overview of experiment automation with Expertus. The automation process is illustrated in Figure 6, and brief description about each item is given below:

1. Create experiment specification for the application, software packages, cloud and experiments. This includes complex system dependencies and semantics of the application (components, # of nodes, parameters), cloud (e.g., URLs, user name,

password, key-pair, node types), software (e.g., `thread pool`, `maxClients`), and experiments (e.g., type, load configurations).

2. Use Expertus and generate scripts. To start the experiments, the user remotely connects via (`SSH`) to the control node and executes the `run.sh` script (also known as the main script). The `Main` script uses other scripts and drives the complete automation, which in fact consists of multiple stages.

3. Platform configuration sets up the target cloud. This may include starting and configuring computing instances, mounting the file systems, creating users, setting user permissions and installing the operating system libraries.

4. Application deployment is for deploying the target application on the configured cloud (e.g., downloading/copying the installation packages).

5. Enterprise systems are inherently complex due to component dependencies, so application configuration is needed to correctly configure the systems.

6. `run.sh` script runs the test plan (6), which in fact consists of multiple iterations. In order to produce consistent and reproducible results, all of the dirty data from previous test cases needs to be removed before starting a new one.

7. Upload the resource monitoring and performance data to the data warehouse (i.e., Experstore).

3.4 Expertus - Code Generator

Expertus is a flexible and extensible code generation framework, which is designed for generating resources that are needed to automate large-scale experiments on distributed systems. We created Expertus by extending the Clearwater approach [299], which we originally developed for generating code for infopipes. The framework can be easily and quickly extended to support new clouds, benchmark applications and software packages. It

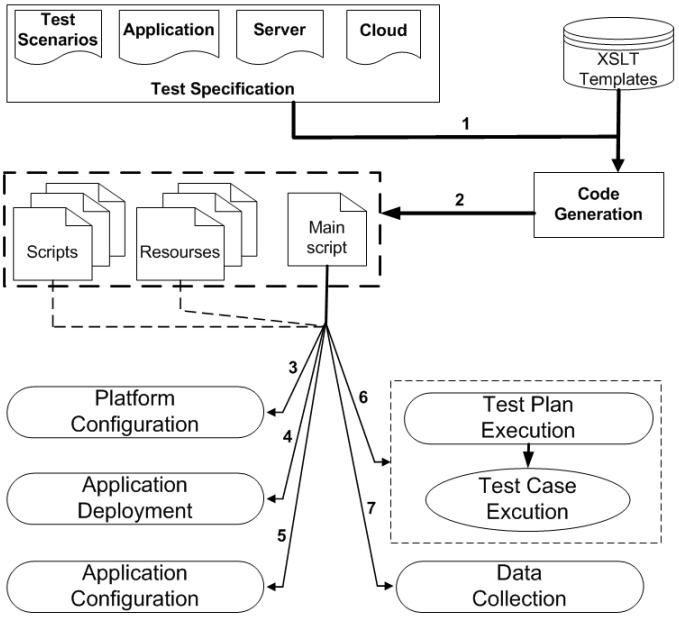


Figure 6: Automated Testing Process with Expertus

achieves this flexibility and extensibility by leveraging the advantages of XML (e.g., extensibility, flexibility) and XSLT-based (eXtensible Stylesheet Language (XSL) Transformation) transformations (e.g., extensibility, flexibility, modularity).

Building a code generator to automate large-scale experiments for n-tier applications is a significant research challenge. Environmental and design changes pressure the input language to change and evolve. Similarly, the generated code (output) often needs customization for a range of software, clouds, and operating systems. This constant evolutionary pressure of input and output formats has so far, limited the practical life span of code generators developed for distributed system software. More precisely, a code generator at this scale should address the following three challenges:

- **Abstraction mapping**: Code generator should translate from high-level design specifications such as XML into a general purpose language and communication layer appropriate to each target participating in the system (e.g., shell scripts).
- **Heterogeneity**: Because of the heterogeneity in the system, a practical solution must accommodate inputs from multiple specification regimes and outputs to multiple target platforms.

- **Flexible Customization**: Provide a mechanism whereby each application developer can augment scripts created by the code generator and introduces his own application-specific properties irrespective of the target platform.

Expertus is architected to operate on XML and XSLT. Use of XML helps us to mitigate the abstract mapping issues in which new tags and elements can be easily added to the XML document without affecting the existing business logic. This also helps us to design a human friendly XML-based specification that can easily adopt to new requirements and changes. Use of XSL helps us to transform a single specification into multiple different output formats by simply changing the XSL template(s). In addition, making changes to an XSL template is much easier than changing the application source code (and then compiling and rebuilding). We use aspect oriented programming (AOP) [188] concepts to provide flexible customization. We allow a user to add `pointcuts` to the template as well as provide aspects `advices` at different levels (e.g., application, cloud, operating system and user).

The code generator adopts multi-pass and template-driven generation techniques. To generate the code, we identify the parts of the application components (source and scripts) which are repetitive and dynamic (or parameterizable), next we create XSL template by specifying those parameters, and finally generate the code through XSL transformation by providing values for those parameters. From an architectural viewpoint, our code generator adopts a compiler approach of multiple, serial transformative stages - a code generation pipeline. The hallmarks of code generator are stages that typically operate on an XML document, which is the intermediate representation, and XSLT actually performs the code generation. The code generator uses multiple transformation stages and finally generates source code for the file system. (Readers who are interested in learning more about our code generator can refer to Clearwater [299]).

Expertus and the experiment specification language are designed to support the following four key entities during the experiment deployment, configuration and execution.

- *Artifacts* includes software packages, servers, nodes, and clouds, for example `httpd.tar.gz`

- *Constraints* are used to provide the precondition for deployment, for example, Apache Tomcat needs `JDK 1.6` or above, or an experiment needs to be executed on a `64-bit` operating system

- *Dependencies* specify the relationship between different entities, for example the database URL for the application server, or JDBC library.

- *Deployment and Start order* are used to make certain we execute the actions in the right order, for example, before configuring the application server, we need to deploy it. Similarly, in a 3-tier system, we need to make sure we start the database server before the application server.

Fully automating the experiment management process requires having all the resources (e.g., scripts, source code, and property file) necessary for deploying, configuring and executing the experiment. To handle the complexity of this process, Expertus divides it into four distinct stages and generates scripts for each stage:

1. Platform configuration: setting up the target testbed (e.g., Emulab, EC2) - starting instances (e.g., in Amazon EC2), setting up the file systems (e.g., in Emulab), creating users and groups.

2. Application configuration: deploying and configuring application components - software packages (e.g., Tomcat), benchmark (e.g., RUBBoS), workload generators, resource monitors (e.g., dstat).

3. Runtime configuration: resetting the environment from one trial of an experiment to another - removing logs, removing temporary files, restoring the database.

4. Execution and monitoring: starting application components, workload generators and resource monitors.

3.4.1 Flexibility and Extensibility through XML and XSLT

Expertus is designed to operate on top of XML and XSLT. The use of XML provides the code generator with a high degree of extensibility. This stems from XML's simple, well-defined syntax requirement and its ability to accept arbitrary new tags, thereby bypassing the overhead encountered when managing both XSLT templates and AOP. For example, a template can add an arbitrary element to the intermediate XML; however, unless the processing code is written to process the new tag, the newly added tag remains untouched.

XSL transformation is the process of converting an XML document into another document using XSL. Typically, XSLT converts an XML document into another XML document (e.g., HTML) or any other type of document. Expertus consists of two types of templates, namely `Base templates` and `Aspect templates`. The former is used to generate application/platform independent parts of a resource, and the latter is used to modify (weave) the generated resource for the target application/platform (e.g., Emulab vs. EC2).

3.4.2 Multi-Stage Code Generation

The code generator adopts a compiler-based approach through its implementation of multiple serial transformation stages. The intuition for this type of approach is straightforward-by dividing the large problem into smaller pieces and processing them one at a time, better extensibility and flexibility can be provided. For example, it processes the application-related transformations at one stage and cloud-related transformations at another stage. As shown in Figure 7, at each stage, Expertus takes an XML document and produces another XML document through XSLT transformation. Expertus treats the first and last stage differently compared to the rest of the workflow. During the first stage, the process takes the experiment specification as the input, and in the final stage, it generates automation resources specific to the file system (instead of an intermediate XML).

At each stage, Expertus uses the intermediate XML document created from the previous stage as the input to the current stage. The current stage uses the intermediate XML

file to retrieve the names of the templates that are needed and then generates another intermediate XML document, which creates another XML document (Transformed XML in Figure 8). If needed, AOP `pointcuts` is added to the intermediate XML during the transformation phase. Consequently, Aspect Weaver is used to weave such `pointcuts`. Aspect Weaver processes the `pointcuts` through `Aspect templates` and creates the woven XML (Figure 8). Finally, the woven file is written to the file system through the file writer. More precisely, the automation scripts are written to the file system during the final stage of the pipeline. At each of the other stages in the worflow, an intermediate XML is generated, and the next stage in the pipeline is called. The complete process for a single stage is shown in Figure 8.

Figure 9 illustrates how code generation works in practice, and this figure shows code snippets from an XSLT template, an intermediate XML and a generated script. The highlighted lines in the XSLT template indicate the parameter names that it should use from the XML document to create the output. The XML file is highlighted with the corresponding parameter values, and those parameters can either come from user inputs (specification) or generated from a previous stage by another XSLT template. Finally, the generated script is highlighted with parameter values. As illustrated in the figure, the XSLT template has extracted the values from intermediate XML, and it has created the output XML file. We use this simple logic to generate resources to automate complex test scenarios.

One of the key challenges in experiment automation is to handle the variance and continuous evolution of applications, software packages, operating systems, and clouds. For example, in Emulab, a user can communicate between two nodes by typing `SSH node2`; in EC2, an equivalent command is `SSH -i gsg-keypair node2`. Similarly, for older Apache versions, `mod_jk` has to be placed in `jk/native2`; however, newer versions require it to be in `jk/`. Likewise, when doing performance testing, crosscutting features like monitoring and logging are important factors, and they might need to change from one experiment to another. To handle variances and crosscutting features, Expertus itself has to

be modular. Expertus achieves this modularization by leveraging AOP techniques.

The `Resource templates` are created by identifying output variances (e.g., SSH communication) and crosscutting features. Each identified variance is embedded into `Resource templates` as `pointcuts` (namespace qualified XML elements). `Aspect templates` are created by embedding the `advices` (code that is run at the joint points), which are needed to support the `pointcuts`. During the transformation phase, `Resource templates` add `joint points` (well defined points in the execution flow) to the intermediate XML, and during the weaving phase, `Aspect templates` transform intermediate XML into woven XML by applying correct `advices`. If new `pointcuts` are needed, we can easily modify the `Resource templates` and add (or change) the `Aspect templates` to include the corresponding logic to process the new crosscutting features.

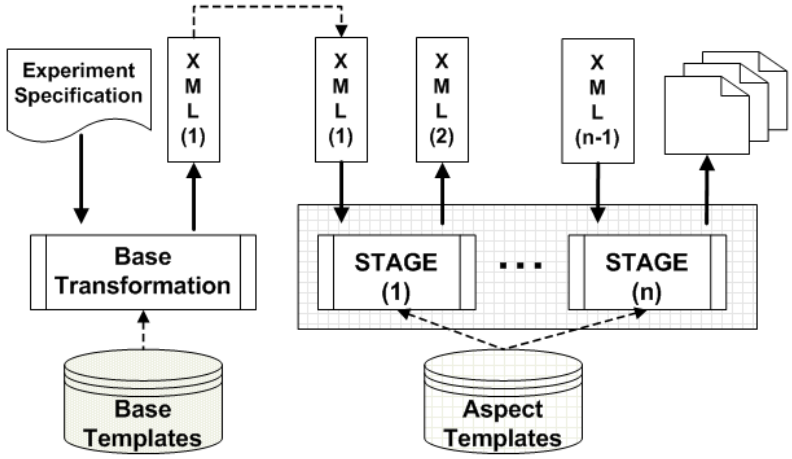


Figure 7: Multi-Stage Code Generation Process

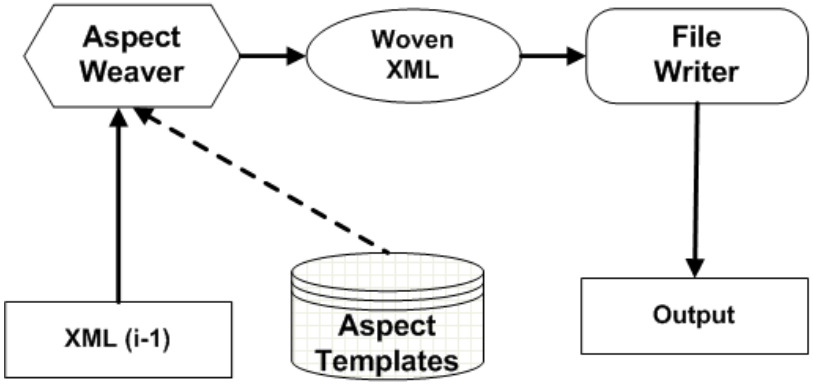


Figure 8: Transformations Steps within a Single Stage

A simple aspect weaving scenario is illustrated in Figure 10, where a cloud-specific aspect template is used to generate a script to have different SSH communication for EC2 and Emulab. In the figure, the XSLT template is highlighted with the `pointcuts`. The intermediate XML is highlighted to illustrate the `joint points`, and the woven code with applied advice is highlighted for EC2 and Emulab.

A concrete example for the complete code generation process with Expertus is shown Figure 11 and Figure 12. As shown in Figure 11 (or Figure 12), the first task is to fine what

XSLT Template

```
.....
cd <xsl:value-of select="//params/env/param[@name='OUTPUT_HOME']/@value"/>
source set_elba_env.sh
echo "  INSTALLING RUBBOS on $HOSTNAME"

mkdir -p <xsl:value-of
select="//params/env/param[@name='RUBBOS_TOP']/@value"/>
# install RUBBoS
tar xzf <xsl:value-of
select="//params/env/param[@name='SOFTWARE_HOME']/@value"/>/<xsl:value-of
select="//params/env/param[@name='RUBBOS_TARBALL']/@value"/> --
directory <xsl:value-of
select="//params/env/param[@name='RUBBOS_TOP']/@value"/>

.....
make
sudo make install
echo "  DONE INSTALLING RUBBOS on $HOSTNAME"
```

Intermediate XML

```
<xtbl name="Rubbos" version="0.1">
 <instances><params><env>
  <param name="OUTPUT_HOME" value="/opt/rubbos/output"/>
  <param name="SOFTWARE_HOME" value="/opt/softwares"/>
  <param name="RUBBOS_TOP" value="/mnt/rubbos"/>
  <param name="RUBBOS_TARBALL " value="RUBBoS-servlets.tar.gz"/>
  <param name="MYSQL_PORT" value="3313"/>
  .....
<env><params>
<tomcat-conf>
  <param name="maxThreads" value="330"/>
  .....
</tomcat-conf>

<instance name="BENCHMARK" type="benchmark_server">
   .....
</instance>
```

Generated Script

```
.....
cd /opt/rubbos/output
source set_elba_env.sh
echo "  INSTALLING RUBBOS on $HOSTNAME"

mkdir -p /mnt/rubbos
# install RUBBoS
tar xzf /opt/softwares/RUBBoS-servlets.tar.gz --directory=/mnt/rubbos
.....
make
sudo make install
echo "  DONE INSTALLING RUBBOS on $HOSTNAME"
```

Figure 9: Generating Output/Intermediate XML with XSLT Transformation

user needs to generate, for the scenario shown in figure, Bob needs to generate a shell script. As highlighted in the figure, he needs to execute `make install` with `sudo` privileges, in contrast Alice does not need to have `sudo` keyword to execute the same command.

1. Create the XSL template by parameterizing the target resource that user wants to generate. As an illustration, the corresponding XSL template for Bob's and Alice's case is shown in the figure. As shown, the template is created with adding <`axpect id="addsudo"`/> before the `make install` command. This represents the aspect which helps to achieve Bob's and Alice's requirements.

2. To generate the code, user need to provide the values for parameter shown in the template using the experiment configuration file.

3. At the code generation, it combines the template and the experiment configuration file to create the intermediate XML file.

4. Next, it goes through aspect weaving process. As shown in Figure 11, Bob's template has implemented `sudo` for the aspect `addsudo` whereas Alice's has not provided anything for her aspect.

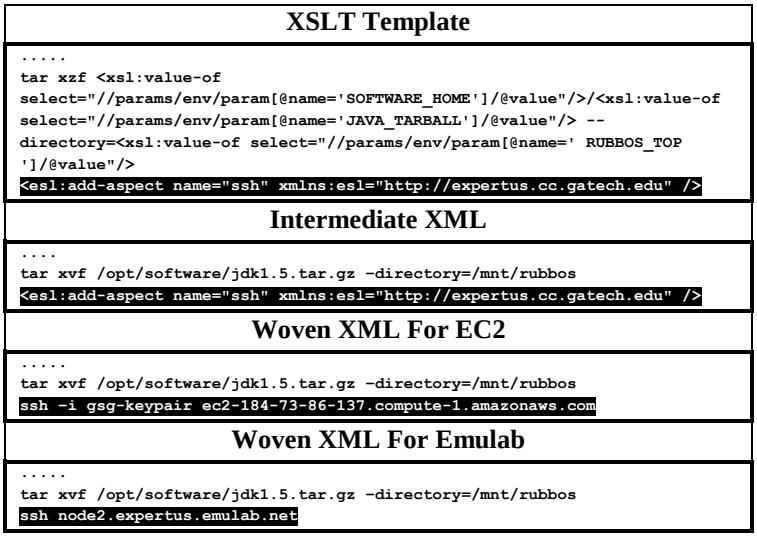


Figure 10: Handling Cloud Variances through Aspect weaving

5. When the intermediate XML is transformed thorough the Aspect templates, those templates create the correct output meet user's requirements.

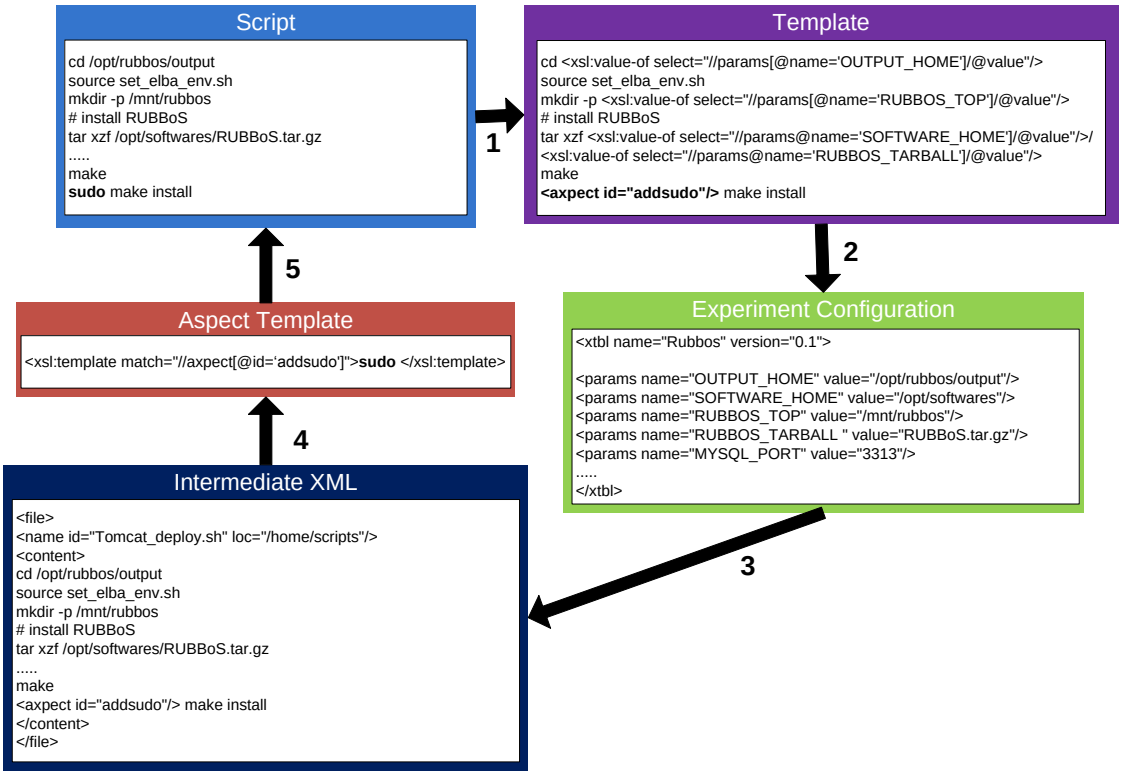


Figure 11: Code Generation with Aspect (an Example for Bob's case)

3.5 Automated Generation of Heterogeneous Experimental Data

In our approach to large-scale experimental measurements, we deploy actual or representative applications (e.g., benchmarks like RUBBoS [25], RUBiS [26], and Cloudstone [293]) on actual or representative deployment platforms (e.g., Amazon EC2 [20]) and execute workloads. Through the large-scale experiments, we produce a huge amount of heterogeneous performance data. The heterogeneous nature of the data arises from the nature of the applications, clouds, monitoring tools, and monitoring strategies. We conduct large-scale experiments and collect data by fully automating the process, and as explained before, the code generator generates all the necessary resources to automate the process.

Once the system is deployed, we execute the workload against the deployed system. In this step, we run the planned experiments according to the availability of hardware resources. For example, we usually run the experiments by increasing the workload. For each workload, we run the easily scalable (browse only) scenario first followed by read/write scenarios. To minimize cache inter-dependencies across experiments, after each batch of experiments, we finish the data collection, ramp-down the system, stop all of the servers, and start the next batch with sufficient ramp-up time. The iterations continue until all the experiments are finished.

During the experiment execution, the automated infrastructure collects information about system resources (e.g., CPU, memory), application specific data (e.g., thread pool usage), application logs (e.g., Apache logs), high level data like throughput and response time, and any other data that the user wants to collect. This process continues for each and

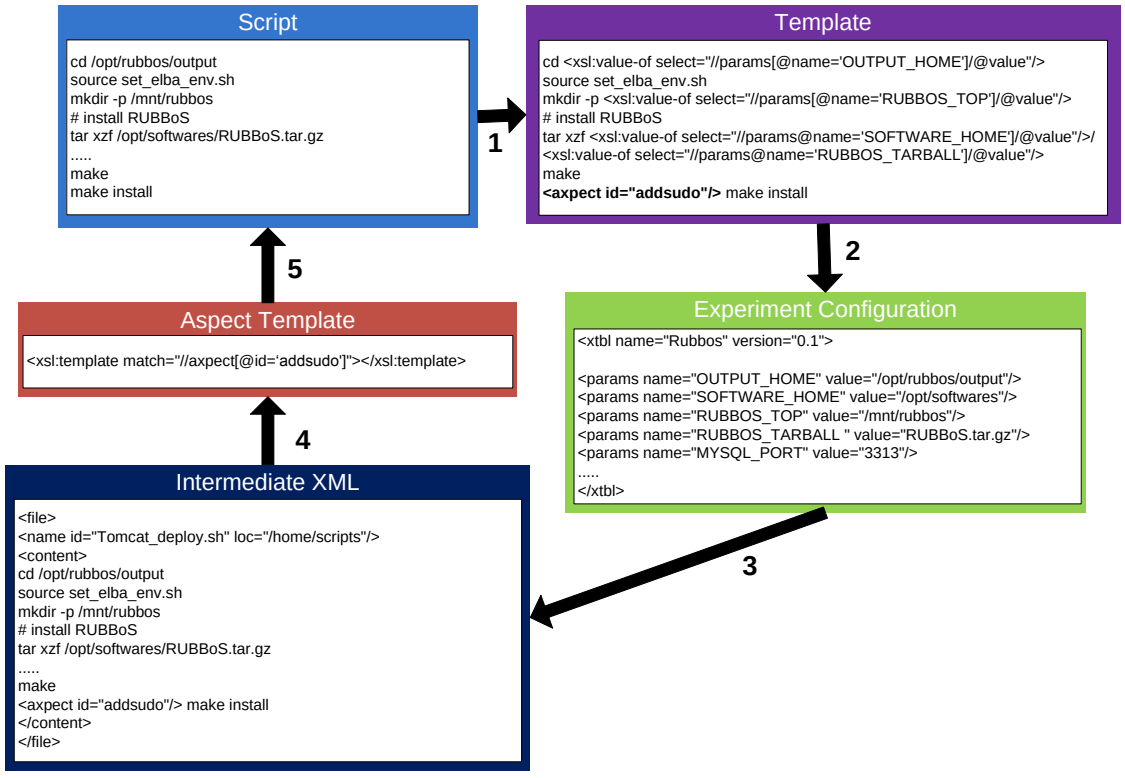


Figure 12: Code Generation with Aspect (an Example for Alice's case)

every workload. In fact, experiments in our domain consist of 50 to 60 workloads, and each workload runs for approximately 30 minutes. The framework is capable of collecting, managing and storing data without any help from the user. The data extractor, as the name implies, extracts this myriad data and stores it in the data warehouse after the experiment is completed.

3.6 Experstore - A Flexible Data Warehouse

The performance and scalability measurement of enterprise applications is a tedious process. In most cases, researchers are initially unaware of what resources need to be monitored, whether it be high-level data (e.g., response time or throughputs) or low-level resource monitoring data (e.g., CPU, memory, cache miss, application logs). Monitoring all the potential resources is not necessary the best approach, because of the performance overhead. As such, a researcher (or a performance engineer) typically starts with a selected set of resources and gradually changes the monitoring set based on the observed results. For example, if the issue is caused by CPU, then the researcher might look into additional data like context switches or interrupts. Moreover, an experiment can be deployed on heterogeneous platforms (e.g., XEN vs. KVM, 2-core vs. 4-core), thus the structure of monitoring data may differ from one experiment to another. Frequently, the user changes the monitoring strategy, which results in new data formats, not to mention new monitoring data. The user may also choose to monitor new resources (possibly using new monitoring tools) based on the observed results. As a consequence of all of these issues, the experimental data cannot be feasibly stored on a set of static tables.

In addition, large-scale experiments often result in failures; hence, storing incomplete or faulty data in the database is a waste of resources and may impact the performance of data processing. Most OLAP (Online Analytical Processing) applications such as experimental data analysis require joining multiple tables or performing self-joins. When the tables are huge, processing becomes very time consuming, and the processing time increases

significantly unless the tables can be loaded into main memory.

We address these problem through Experstore —a special data warehouse solution to store performance measurement data. Our data warehouse is fully dynamic; tables are created and populated on-the-fly based on the experimental data. More specifically, at the end of each experiment, we create a set of tables to store the data, and the schema is based solely on the structure of the data (e.g., how many columns and tables). Tables names are created dynamically by combining the experiment ID and the timestamp, and the names of the tables are stored in a mapping table called 'Resource Mapping Table'. In this approach, if an experiment fails, we can simply drop all the tables corresponding to the failed experiment. Since a new set of tables are created for each experiment, data processing efficiency increases dramatically, because the small tables (corresponding to each experiment) can be easily joined in memory.

Data to Table Mapping: As mentioned before, the tables are created dynamically, based on the structure of the data. In fact, we achieve this by creating an intermediate representation of the data, where we describe for each resource monitoring file how to process the file and which parser to use. One file might contain data for more than one resource type, for example, CPU, memory and network I/O. For each resource type, we create a 'profile', which maps a file's structure to an applicable schema. As an example, the profile identifies which columns of a CSV file correspond to a particular database table. Next, we have a mapping, where we specify which profiles are applicable to a given node. A mapping contains node name, file name and corresponding profile. A sample profile and a mapping file is shown below:

Listing 3.1: Code Listening for Profile and Mapping

```
<profile>
    <separator>,</separator>
    <resource-name>CPU0</resource-name>
    <processor-class>dataimport.filter.CSVFileProcessor
```

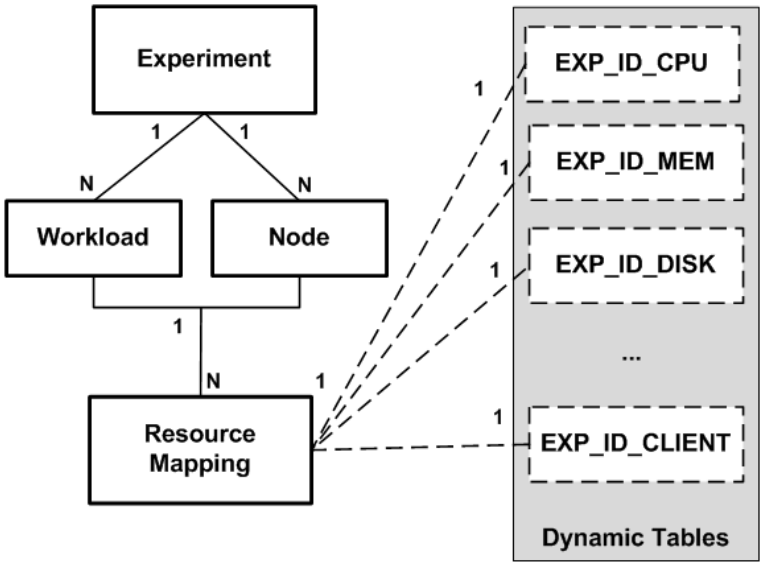


Figure 13: Experstore - Static and Dynamic Tables and their Relationships (tables which are marked as dynamic tables are created on-the-fly, and the remaining are static tables to store experiment meta-data)

```
    </processor-class>
    <column index=''0'' colname=''user'' datatype=''double''/>
    <column index=''1'' colname=''system'' datatype=''double''/>
    <start-index>10</start-index>
    <end-index>0</end-index>
</profile>

<mapping nodename=''Apache'' filname=''169.254.100.3.csv''
                startwith=''false'' endwith=''false''
                profiles=''CPU0,DISK,CPU1,NETWORK,SYSTEM''/>
```

The structure of the data warehouse is shown in Figure 13. As shown in the figure, it consists of four tables to store experiment meta-data (e.g., experiment name, platforms, node and workload information), which are typically fixed across experiments. The highlighted tables are the tables that are created on-the-fly, as shown in the figure. 'Resource Mapping

Table' (also known as Dictionary) stores the names of the dynamically created tables along with the resource names. For example, it has a record for CPU utilization for experiment ID (EXP_ID), and the value is EXP_ID_CPU. Likewise, all the monitoring data for a given experiment is stored. In fact, it has a record for each unique node, workload, and resource.

3.7 Expertract - Automated Data Extractor

The large-scale experiment measurements with Expertus generate an enormous amount of metadata in the form of log files, i.e., structured and semi-structured text files. This data needs to be extracted and stored in the environmental data warehouse for later analysis. The primary challenge with this activity is the fact that different tools and software packages produce the data of interest, differently. Thus, the goal of an automated data extractor is to build a generalized parsing approach for experimental data to support both the known and unknown data formats. To begin developing an approach, we explored the more recent and foundational research in the Extract, Transform and Load (ETL) domain. Next, we built a system bound by the existing data files known in the environment, and more specifically, we focused on parsing 'fixed width' flat files. (Based on our research, this problem reduces to other challenging formats like delimited-based files.)

Generally speaking, the log files that comprise our ETL domain have significant variability. The following categories are just a few of the points of difference:

- Structure - semi-structured [flat files, delimited files, HTML] and structured [XML].
- Data Record Structure - what arrangement or construction within the file represents one data record and how do the data fields relate - either explicitly or implicitly - ontologically.
- Data Type - numeric, string and other ASCII characters.
- Data Variability - how consistent the data is within a specific - either explicit or implicit - data field and among the fields within a file.

- Data Validity - similar to Data Variability but specifically related to identifying error conditions within a specific - either explicit or implicit - data field.

These observations suggest perhaps an alternative view of the original problem. That is any log file of interest contains data and an inherent presentation. This presentation is a mixture of inherent data ontology and human readability factors. Any successful approach must disambiguate these two concerns. Specifically, an approach needs to be able to handle three aspects of any given log file:

- Data - this concerns aspects of data quality and validation.
- Ontology - that is the logical relationship among the data elements in the file.
- Presentation / Layout - this concerns how the data is expressed in the file.

During the course of our system design, we concentrated on four primary monitoring file data patterns, and these cover more than 98% of the monitoring data collected through our experiments. These patterns are:

1. One header: The most basic case, a given file has one header. This header could contain a row describing records, but at a minimum it must contain a row describing fields.
2. Multiple headers with sequentially corresponding data: This pattern is basically (1) except that another header appears later in the file.
3. Multiple headers with non-sequential corresponding data: This pattern differs from (2) in that data later in the file matches the first header in the file at some random position later in the file.
4. Multiple headers appear randomly in the file and data is entirely non-sequential, i.e., randomly distributed throughout the file.

We developed the tool to be user interactive, so a domain expert can help the tool to correctly interpret the data format. At the end of this process, we build an ontology for the file, and then we use the created ontology to process the file during data extraction. We leverage this for unknown or unseen file formats, and we use existing ontologies to describe the file format for the known files. The actual header-to-row-of-data matching algorithm that we delivered followed a Greedy algorithmic approach–detailed below. We augmented this core matching functionality with the following surrounding functionality:

- Simple command line user interface to capture user instructions for parsing header rows.
- Object-oriented design that supports separating file contents from file format and layout.
- *Row-Encoding Algorithm*: Only prompts the user when the system "thinks" the row is a header row. Headers have two distinct characteristics: more alphabetic and special characters relative to the total length of the string. "Noise" rows, i.e., rows that should be ignored for later processing, have one of two properties but not both, which differentiates them from header rows.
    1. Uses length-weighted character frequency to do first pass encoding.
    2. Then checks for the prevalence of special characters.
- *Header-to-Data Row Matching Algorithm*, this algorithm leverages multiple heuristics to achieve its objective:
    - Generate a byte-array representation of the string.
    - Compute character frequencies and scale `weights` based on character frequency. For example, if a `tab` appears once in a string, this character receives significantly more weight than `spaces` that occur in over half of the string, for example. Alphanumeric characters receive `0` weight.

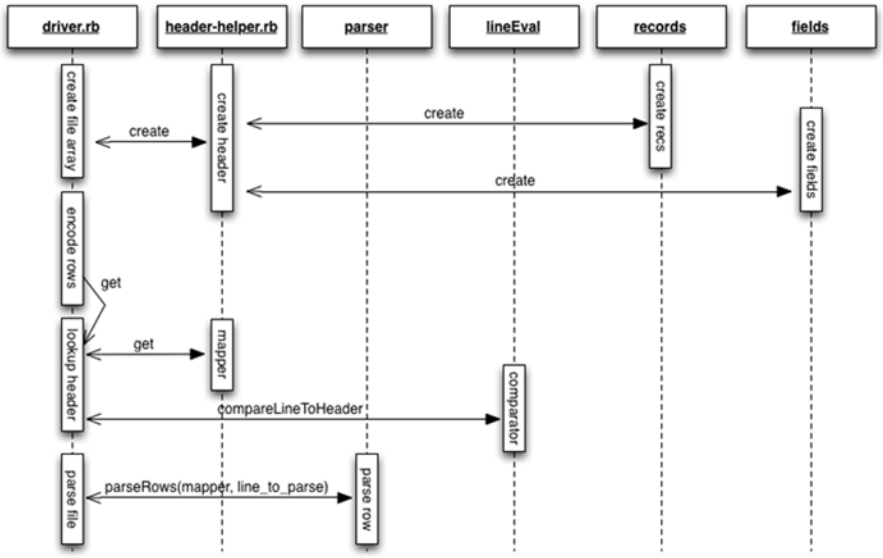


Figure 14: Experstore - Sequence Diagram for Automated Data Extractor

– If more than one header appears in a document, do a byte-wise comparison of the header row to the row of data of interest. Whichever header-row-of-data comparison results in the lowest absolute difference is the header selected to process the row-of-data.

The sequence diagram shown in Figure 14 represents the primary instruction flow for the parser. It shows the flow of events for: initially loading a data structure to hold the file contents; encoding rows of the data structure to perform matching later on; and finally, matching rows that are classified as a row-of-data to the corresponding header.

3.8 Experlyis - Web Portal for Data Analysis

Experlyis is an attempt on our part to aid in the analysis of a large amount of data collected from sundry sources. It equips the user with the capability to identify patterns, trends and relations, which generally gets obscured by the massive quantities of data. Also, it equips the user with capabilities to control the way in which data is represented, which further enhances his productivity and analytical capabilities. The application provides a simple interface that facilitates the comparison of data from seemingly unrelated sources, thereby

enhancing the user's ability to see and detect potentially interesting, latent relations that were previously unknown. As an illustration, two examples for graphs generated with Experstore are shown in Figure 15.

We believe the application can be really helpful for any system where components have dependencies, hence impact each other. Moreover, the high degree of customization for both graphs and data makes the application highly unique. For example, let's assume we have a process that when it is scheduled, it is mostly followed by processes, which require heavy disk activity. Such trends are very hard to identify, but our application makes this trivial to pinpoint.

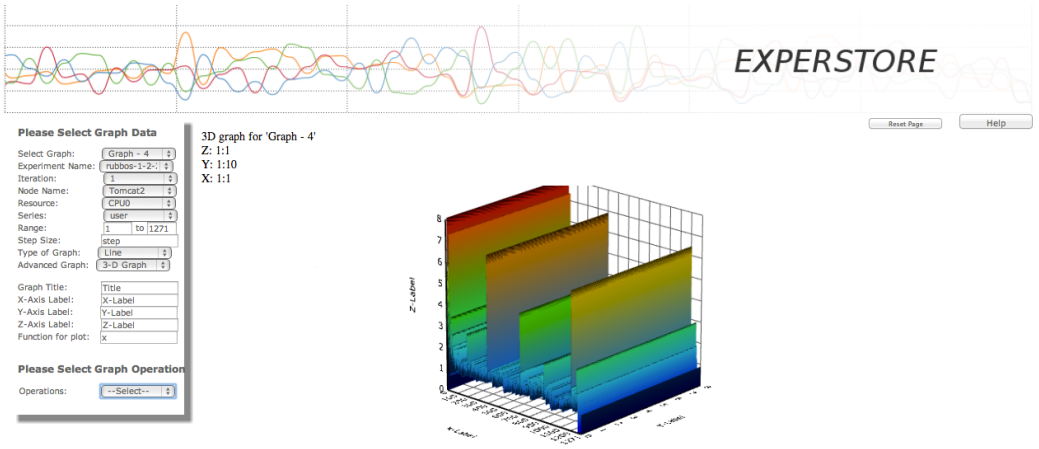
We have tried to keep the design of the application very simple and clean. The core of the application is a `Node.js` server that runs as a webserver. The whole application is written using `Node.js` libraries. On the interface end, the application uses the `HighCharts` framework to plot 2-dimensional graphs and 3D `WebGL-surface-plot` for 3-dimensional graphs. The server and the front-end communicate using jQuery and JSON calls. From the server, we connect to the MySQL database using the Node.js module, which acts as a connector. The high level architecture is shown in Figure 16.

In order to develop this application, we have used the following technologies:

- *Highcharts* is a charting library written in pure JavaScript, which offered us an easy way of adding interactive charts to our application. This was our major graph generation framework and was central to our work. The library was chosen, because it has decent capabilities to handle interactive and dynamic graph generation, which happen to be the prime requirements for our application.

- *WebGL-surface-plot* library is a 3D surface plotting tool in JavaScript that uses WebGL. The 3D surface plotting library itself allows one to plot 3D surfaces, and WebGL takes advantage of hardware accelerated graphics to provide smooth interaction and visuals such as shading and lighting. The need for a secondary library arose to incorporate 3-dimensional graphs into the application, as Highcharts lacks support for 3D charts

(a) # 2D Graphs



(b) 3D Graphs

Figure 15: Experstore - Web Portal for Data Analysis

- *Node.js* is a software system designed for writing highly scalable internet applications, notably web servers. We selected Node.js as the backend for our application, as the web server in our case was quite simple. Moreover, Node.js uses event driven, asynchronous I/O to minimize overhead and maximize scalability, which were desirable qualities.

Approach and Features: Throughout the development, we focused on delivering as many customization capabilities in the graphs as possible. Adhering to this principle helped us to greatly enhance the utility of the application. Factors such as the variety of graph types, the variance of graphing scales, and changing the values associated with graph ranges were

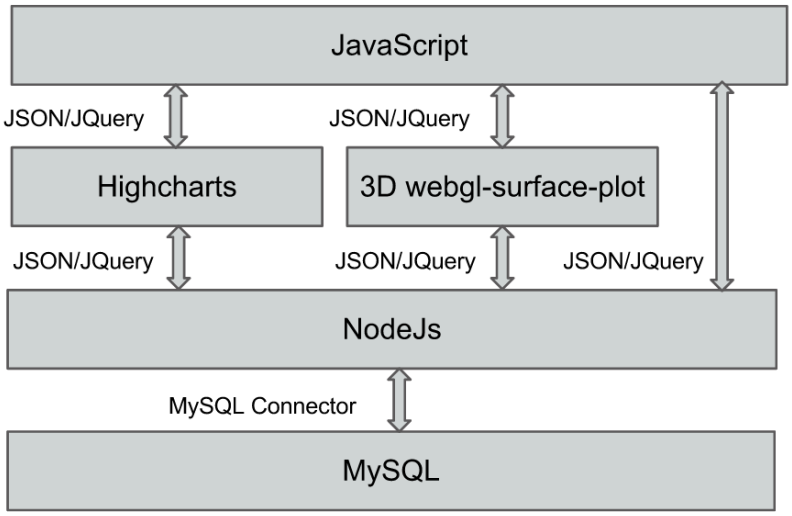


Figure 16: Architecture - Data Analysis Web Portal

some of the major challenges that we faced. In order to achieve the desired results, we experimented with different approaches for handling the data.

3.8.1 Fixed vs. Customizable Data Values

Initially, we designed the application to directly utilize the data that we obtained from the database for graph generation. But later, we realized that this makes comparison among values with significant range differences extremely difficult. Imagine comparing something that varies in thousands to something that has variations in the range of hundreds. We alleviated this condition by allowing mathematical transformations on values before they were plotted on graphs. Using the previous example, we could now multiply the values on the second series by a value, 10 in this case, making direct comparison with the first series possible.

3.8.2 Multiple Graphs vs. Multiple Series

One of the prime concerns while designing the application was supporting the comparison among multiple datasets (many series in the same graph?). This needed to be weighed against the amount of information the user would need (how many graphs?). We decided later to pursue a hybrid approach by allowing the user to have multiple graphs, each of

which could have multiple series.

3.8.3 Result Migration

Another prime concern while designing the application was facilitating the sharing of analytical data among users, so collaborative efforts could prove fruitful. In order to deliver this feasibly, we had to choose between building printing or export capabilities. Once again, we adopted a hybrid approach, and we facilitated both printing and export capabilities. The application can currently export graphs as a PNG image, a JPEG image, a PDF document or an SVG vector image.

3.8.4 Advanced Graphs

In order to enhance the capabilities of the application, we modularized and integrated advanced features like frequency distribution graphs, bucketed graphs, dynamic graphs, correlation and cumulative graphs and 3-dimensional graphs. This helped us model the application in such a way that future enhancements could be incorporated easily.

3.8.5 Interactivity

Another point of concern was user interactivity, as most of the utility of the application depended on this characteristic. We took the approach of providing intuitive interaction capabilities to the user. Some of these capabilities included: auto-customizable axes, mouse drag- based zooming, multiple graph types, graph modification, series addition, series or graph-based deletion, automated range generation for selected datasets, etc…

3.9 Evaluation

We have been using Expertus extensively and actively to perform a large number of experiments with various configurations. Through these, we have thoroughly tested our framework and improved it to have better usability, extensibility and flexibility. In this section, we demonstrate our experiences of using Expertus for experiment automation by varying the

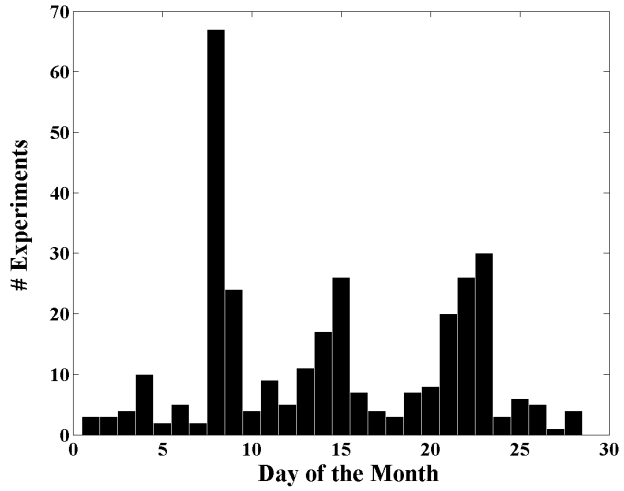


Figure 17: Number of Experiment Generated During April-2013

support for new clouds, new applications and new software packages.

3.9.1 Active Use of the Tool

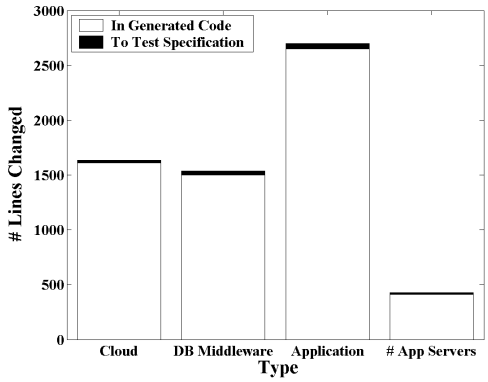
Expertus is actively being used for large experimental studies. As an illustration, we have collected data from a number of experiments during April 2013. This is illustrated in 17, and as shown in the figure, we run on average, over 10 parallel experiments daily. In some instances, we run over 50 concurrent experiments.

3.9.2 Usability of the Tool

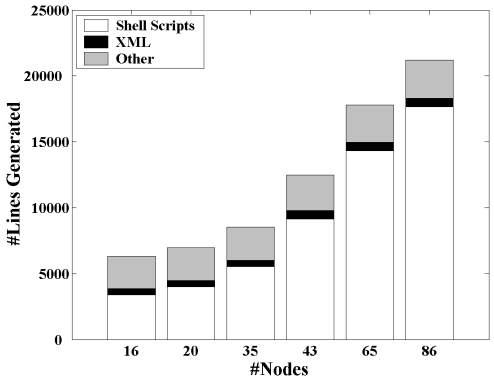
Here, we consider how quickly a user can change an existing specification to run the same experiment with different settings (e.g., MySQL Cluster [23] vs. C-JDBC [105]), the same experiment on different clouds (e.g., Emulab vs. EC2), the same experiment with different numbers of nodes (e.g., two vs. four app servers), or an entirely different set of applications (e.g., RUBBoS vs. Cloudstone).

For our analysis, we created a specification (say `a.xml`) to run the RUBBoS application on Emulab with a total of 16 nodes and generated automated resources through Expertus. We then changed `a.xml` to generate automated scripts for EC2, which required only a single line change (i.e., <`param name="platform" value="EC2"/`>) in `a.xml` and a

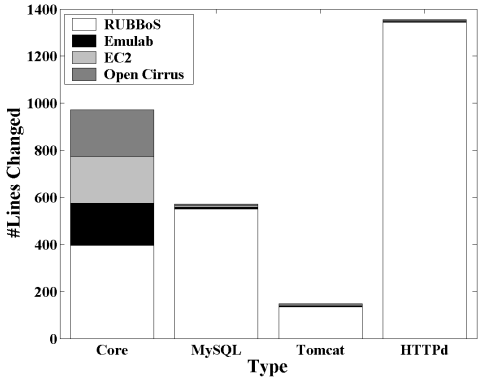
modification to the IP address. Even though the number of lines changed in the configuration file is small, the changes to the generated code are non-trivial and significantly larger. Our results are illustrated in Figure 18(a). We followed the same procedure and modified `a.xml` to change the database middleware from C-JDBC to MySQL Cluster. This change required only 36 lines (mostly MySQL Cluster specific settings), but, as shown in Figure 18(a) the differences in generated code were substantial. Similarly, with 4 lines changes to `a.xml`, we were able to move from 2 to 8 Application servers. Furthermore, with only 52 template line changes, we were able to extend the support from RUBBoS to Cloudstone.



(a) # Changes: Specification vs. Generated



(b) #Nodes vs. # Lines generated

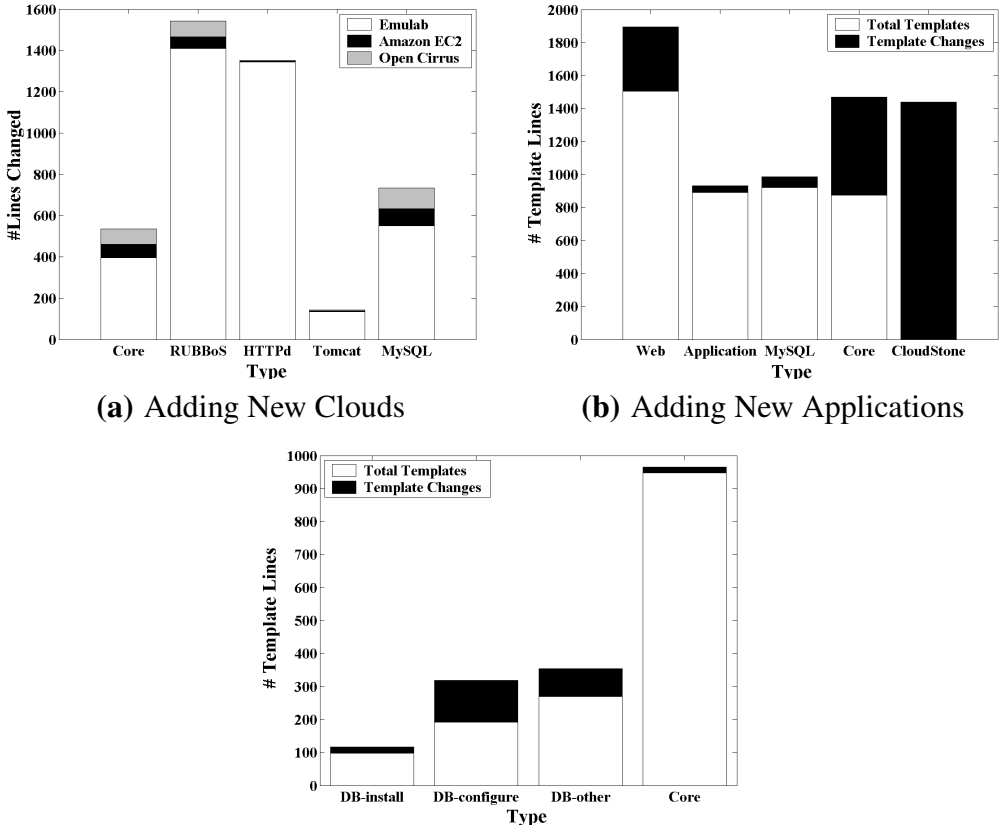


(c) RUBBoS To RUBiS

Figure 18: (a) # Lines changed in Experiment Specification vs. Generated Code, when Changing Cloud, Database, Application and # APP Servers; (b) Magnitude of Generated Code when Increasing the # Nodes in the Experiment; (c) #Lines (template) Changed for Adding a new Application(RUBBoS to RUBiS)

3.9.3 Generated Script Types and Magnitude

The biggest advantage of our approach becomes visible when automating experiments for complex applications. The quantity of resources being generated depends on the application (e.g., RUBBoS, RuBiS), software packages (e.g., Tomcat, JBOSS), deployment platform (e.g., Emulab, EC2), number of experiments, number of servers, and number of configuration parameters. To show the significance of the generated code, six different hardware configurations (on Emulab) were selected, and we counted the number of lines generated for each configuration. Our results are shown in Figure 18(b); as shown in the figure, when the number of nodes increases, the size of the generated code grows significant. The magnitude of generated code implies two factors: first, it shows the effectiveness of our approach,



(a) Adding New Clouds **(b)** Adding New Applications

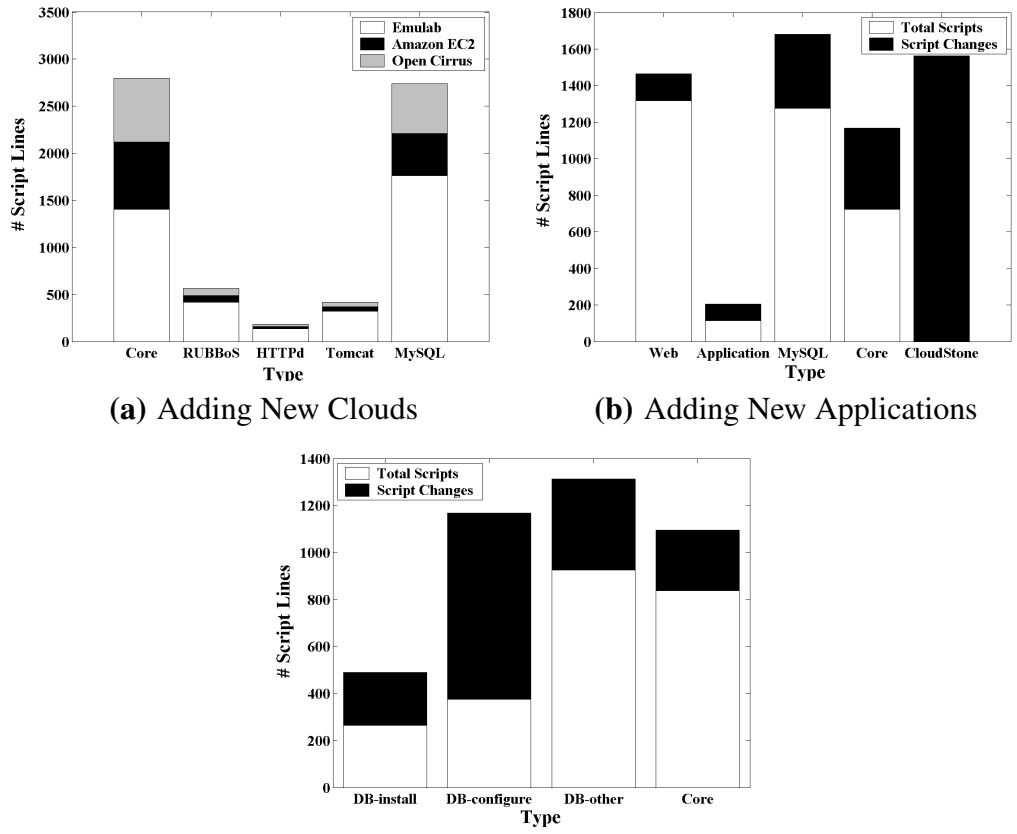
(c) Adding New Database

Figure 19: Template Line Changes: (a) Moving from Emulab to EC2 and Open Cirrus (figure shows total # lines for Emulab); (b) Supporting CloudStone from RUBBoS; (c) Moving from C-JDBC to MySQL cluster

and second, it shows the challenges that accompany manual approaches. For example, to perform an experiment with 43 nodes requires around 15K lines of shell scripts and writing all of those manually is a non-trivial task.

3.9.4 Richness of the Tool

Richness is considered in two dimensions: 1) magnitude of completed experiments; 2) the number of different software packages, clouds, and applications it supports. Expertus has been used over three years to conduct a large number of experiments spanning five clouds (Emulab, EC2, Open Cirrus, Wipro, and Elba), three applications (RUBBoS, RUBiS, and Cloudstone), five database management systems (C-JDBC, MySQL Cluster, MySQL, PostgreSQL, Oracle), various resource monitoring tools (dstat, sar, vmstat), and different



(a) Adding New Clouds **(b)** Adding New Applications

(c) Adding New Database

Figure 20: Changes to Generated Code: (a) Emulab to EC2 and OpenCirrus; (b) RUBBoS and CloudStone; (c) C-JDBC and MySQL Cluster

types and numbers of nodes. A high level summary of supported software packages, platforms, and applications are shown in Table 1 with the number of template lines indicated for each item.

Table 1: Supported Software packages, Clouds and Applications

	Platform					Application		
Type	EC2	Emulab	Open Cirrus	Elba	Wipro	RUBBoS	RUBiS	Cloudstone
Apache HTTPd	√	√	√	√	√	√	√	√
Tomcat	√	√	√	√	√	√	√	√
MySQL	√	√	-	√	-	√	√	√
MySQL Cluster	√	√	√	-	√	√	-	√
C-JDBC	√	√	-	√	-	√	√	-
Postgres	-	√	-	√	-	√	√	-
Sequoia	-	√	-	√	-	√	-	-
PHP	√	√	√	√	√	-	-	√
JOnAS	-	√	-	-	-	-	√	-
Core	√	√	√	√	√	√	√	√

Table 2 provides a high level summary (only the results we used for our publications) of different experiments performed using RUBBoS, RUBiS, and Cloudstone applications. An experiment refers to an execution of a particular workload against a combination of hardware and software configurations. In the table, nodes refer to the total number of machines that we had used during our experiments. We calculated the number of nodes by multiplying the number of experiments by the number of nodes for each experiment. Configurations mean the number of different software and hardware configurations that were studied. Experiments-per-week gives an idea of the average number of experiments executed in one week. Typically, an experiment's execution takes one hour, which is the aggregate duration of: reset time, start time, sleeping time, ramp-up time, running time, ramp-down time, stop time, and data copy time. As an example in Eumlab we, conducted approximately 132 hours of experiments per week. Finally, human effort refers to the actual human time

spent on experiment configuration (i.e., creating or modifying experiment configuration file). As presented in the table, our approach significantly reduced the human effort required to explore the configuration space quickly.

Table 2: Number of Test Scenarios Performed with Expertus

Type	Emulab	EC2	Open Cirrus	Elba	Wipro
Experiments	9215	1436	480	3567	120
Nodes	102687	25848	4987	9863	430
Configurations	392	86	28	163	8
Experiments Per Week	132	34	120	77	14
Human Effort (in minutes)	87	110	44	47	51

3.9.5 Extensibility and Flexibility

3.9.5.1 Supporting New Clouds

Our approach is fully realized when trying to support new clouds; we can use existing templates and only implement the absolutely necessary pieces, which is usually a quick and easy process. Traditionally, using manual approaches to support new clouds meant writing a set of scripts to deploy, configure, execute the test cases, and finally collect the data, which can be a cumbersome and tedious task. Here, we show the extensibility of Expertus by counting the number of changes we made to support the RUBBoS application on EC2 and Open Cirrus, once the tool had already supported RUBBoS on Emulab. Not surprisingly, we observed many differences among the three clouds that make migrating applications from one to another a non-trivial task.

To get a good understanding of how easy it is to extend Expertus to support new clouds and the significance of the generated code, we show our own experiences on migrating the RUBBoS application from Emulab to EC2 and then to OpenCirrus. Initially, we developed XSLT templates for the Emulab cloud and then used the same templates to extend the applications to new clouds. To begin, we calculated the total number of template lines for the Emulab configuration, and then we counted the total number of template lines that were modified (i.e., added, removed, and updated) to support the target cloud environment. Our

results are shown in Figure 19(a). As illustrated in the figure, once we have implemented templates for Emulab, the amount of change required to support EC2 was very small (less than 10% of the template needed to change). To better understand these differences, we have divided the changes into five categories: 1) Changes to core templates (cloud independent resources), 2) Web server (e.g., Apache HTTPd), 3) Application server, 4) Database server and, 5) RUBBoS specific templates. As shown in the figure, we made very few changes to HTTPd and Apache Tomcat; in contrast, we changed 10% of the total lines in the core templates, which was mainly due to the differences in EC2. Likewise, moving from Emulab to Open Cirrus involved a similar number of change, and our results are shown in the same figure (Figure 19(a)).

To show the significance of the template changes on generated code, we created an experiment specification (say `a.xml`) with 16 nodes and generated code for Emulab. Next, we modified this base configuration for EC2, generated the code, and calculated the differences between the two sets of code. To highlight these differences, we divided the generated code into the same five categories as before. Our results are illustrated in Figure 20(a). As shown in the figure, a small change in the template makes a huge difference in the generated code. Likewise, we modified the `a.xml` to support Open Cirrus, generated the code, and compared this set of generated code. As shown in Figure 20(a), the difference between Emulab and Open Cirrus is similar to that of EC2.

3.9.5.2 Supporting New Applications

Here we show the number of template changes required to support two different applications, (RUBiS and Cloudstone) once we have support for RUBBoS. Both RUBBoS and RUBiS follow the same application architecture and deployment procedure whereas Cloudstone requires a slightly different deployment topology.

Supporting RUBiS necessitated making the following modifications to the templates: 1) changed the web server load balancing URL, 2) built the RUBiS web application (in

lieu of RUBBoS), 3) specified the dataset location, and 4) changed the workload executor. Supporting RUBiS required creating three new templates for the client side. The number of changes to the template to support three different clouds (Emulab, EC2 and Open Cirrus) is shown in Figure 18(c). As shown in the figure, we made only a 30% change to the core template (the original RUBBoS template) to integrate RuBiS into Expertus.

Supporting Cloudstone required more changes compared to that of RUBiS. In Cloudstone, the Web server became the key entity whereas in RUBBoS/RUBiS, this functioned solely as a load balancer. Importantly, we were able to reuse most of the templates for the application and database tier. Figure 19(b) provides detail on the number of template lines we changed for this task. To facilitate analysis, we divided the template changes into five categories to highlight the aforementioned changes. Due to the major difference between RUBBoS and CloudStone, we had to create a set of new CloudStone templates, which aggregate around 1600 templates lines. Nevertheless, more than 95% of those lines originated from new XML and PHP configurations necessary for supporting CloudStone, and the remainder were reused from existing templates. To show the significance of the templates changes on the generated code, we created a specification with the same number of nodes for both RUBBoS and CloudStone. We generated the resources for Emulab, and we measured the differences, as shown in Figure 20(b). We observed a significant difference in Web, App and Database tiers.

3.9.5.3 Supporting a New DB Middleware

To illustrate the flexibility of Expertus, we showed how much the template would need to change to support the MySQL Cluster [23] database given that Expertus already had support for C-JDBC [105] (with MySQL). Concretely, we calculated the number of template lines that changed during the migration from C-JDBC to MySQL Cluster. This involved a few template changes, and the creation of a few new templates to handle three node types for MySQL Cluster.

MySQL Cluster is implemented with a combination of three types of nodes (Management, SQL and Data nodes) as compared to a single type in C-JDBC. In addition, C-JDBC is easy to scale-out (i.e., to add more database nodes). To do so, one only needs to deploy a new database server and copy the data dump. In contrast, scaling-out MySQL Cluster requires a completely different procedure (i.e., database partitioning), and the database needs to be repopulated using SQL statements. Additionally, C-JDBC has only one database URL (i.e., the URL for C-JDBC), but MySQL Cluster contains multiple SQL nodes, which requires each application server to be configured to communicate with each SQL node. As a result of all these changes, moving from C-JDBC to MySQL Cluster involved changing about 24% of the database templates. However, once the template is created, we can easily scale up MySQL Cluster to any level with simple configuration changes.

Figure 19(c) illustrates the total number of template lines, the number of lines changed, and the number of lines that changed in the generated code. To obtain additional insight, we categorized the changes into four types: DB install, DB-configure, DB-other (start, stop, reset), and core changes. We then changed the specification for C-JDBC (with 8 database servers) to generate resources for Emulab cloud (using the same number of database servers). We analyzed the generated code in the same four categories mentioned previously. Next, we modified the experiment specification for MySQL Cluster and generated code for Emulab. As expected, we observed huge differences in the generated code, and this difference became non-trivial when we increased the number of nodes. Our results appear in Figure 20(c). As shown in Figure 19(c) and Figure 20(c), a small template change created a huge difference in the generated code.

3.9.6 Testing for Heterogeneous Data Formats

For the purpose of evaluating the robustness of the parser, the following file patterns were tested: 1) one header, 2) multiple header rows with sequentially corresponding data, 3) multiple header rows with non-sequential corresponding data, and 4) multiple header rows

appearing randomly in the file (with data occurring non-sequentially). We used the collected data, which matched these aforementioned patterns, and Table 3 outlines the observed results. We varied the file patterns varied and the structure of the header. Based on the selected sample, headers contained either one row or two rows. A header with one row, *Only Field Row Header*, only contained data fields. Alternatively, a header with two rows, *Record & Field Row Header* contained a row, which enumerated the data records, and another row, which listed the corresponding data fields for each record. For this latter case, the numbers of records and fields were varied from 1 to the original maximum values found in the sample: 8 and 16 respectively. If the headers were correctly matched to the applicable row of data, the specified test received a *PASS* grade; otherwise, it received a *FAIL* grade.

Table 3: Evaluation Summary of Supported File Formats

Pattern	Only Field Row Header	Record & Field Row Header
One header	PASS	PASS
Multiple header (sequentially data)	PASS	PASS
Multiple header (non-sequential data)	PASS	PASS
Multiple header (randomly headers)	N/A	FAIL

3.10 Related Work

Benchmarking is an essential approach used in both academia and industry to gain an understanding of system behavior, formulating and testing of hypotheses, configuring and tuning systems, obtaining solution information, and resolving performance bottlenecks. However, few efforts have been undertaken with the explicit purpose of building software tools for large-scale testing of distributed applications to reduce the complexity of benchmarking tasks and activities [65, 170, 185, 259, 260, 335]. The ZOO [259] has been designed to support scientific experiments by providing an experiment management language and by supporting automatic experiment execution and data exploration. Zenturio [260] on the other hand, is an experiment management system for parameter-based studies, performance

analysis and software testing for cluster and grid architectures.

DART [114] is an automated regression testing framework built on Emulab. It provides a set of primitives for writing tests for distributed systems, and it also provides a run-time mechanism to execute the tests in a fast and efficient manner. However, extending these tools to other platforms is still a challenging problem. In our approach, we try to develop a generic framework to facilitate large-scale experiment through automation. Compared to other existing approaches, our solution provides good extensibility, flexibility and modularity by smartly combining the most commonly used technologies.

Our project parallels several others using XML and XSLT for code generation. For example, the SoftArch/MTE and Argo/MTE teams have also had favorable experiences using XML + XSLT generators to ”glue” off-the-shelf applications together [95, 154]. Apache Axis2 [296] uses an XSLT-based code generator for producing multi-language stubs and skeletons from WSDL. The effort that most resembles our approach is Weevil [336], which also focuses on workload generation and script creation. The creators of Weevil observed some of the limitations with their approach, and they proposed four enhancements to explore richer scenarios to obtain results with higher degrees of confidence [335]. To our knowledge, these efforts have not explored the issues of extensibility, flexibility, or modularity that are presented here in this chapter.

3.11 Summary

Expertus, our automated experiment management framework, has been developed to minimize human errors and maximize efficiency when evaluating computing infrastructures experimentally. We have continuously used our experiences and research work to improve the quality of the framework. In this chapter, we illustrate how our approach can be used to employ several traditional benchmark applications to evaluate heterogeneous cloud platforms. Our evaluation results, presented in this chapter, show the feasibility and usefulness of our approach for cloud benchmarking. We have also identified several limitations of our

tools and continue our research work to address them. Nonetheless, we believe our framework and results clearly document the need for a novel, large-scale approach to experimental analysis, like Expertus, because of cloud applications' novel characteristics. Our tool can significantly reduce the deployment and configuration cost of running n-tier benchmark experiments in today's production cloud environments, which indicates great promise for the future.

CHAPTER IV

FAULT TOLERANT DEPLOYMENT RUNTIME

Infrastructure-as-a-Service (IaaS) cloud environments provide users with the infrastructure of a data center while relieving them of the burden and costs associated with its management and maintenance. IaaS clouds provide an interface, so users can create, configure, and control a set of virtual machines that typically host a composite of software services. In the previous chapter, we presented an automated approach to large-scale experiment management that helps to better understand the cloud prior to moving applications to cloud. This work is concerned with the runtime state of composite services during and after their deployment. We propose Activation Engine on Service Overlay Network (AESON)— a deployment runtime, which uses information describing the relationships among the service components, that automatically detects node (virtual machine) failures and eventually brings the composite service to the desired deployment state. The deployment runtime uses the generated scripts during both activation and recovery. We have designed AESON as a decentralized peer-to-peer publish/subscribe system leveraging IBM's Bulletin Board (BB), a topic-based distributed shared memory service built on top of an overlay network.

4.1 Introduction

The emergence of cloud computing has enabled a new model of deployment and management of applications and services. In particular, with the advent of Infrastructure-as-a-Service (IaaS), many companies started relying on external cloud providers to host some of their Web and enterprise applications. One of the motivations for the adoption of IaaS clouds is to shift the burden of IT management to the cloud provider so that companies can focus on their own core businesses.

Typical applications deployed to IaaS clouds require complex software stacks comprising a multitude of distributed, cross-dependent components such as load balancers, Web, application, and database servers. At deployment time, all required software components and middleware must be cross-configured to ensure the application works correctly, and the entire infrastructure must also be configured to support non-functional requirements such as performance, availability, and security. Not surprisingly, deploying and correctly configuring such applications is a non-trivial, error-prone proposition. In order to mitigate this problem, previous work has focused on modeling such applications to automate their deployment and configuration in IaaS clouds ([133, 169, 244]).

Given the massive scale of the data centers of IaaS providers, which comprise a large number of heterogeneous hardware components, it comes as no surprise that hardware failures happen often in these environments, not to mention human mistakes made during data center operation, aggravating this problem. Failures in the IaaS data centers can translate into partial or complete outages of the hosted applications, potentially causing the affected companies to suffer revenue losses. Unfortunately, since the cloud provider lacks semantic knowledge of the hosted applications and their topologies, it cannot be trusted to recover them from failures, which may entail application-specific actions including restarting some of its components and reconfiguring others. Hence, part of the management burden still lies on the hosted application owners.

We advocate that the deployment of an application in an IaaS cloud must encompass a self-contained solution to properly recover the application from failures should the hosting cloud misbehave. Therefore, we advance the state-of-the-art by going beyond mere automation of the initial deployment and configuration; we propose AESON (Activation Engine on Service Overlay Network), a deployment runtime that automatically detects node (virtual machine) failures and eventually brings the application to the desired state by using a model describing the relationships between the application components. AESON relies on the application model to deploy the application as well as to recover it from failures that may

happen during and after deployment. At AESON's core is a decentralized peer-to-peer publish/subscribe system leveraging IBM's Bulletin Board (BB) [85], a topic-based distributed shared memory service built on top of an overlay network.

AESON is a lightweight and flexible runtime with negligible performance overhead, capable of tolerating simultaneous failures. It can handle three types of failures: *node crash*, *node hang*, and *application component failures*. AESON guarantees correctness (execute actions in correct order) and eventual completeness (execute all actions) of the application deployment and recovery. We have experimentally verified AESON's completeness, correctness, and single and multiple concurrent recoveries, as well as evaluated its performance overhead.

The remainder of this chapter is structured as follows. We introduce key concepts and formal definitions used throughout the chapter in Section 4.2, describe AESON's approach and architecture in Section 4.3, discuss key properties of AESON in Section 4.4, evaluate its performance overhead in Section 4.5, summarize the related work in Section 4.6, and provide a chapter summary in Section 4.7.

4.2 Background

4.2.1 Composite Creation and Activation

We define a *composite* as a collection of inter-dependent virtual machines (VMs) on which a distributed application runs to provide a service. E.g., a composite for a Bulletin Board application, such as RUBBoS [25], could comprise one Apache server, two Tomcat servers, and one MySQL server.

A composite can be modeled through a virtual image construction tool [244] as follows (see Figure 21). First, the user chooses a base virtual image and customizes it by adding the software packages required by the application (we describe image creation in detail in [244]). In Figure 21, three virtual images (Apache, Tomcat, and MySQL) are created. The next step is to model the composite by specifying the images to use, the number of instances of each

image, and the connections between the instances. In our sample, we depict a four-node composite model using the previously created images. The resulting composite model can later be used to deploy the composite, i.e., to instantiate and configure the VMs.

A composite model created by the above process contains operations (in the form of scripts) to be run at deployment time to configure and cross-configure the software components, as well as operations to be executed during failure recovery. The composite model captures all cross-dependencies between the defined operations, as exemplified in Figure 22.

We define *activation* as the fully automated process of starting the VMs for a given composite and running the deploy-time operations to configure the application components on those VMs so that the application can function properly. During the deployment of the composite modeled in Figure 21, one Apache VM, two Tomcat VMs, and one MySQL VM will be started. Next, the deploy-time operations executing on the VMs take care of, for instance: configuring Apache to act as the load balancer for the two Tomcat VMs; deploying the application logic to Tomcat; and configuring Tomcat to access the application database on the MySQL VM. In the chapter, we refer to each individual operation performed on deployment as an *activation action*, and to each operation performed during recovery (when a failure occurs) as a *recovery action*.

4.2.2 Definitions and Terms

In this section we formally define some of the terms that are used in rest of the chapter.

Definition 1 *Execution of an activation (or recovery) action is an atomic event that is either completed successfully in time t_{max} or fails in one of two ways: a system error cause it to terminate before t_{max}, or it is marked as a faulty execution after t_{max} (the user-specified value for the maximum execution time) has elapsed.*

Definition 2 *Dependency* $\mathrm{d} = \{a_1, VM_1, a_2, VM_2\}$ *is a quadruple, where action a_1 on VM_1 should be executed only after action a_2 on VM_2 is completed.*

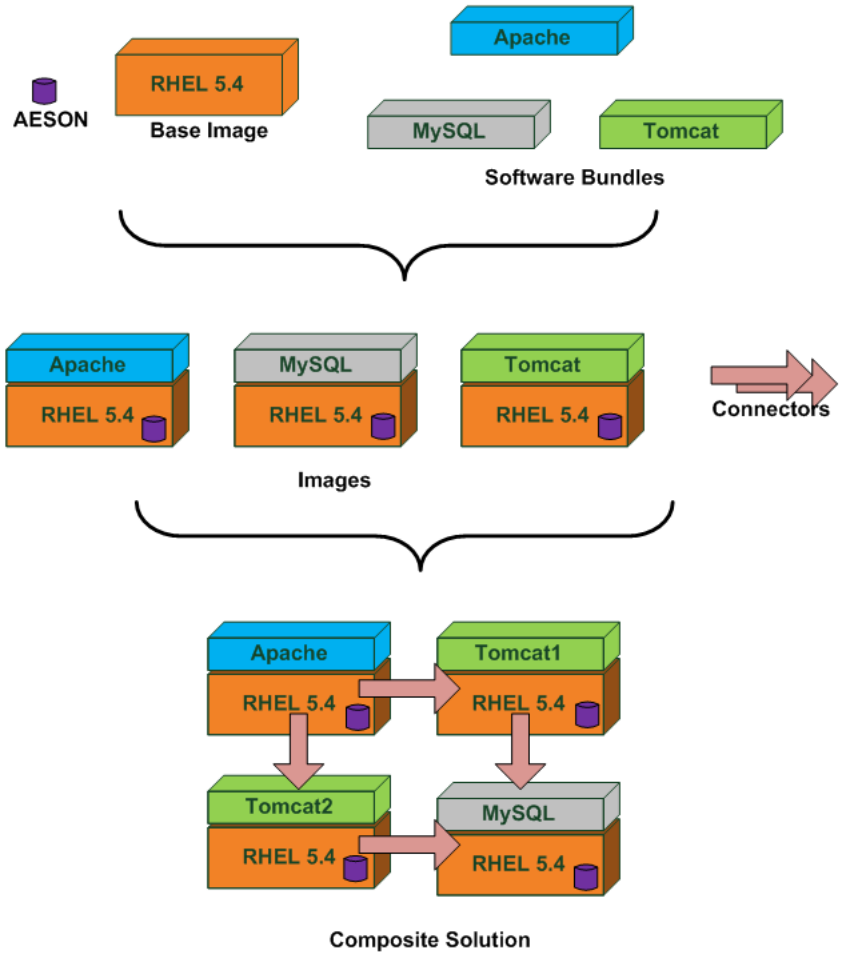


Figure 21: The Process of Creating Images and Modeling a Composite Service.

Definition 3 *Sensitivity* s = $\{r_1, VM_1, r_2, VM_2, \mathrm{l}\}$ *is a quintuple, where recovery action* r_1 *on* VM_1 *should be executed if recovery action* r_2 *on* VM_2 *has been executed and local state of* VM_1 *has reached the level* l.

Definition 4 *Parameter propagation* p = $\{a_1, VM_1, p_1, a_2, VM_2, p_2\}$ *is given by a sextuple, where action* a_1 *on* VM_1 *takes a set of input parameters* p_1 *from the set of output parameters* p_2 *produced by executing* a_2 *on* VM_2.

Definition 5 *Main recovery refers to recovering a failed node by either restarting or recreating a virtual machine.*

Definition 6 *Secondary recovery refers to the execution of recovery actions on non-failed*

nodes, which is driven by the specified sensitivity of the nodes.

To illustrate the concepts discussed above, Figure 22 shows a snippet of the configuration file of our sample composite's Tomcat server. It defines two activation actions, namely `deploy` and `configure`. The `deploy` action does not depend on any other activation actions; it deploys the RUBBoS Web application to the Tomcat container. In contrast, Tomcat's `configure` action depends on MySQL's `configure`, since the former consumes as an input parameter the output parameter (MySQL URL) produced by the latter. Finally, the configuration file defines a *secondary recovery* action `reconfigure`, stating that the Tomcat server is sensitive to MySQL's `restart` recovery action after Tomcat has reached the level `configure`. In other words, on failure, if the MySQL VM is restarted, the Tomcat VMs will run `reconfigure`, provided they have already run the `configure` operation.

```
<Activation>
  <VirtualSystem id="rubbos_tomcat">

    <ProductActivation class="deploy">
      <Program cmdLine="true"  href="AS/deploy.sh"/>
    </ProductActivation>

    <ProductActivation class="configure">
      <Properties>
        <Property key="mysqlurl"/>
      </Properties>
      <Program cmdLine="true"  href="AS/configure.sh"/>
      <ProductDependency class="configure" vsId="rubbos_mysql">
        <PropertyMapping sourceKey="mysqlurl" targetKey=" mysqlurl "/>
      </ProductDependency>
    </ProductActivation>

    <ProductActivation class="reconfigure">
      <Sensitivity sensitivity="restart" vsId="rubbos_mysql" level=" configure "/>
      <Program cmdLine="true" href="AS/configure.sh"/>
    </ProductActivation>
  </VirtualSystem>
</Activation>
```

Figure 22: A Snippet of the Configuration File for the Composite's Tomcat Server.

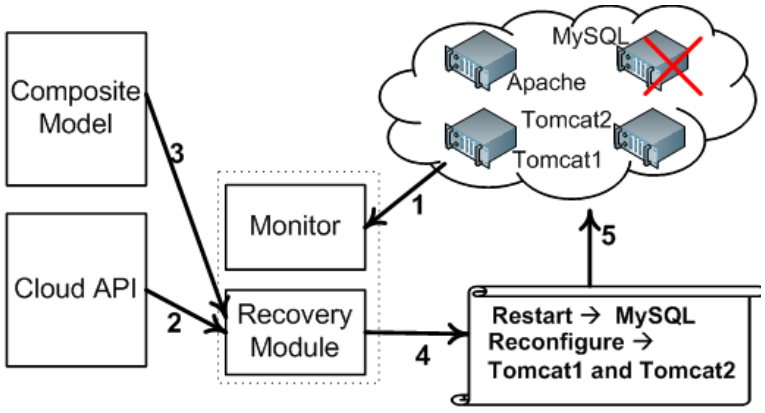


Figure 23: AESON: Monitoring and Failure Recovery.

4.3 AESON - Approach

AESON performs three key activities for a composite service: *activation*, *monitoring*, and *recovery*.

It constantly monitors the status of the composite service during and after deployment. In the event of a node failure, AESON detects the failure, determines an appropriate recovery plan (a subset of the defined recovery actions), and executes the plan to bring the service back to normal operation. This is illustrated in Figure 23, where the RUBBoS application is (being) deployed and a failure occurred in the MySQL server (1). The monitor detects the failure and notifies the recovery module; subsequently, the recovery module talks to the cloud API to execute the required *main recovery* action (restart or recreate the failed VM) (2). Next, by using the composite model, appropriate recovery actions are executed on non-failed nodes (in the RUBBoS case, Tomcat1 and Tomcat2 are reconfigured) (3). As we will discuss later in the chapter, the executed recovery plan is created in a distributed manner by using the specified sensitivities (definition 3). After the recovery plan execution, the service will be back to normal operation (if failed after deployment), or to a clean state (if failed during deployment) from which AESON can take the service to the desired deployment state.

4.3.1 System Architecture

Our primary goal when designing AESON was to embed in a composite a management middleware responsible for recovering it from failures. To realize such *self-contained* management, we designed AESON as a peer-to-peer (P2P) system. In this design, the role of orchestrating deployment and recovery is *distributed* to all nodes; we leverage the multi-node nature of composites to make AESON itself fault-tolerant: it runs on each node, avoiding a single point of failure (see Figure 21). In contrast, a self-contained centralized orchestration approach would exhibit a single point of failure.

For AESON, each composite node is a management peer. This P2P design naturally captures the dynamic nature of cloud applications, where nodes (peers) can fail and join at any time. AESON leverages the publish/subscribe communication mechanism implemented by IBM's BBSON [85], which offers an abstraction that further facilitates our design. Since a publisher can publish even with no subscribers, the separation between communication and execution of activation/recovery actions is clean and elegant.

As shown in Figure 24, AESON consists of four main components:

Configuration

Every action that AESON takes is based upon the composite model, which dictates what to do during both normal activation and recovery.

Communication

Each peer communicates using the BBSON APIs for publishing and subscribing to *topics*. AESON maintains one topic per node for activation and recovery orchestration, plus two topics used for leader election, as explained later.

Failure and Recovery

Constant status monitoring is needed to detect failures and recover from them. AESON uses a monitoring utility provided by BBSON.

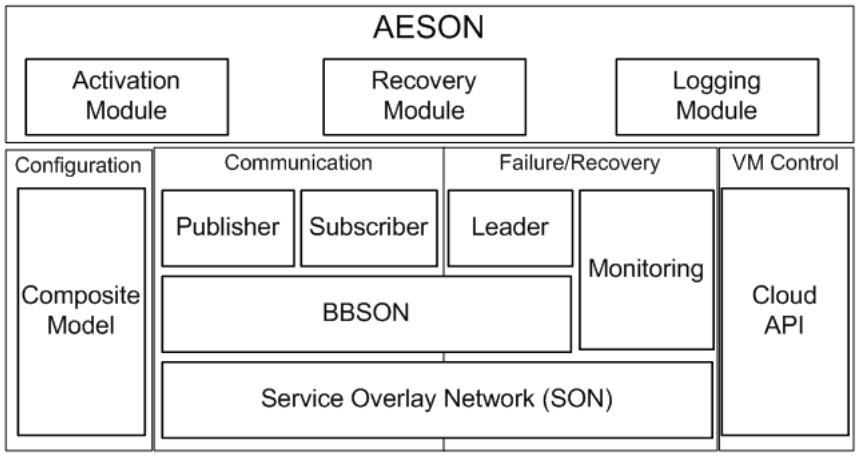


Figure 24: AESON Architecture.

VM Control

Interactions with cloud APIs are needed to start, stop, create, re-create, and restart VMs.

AESON comprises *activation*, *recovery*, and *logging* modules. The activation module is responsible for tasks related to normal activation, such as: checking, validating, and enforcing dependencies; and execution of activation scripts and related parameter propagation. The recovery module is used during failure recovery. It checks the sensitivity of nodes and executes appropriate recovery actions, propagating parameters when recovery actions so require. The logging module logs all local ongoing activities and their statuses, so that each node that is restarted can know which of its actions it has completed and which ones it has yet to perform.

4.3.2 Normal Activation

Normal activation refers to the process of deploying and configuring a composite service in the absence of failures. For example, in the RUBBoS application case, at the end of the normal activation an end-user should be able to access the Web portal, view and write comments. At start-up, each node first joins the AESON group of the composite; only after all nodes have joined the group does normal activation start. It ends after all activation

actions are completed successfully, at which point the composite will have reached the desired deployment state as specified by the composite model. In Section 4.4.1 we highlight how normal activation eventually ends in AESON.

During normal activation, the nodes collaborate and coordinate with each other to reach the desired state. Each node performs the following activities when it executes a normal activation action. (1) It checks whether all dependencies are fulfilled; if not, it asks the status of the missing dependencies by posting on its own topic and waits until all dependencies are satisfied. (2) If all dependencies are satisfied, the node publishes on its topic indicating that it is going to start the action. (3) The node executes the script associated with the action by passing all the required input parameters. Those parameters may come from the model or from another script on which this action depends (e.g., database URL may come from database-config.sh). Next, the node extracts any output parameters (output produced by the script), and publishes the status of the execution along with the output parameters. If the execution completed as a failure, then it informs the group, which will terminate the activation process across all nodes.

AESON checks the cross-dependencies of actions to guarantee a correct execution order. If a dependency of an action to be run has not been satisfied yet, the dependent node publishes on its topic a status request. Once the affected node sees such a request, it updates the status of the requested action on its topic. Upon seeing a positive update, the node waiting for it can then run the dependent action.

4.3.3 Fault Model

AESON is designed to support three types of failures: *node crash*, i.e., it cannot be accessed through ICMP (ping) or through the cloud API; *node hang*, i.e., it cannot be accessed through ICMP but can be accessed through the cloud API; and *application component failure*, i.e., the node can be accessed through both ICMP and Cloud API, but an application component running on the node has either crashed or hung. Node crashes and hangs can be addressed by

using two levels of application-independent monitoring; in contrast, to identify application component failures we need application-specific monitoring. The current prototype is implemented to monitor node crashes and hangs only; application-specific monitoring will be addressed in future work.

We have leveraged BBSON to realize our fault model. Each node constantly keeps sending and receiving heartbeats. In this model, a node is assumed to be failed if other nodes do not receive heartbeats for a specified time period. Thus, when a VM hangs or crashes, AESON detects it through BBSON. Subsequently, AESON waits for the specified time to see whether the failure is intermittent or permanent; if it is permanent, AESON uses the cloud API to take the required main recovery action and trigger the recovery plan. In our fault model we use two types of operations (main recovery) to recover a failed VM: **restart** and **create**. If the VM cannot be accessed through ICMP but can be accessed through the cloud API, then the resolution is to restart it; otherwise, AESON creates a new VM. Also, if the restart resolution does not solve the problem, AESON creates a new VM.

In a restarted VM, the recovery module runs only the activation actions that have not been completed. It uses the logging module to find the completed actions. In contrast, in a newly created VM, all normal activation actions will run. Notably, doing either restart or create on the failed node has consequences on non-failed nodes as well. For example, if a VM that runs the database is recreated, then the application server may need to restablish its database connection. Hence, non-failed nodes may also need to perform some actions as part of the recovery. In the next subsection we will detail the recovery process on non-failed nodes.

4.3.4 Recovery

The recovery process is divided into two stages: ***main recovery*** (recovering the failed node, e.g., MySQL server in Fig 23) and ***secondary recovery*** (recovering the non-failed nodes, e.g., Tomcat1 and Tomcat2 in Fig 23). In main recovery, AESON decides the appropriate

recovery action (i.e., restart or create) by retrieving the status of the failed node via the cloud API. In contrast, secondary recovery is solely driven by the composite model. Given the model and the failure, the nodes will be able to coordinate to execute the underlying recovery plan.

The execution of secondary recovery takes place in a distributed and dynamic manner. Concretely, each node decides the next needed recovery action based on the actions performed by other nodes, its current local state, and its *sensitivities* defined in the model. In our example of Figure 23, assume a recovery action is "restart the MySQL server." If the application server has not deployed the Web application yet, then no recovery action is needed. However, if the application server has already created database connections, then it needs to restablish them. In other words, the application server needs to perform a recovery action due to the database failure.

When a node failure is detected during deployment, normal activation is paused on all nodes until the recovery process is completed. However, if a node is in the middle of a normal activation action, it completes the ongoing action and then abstains from executing any further activation actions until the recovery has been completed.

During recovery, AESON has the notion of an elected "leader" (see Algorithm 1). A leader is needed mainly to prevent two nodes from initiating the main recovery action (i.e., restarting or recreating the failed VM). The leader eventually finds the node failure and initiates the main recovery action. Once the newly created or restarted VM is active, it automatically joins the AESON group. If the VM is restarted, it already has the IP address of all other nodes; thus, it uses one of those as the bootstrap node[1]. In contrast, if the VM is recreated then a bootstrap node IP address is passed as a parameter to the VM.

When the failed node rejoins, the leader informs the group indicating that it has successfully completed the main recovery. Consequently, if a node has sensitivity to the main recovery action, then it will run the recovery action on that node, which drives recovery

[1]If a new node wants to join a P2P system, then it is required to know the address of at least one peer.

action(s) on other nodes based on their sensitivities. The leader notifies the group when all sensitivity-driven recovery actions have been executed. Once the recovery is completed, normal activation is resumed (if the failure happened during deployment). One should note that the newly rejoined node performs normal activation actions and does not perform any recovery actions.

At any given time AESON can handle two types of recoveries: single recovery (i.e, only one ongoing recovery) and multiple recoveries due to multiple failures, including failures that happen during ongoing recoveries.

4.3.4.1 Single Recovery

In this case, the leader eventually finds the node failure and initiates the main recovery action. Importantly, if the leader happens to be the failed node, then another member of the group is elected the new leader; the newly elected leader initiates the main recovery action. Once the main recovery action is completed, a chain reaction starts, where the execution of one recovery action may trigger recovery actions on other nodes. Once all secondary recovery actions are completed, the leader marks recovery as finished and notifies the group. Upon receiving the recovery-completed notification, each node resumes normal activation, if the failure happened during deployment, or else the system is ready.

4.3.4.2 Multiple Recoveries

It is conceivable that more than one node can fail simultaneously, or that a node can fail during the recovery of another failure. Regardless of the way in which nodes fail, in AESON, secondary recovery is performed only when all nodes are connected in the group. Hence, the first task is to initiate the main recovery actions for the failed nodes. Once all nodes are recovered (rejoined the group), AESON resumes/starts the secondary recoveries, from the latest sequence to the first. In effect, multiple failures become a series of single recoveries, where each single recovery corresponds to a separate recovery plan.

A major challenge in handling multiple recoveries is the leader failure in the middle of

- Step1: Subscribe to the `Election` topic and publish process ID to the Election topic.
- Step2: Subscribe to the `LeaderInfo` topic.
- Step3: Wait until the membership of the leader is posted in `LeaderInfo` by a single process, or a timeout expires.
- Step4: If the timeout expires before the leader info is available in `LeaderInfo`, elect a new leader using the `Election` topic:
  - Select the preferred process with minimum ID among the publishers to the `Election` topic as the current leader.
  - If not selected as a leader, go to Step3.
- Step5: When there are more than one leader published in the `LeaderInfo` and I am not the leader with the minimal ID, then unpublished myself from `LeaderInfo`.
- Step6: When the leader process fails (its record disappears from the LeaderInfo topic):
  - If I am the node with the preferred node with the minimal ID published in the `Election` topic and post myself in the `LeaderInfo`.
  - If not selected as a leader, go to Step3

Algorithm 1: Sketch of AESON's Leader Election

ongoing recoveries. For example, assume multiple nodes failed and for some (not all) of them the leader called the cloud API to start the main recovery; later, the leader crashes. As mentioned earlier, in the event of the leader failure, a new leader will be elected using the leader election algorithm (see Algorithm 1); however, the new leader needs to start from where the previous leader has left off. This is possible with our approach because each peer in the system has the same view on the current system state. Thus, when the leader fails, the new leader can take over and guide the recovery of the system.

4.4 System States and Properties

In this section, we briefly outline AESON's state transitions that happen during both normal activation and recovery. We also present key properties that AESON supports.

Figures 25(a) and (b) represent, respectively, AESON's global state transitions and local state transitions. The global state is an aggregated view of the local states of all nodes. For

the reader to better understand the figures, we define below some of the terms used:

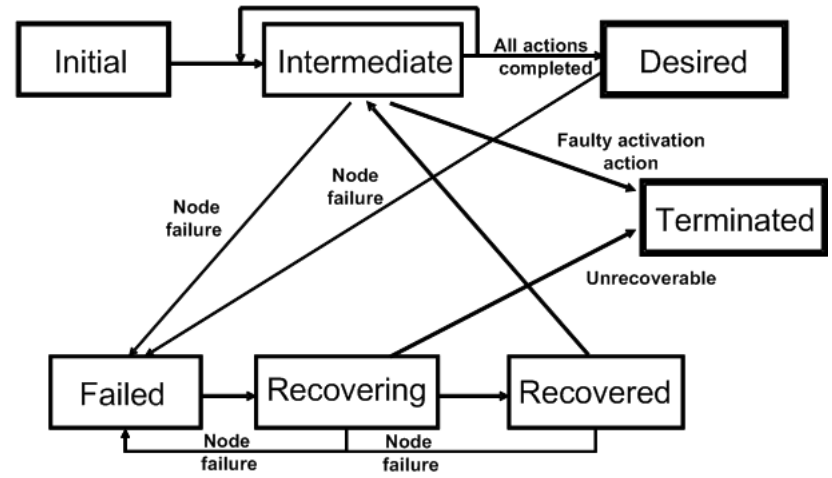
- **Initial**: For the global case, it means that all nodes in the composite have joined the group of peers but none has started normal activation. For the local case, the node in question has joined the group.
- **Desired**: It refers to the completion of normal activation and/or recovery.
- **Intermediate**: It is a global state between *initial* and *desired*.
- **Terminated**: It is the state resulting from AESON's aborting its operation due to an unrecoverable failure or a permanent execution error exhibited by an activation or recovery action.
- **Dirty**: It refers to the time during which an activation or recovery action is being executed.
- **Failed**: AESON has detected a failure, but the corresponding recovery plan has not started yet.
- **Recovering**: AESON is in the middle of a recovery plan execution.

In a nutshell, the local state transition graph shows the states each node can be in while AESON performs its activities for activation and recovery, whereas the global state transition graph summarizes the holistic view of the system, based on all local states.

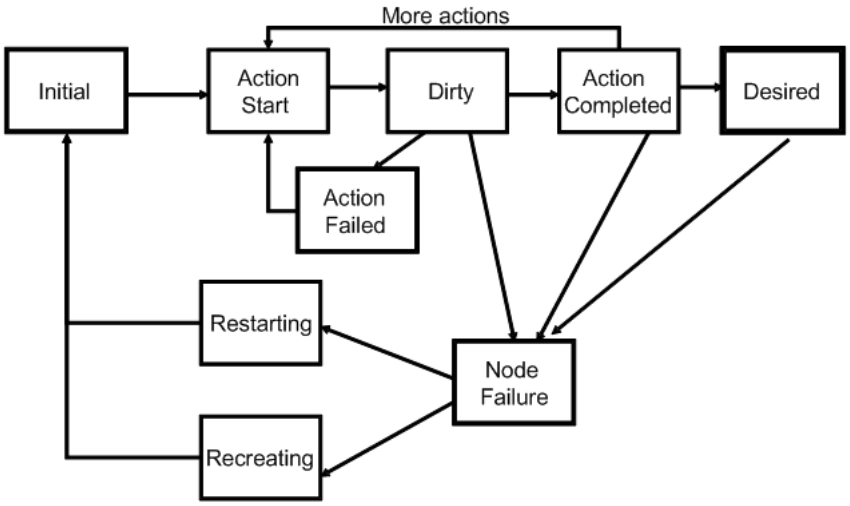
4.4.1 System Properties

We now outline important properties supported by AESON, some of which are directly inherited from BBSON [85].

Dependency Preservation: Let X and Y be activation actions on two VMs, VM_1 and VM_2, respectively. If Y depends on X, then X finishes on VM_1 before Y starts running on



(a) Global State Transitions



(b) Local State Transitions

Figure 25: State Transition Diagram for Global and Local States.

VM_2. AESON supports this because of two BBSON properties: eventual inclusion and correctness.

Timeout Enforcement: Let the user-specified maximum time for an activation or recovery action X be t_{max}. X's execution will be either successfully completed in time t (where $t \leq t_{max}$), or marked as faulty otherwise.

Sensitivity Enforcement: Let P and Q be two recovery actions on two different VMs. If P is sensitive to Q, then if Q is executed, P will be executed. *P's* execution will start after *Q's* is completed.

State-Change Enforcement: Let VM_1 and VM_2 be two different VMs. Suppose that VM_1 performs an action P; as a result, it will publish *P's* execution completion notification on its own topic. This notification will eventually reach VM_2, since all VMs subscribe to the each other's topics and BBSON guarantees eventual consistency. Furthermore, in the case of recovery, AESON implements a two-phase protocol to ensure that no state change is lost (see Algorithm 2). With BBSON, given two consecutive posts to a topic, the subscribers may only see the last one. During the execution of recovery plans, AESON's implementation relies on not missing any posts.

Based on the above properties and those guaranteed by BBSON, we can make the following claims.

4.4.2 Claim 1: Eventual Activation Completeness

Under the assumption that each provided activation action behaves normally and finishes its execution within time t_{max}, AESON guarantees that all activation actions of the composite will complete.

Eventual activation completeness, as herein defined, comes as a corollary of *state-change enforcement*. Since all nodes will be eventually notified of the execution progress of each activation action, all actions will eventually complete, even the ones that depend on the execution of others.

4.4.3 Claim 2: Correctness

As a corollary of *dependency preservation* and *sensitivity enforcement*, AESON guarantees that activation and recovery actions will be executed in a correct order, as defined in the composite model.

4.4.4 Claim 3: Eventual Single-Recovery Completeness

In the case of a single-node failure, AESON guarantees that the composite will be restored to the desired state, under the following assumptions: (1) each provided recovery action

- Step1: When a node N_i wants to enforce the state transition for action a_i N_i posts a state update message on N_i's topic.
- Step2: All the connected nodes respond by posting $ACKs$ on their topics.
- Step3: If N_i unable to receives $ACKs$ from all the connected node, then it waits time t and reposts the state update.

Algorithm 2: Two Phase Protocol for State Enforcement

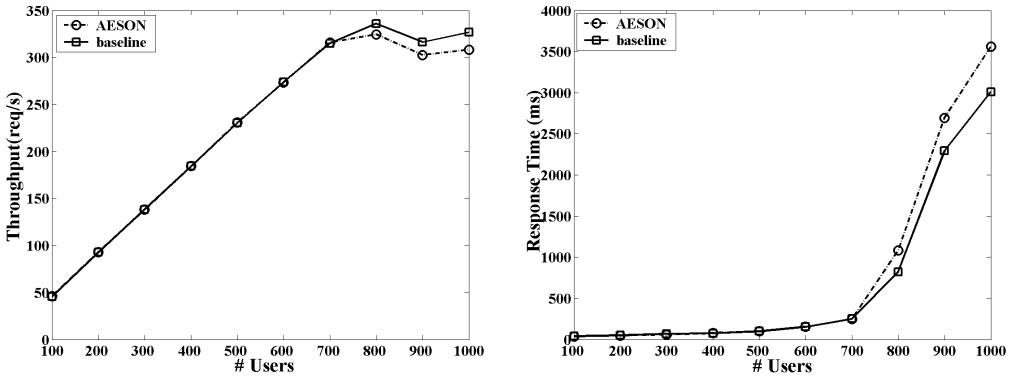
behaves normally and finishes its execution within time t_{max}; and (2) the main recovery action (VM restart or create) fixes the observed node failure.

Given BBSON's eventual exclusion property, i.e., when a peer leaves the group all nodes eventually notice it, all nodes will eventually perceive the node failure. Because of *state-change enforcement*, all nodes will be notified of the initiation and completion of the main recovery action. Similarly, all nodes will be notified of the execution progress of each secondary recovery action. Therefore, all recovery actions will eventually complete, regardless of their sensitivities (dependencies).

4.4.5 Claim 4: Eventual Multiple-Recovery Completeness

When facing simultaneous failures, or failures that happen during an ongoing recovery plan, AESON guarantees that the composite will be restored to the desired state under the following assumptions: (1) each provided recovery action behaves normally and finishes its execution within time t_{max}; (2) the main recovery action (VM restart or create) fixes the observed node failure; and (3) at least one of the nodes does not fail.

Each failure is eventually perceived due to BBSON's eventual exclusion property. Because of *state-change enforcement*, all nodes will be notified of the initiation and completion of each main recovery action. Similarly, all nodes will be notified of the execution progress of each secondary recovery action, of all recovery plans. Therefore, all recovery actions will eventually complete, regardless of their sensitivities (dependencies).



(a) Throughput **(b)** Response Time

Figure 26: Performance Overhead of AESON (Throughput and Response Time Comparison).

4.5 Performance Evaluation

We have experimentally verified AESON's completeness, correctness, and fault tolerance.However, due to the associated complexity of graphical representation, we have limited our discussion only to the performance overhead of AESON, which follows.

In order to assess AESON's performance overhead, we modeled and deployed a composite for the RUBBoS Bulletin Board benchmark application [25]. Our composite comprises one Apache server, two Tomcat servers, and one MySQL server. Each individual application component is assigned to a dedicated host and each host has a running instance of AESON. As a baseline for comparison, we also manually deployed the RUBBoS composite, running it without AESON. We varied the RUBBoS workload intensity by generating requests originating from 100 to 1000 users. We measured the client-perceived performance overhead (response time and throughput) as well as the resource utilization at each server.

Each data point shown in the graphs of Figures 26 and 27 represents the average of 3 experiment runs. The observed average throughput and response time values are shown in Figure 26(a) and Figure 26(b) respectively. As shown in Figure 26, AESON shows a performance similar to the baseline case up to 700 users. From that point on, AESON shows an average of 6% performance drop.

We also analyzed the CPU utilization in the deployed servers (see Figure 27(a)) and found that the observed performance degradation (for both baseline and AESON) is caused by a high CPU utilization at the Tomcat servers. For low CPU utilization, both AESON and the baseline have a negligible difference in throughput and response time; when the system is highly utilized, AESON causes a small performance overhead (6%).

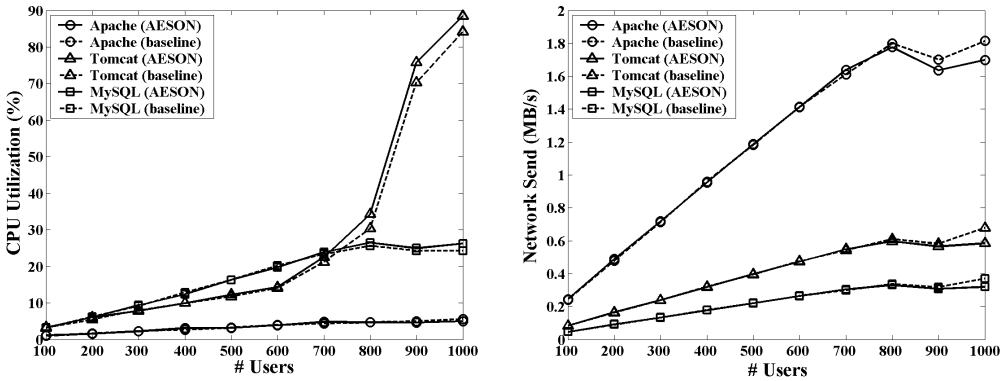
AESON is designed as a P2P system and at runtime each peer communicates by sending application messages, which causes network overhead. Thus, to understand the overhead we measured he number of network bytes sent and received at each server. The measured results for network are shown in Figure 27(b) and Figure 27(c). The network overhead introduced by AESON is negligible. Notably, after 800 users, the baseline approach exhibits higher network traffic than AESON. This is due to the throughput drop we observed in Figure 26(a). Since the throughput is higher for the baseline case, it processes more requests; hence, more data packets.

In summary, in our experiments, AESON showed a small performance overhead.

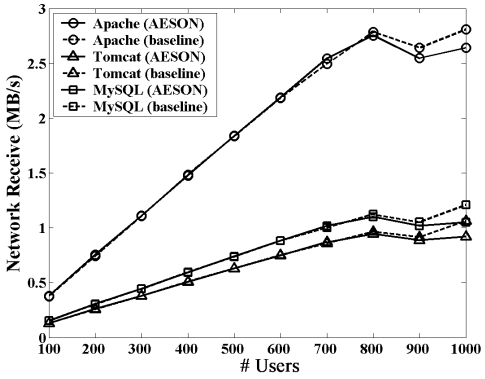
4.6 Related Work

The issue of detecting failures and doing something to fix them has been around as computers, for instance, this was first discussed in the concept of system diagnosis [257]. Some approaches have focused on automatic restarting of components before or after they have failed [91, 97]. There have also been some works in applying techniques from Markov decision theory to dependability problems [124]. Yet, most studies try to recover a service after deployment and focus on the mechanisms by which recovery can be automated and made more efficient, rather than determining when and where the recovery actions should be applied.

Yuanshun *et al*. studied the self-healing function from the consequence-oriented point of view [121]. To fulfill the self healing requirements of efficiency, accuracy, and learning ability, a hybrid tool that takes advantages from Multivariate Decision Diagram and Naïve



(a) CPU Utilization **(b)** Network Traffic (Send)



(c) Network Traffic (Receive)

Figure 27: Performance Overhead of AESON (Comparison of CPU Utilization and Network Traffic).

Bayes Classifier is proposed. Kaustubh *et al*. combined monitoring and recovery to realize the benefits that cloud not have been obtained by using them in isolation, they also presented two algorithms and evaluated those using fault injections [177]. A fine-grained technique for surgically recovering faulty application components, without disturbing the rest of the application is presented in [99]. Arun *et al*. proposed operating system virtualization techniques to provide automatic and transparent mechanism for proactive FT for arbitrary MPI applications [240].

A failure during a failure recovery process poses a tough problem because of the complexities involved, importantly, AESON is designed to address this problem. Previously, Naveed *et al* have also proposed an approach using the dependency model of the system to address a similar problem [63]. One of the biggest advantages of ours compared to most of

the approaches discussed above is we have designed AESON as a decenterlized P2P system to prevent the single point of failure. In addition, AESON does not make any assumptions about the application; second, it supports three types of faults; and third, AESON supports fault detection and recovery during both deployment and after deployment.

4.7 Summary

We presented AESON, a model driven fault tolerant composite service activation runtime to address dependability issues of enterprise applications that are deployed (and being deployed) in IaaS clouds. AESON uses a composite model to automatically heal from virtual machine failures during both deployment time and after deployment. To prevent the single point of failure, AESON is designed as a P2P pub/sub system by leveraging IBM Bulletin Board (a topic-based distributed shared memory service built on top of an overlay network). We experimentally evaluated the correctness, completeness, and ability to recover from single and multiple virtual machine failures. We analyzed AESON's performance overhead showing a negligible impact.

CHAPTER V

VARIATIONS IN APPLICATION PERFORMANCE AND SCALABILITY IN CLOUDS

Cloud computing is an emerging technology paradigm which provides the on-demand delivering of software, platforms, and infrastructure as services. The increasing popularity of clouds has driven the research community to find answers to a large variety of new and challenging questions. We aim to answer some of these questions by experimentally evaluating variances in performance and scalability when a multi-tier application is migrated from a traditional datacenter environment to an Infrastructure-as-a-Service cloud (IaaS). For our study, we used the automated infrastructure to automate the experiments for comparing the performance and scalability of three different clouds: Amazon EC2, Open Cirrus, and Emulab. We evaluated these three clouds using a representative multi-tier macro-benchmark (RUBBoS) with two different types of client workloads–a mixture of browse only and read/write. Interestingly, our experiment results show that a best-performing configuration in one cloud can become the worst-performing configuration in another cloud. Subsequently, we identified the bottleneck components, high context switch overhead and network driver processing overhead, actually occurred at the system level. These overhead problems were confirmed with finer granularity through micro-benchmark experiments that measure component performance directly. We contextualize our empirical findings on the basis of a profit model that illustrates their economical impact. Finally, we experimentally assess practical solutions to address these problems by evaluating a few concrete, alternative approaches.

5.1 Introduction

The flexibility and scalability of commercial clouds make them an attractive application migration target; yet, due to clouds associated complexity, running newly migrated applications efficiently in them is a difficult undertaking. For example, while clouds are good candidates for acting as supplementary system platforms for occasional Internet application (e.g., electronic commerce) overload, reports on modern clouds (e.g., Amazon EC2) consistently mention how virtualization and network latency can affect overall system performance [331, 332, 351]. Such issues are compounded by dependencies among system components as requests are passed among the tiers, which are characteristic of real, multi-tier applications. Despite some published best-practices for the popular cloud environments (e.g., [130]), the tradeoff between guaranteed performance (e.g., bounded response time) and economic efficiency (e.g., high utilization for sustained loads) remains a serious challenge for mission-critical applications. The goal of this chapter is to highlight some of these challenges as an intermediate step toward addressing them.

Here we analyze the performance and scalability characteristics when a multi-tier application is migrated from a traditional datacenter (on premises datacenter) to a shared Infrastructure-as-a-Service (IaaS) cloud. In our analysis, we use a representative multi-tier macro-benchmark application (RUBBoS [25]) and perform a large-scale experimental study on three testbeds. We use Emulab [21] (a more traditional compute cluster environment) as our reference testbed and compare the performance and scalability characteristics to Open Cirrus (a scientific research cloud with better isolation) and to Amazon EC2 [20] (a popular commercial cloud).

Our experiments cover scale-out scenarios under varying hardware and software configurations. We use two different types of multi-tier workloads for our experiments, namely a mixture of browse-only and read-write. These RUBBoS experiments are generated and executed automatically using the automated framework discussed in Chapter 3. Concretely, we use automated experiment management tools, which setup, execute, monitor, and analyze

large-scale application deployment scenarios. To compare three different clouds fairly, we looked at scalability characteristics (i.e., going up or down) rather than performance numbers (e.g., throughput). In fact, we looked at both "horizontal scalability" (increasing the number of compute nodes in the application) and "vertical scalability" (using different types of compute nodes e.g., small, large). For the detailed analysis of non-trivial performance issues and system bottlenecks, we use micro-benchmarks (both standard and custom) to zoom into individual system components and confirm our initial hypotheses under a higher degree of magnification.

In the course of our analysis, we found that configurations that work well in one cloud may cause significant performance problems when deployed in a different cloud. In fact, we found that the best-performing RUBBoS configuration in Emulab became the worst-performing configuration in EC2 due to a combination of factors. These factors such as network driver overhead and thread context-switching overhead were often subtle and not directly controllable by users.

We contextualize our experimental results on the basis of a profit model that illustrates their economic impacts. Furthermore, we provide a set of candidate solutions to overcome the observed performance problems. Our solutions are bound by only making application level modifications to overcome the issues induced by cloud platforms. In our study, we focus on a very small fraction of the problem; yet, we observe a set of interesting phenomena. More generally, this study shows that clouds are a relatively immature technology, and more significant experimental analysis is needed before public clouds can become truly suitable for mission-critical applications.

The remainder of this chapter is structured as follows. Section 5.2 provides an overview of the benchmark application, the experimental setup, and the tools used during the experiments. In Section 5.3 we analyze browse-only performance and scalability for the three platforms, and Section 5.4 presents read-write mix results. In Section 5.4.4, we discuss the performance impact associated with multi-threading. Section 5.4.5 provides results on

Table 4: Hardware Configurations in Three Clouds.

Platform	Type	Processor	Memory	Arch
Amazon EC2	Small	1 EC2 Unit	1.7 GB	32-bit
	Large	4 EC2 Units	7.5 GB	64-bit
	Ex Large	8 EC2 Units	15 GB	64-bit
	Cluster	33.5 EC2 Units	23 GB	64-bit
Emulab	PC3000	3GHz	2 GB	64-bit
Open Cirrus	X3210	3.00GHz (Quad)	4 GB	64-bit

network overhead, and in Section 5.5 we analyze the economic benefits of different hardware configurations. Related work is summarized in Section 5.6, and Section 5.7 summaries the chapter.

5.2 Overview of Experiments

The experiments discussed in this chapter were run in three testbeds (i.e., Emulab, EC2 and Open Cirrus). Table 4 shows the server types we used in three platforms (EC2 Compute Unit provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Xeon processor). We used RUBBoS benchmark application (see Appendix A.1) for our study and we used MySQL Cluster as the database clustering middleware (see Appendix A.3). We evaluated both horizontal and vertical scalability on EC2, in contrast we limited our analysis only to horizontal scalability in Emulab and Open Cirrus. By horizontal scalability we mean the increase of the number of nodes in the application (e.g., from 12 large instances to 16 large instances), and by vertical scalability we mean the same number of nodes but with better hardware settings (e.g., from 12 large instances to 12 ex-large instances). The nodes were connected over 1GB Ethernet in Emulab, Infiniband in Open Cirrus, and for EC2 the connection types were not exposed. To focus this study on the differences among three platforms, we allocated each server to a dedicated node (either to a virtual or physical). Specifically, the sharing of physical resources (e.g., CPU) in a node is among the threads within each server, not between servers. For each concrete hardware configuration we

observed the most appropriate software configuration (e.g., number of threads and thread pool sizes) by using the approach mentioned in our previous work [334].

Each of our macro-benchmark experiment trials consisted of three periods: 8-minute ramp-up, a 12-minute run period, and a 30-second ramp-down. Performance measurements (e.g., CPU or network bandwidth utilization) were taken during the Run period using lightweight system monitoring utilities (e.g., `dstat` and `sar`) with a granularity of one second.

While the macro-benchmark data gave us the interesting bottleneck phenomena described here, we often need more detailed data to confirm concrete hypotheses on the non-trivial causes of such phenomena. We have used both existing tools and custom tools to obtain such micro scale data. We employed a number of hardware monitoring and soft resource monitoring tools such as LMBench [28] and NetPipe [18].

Table 5 provides a high level summary of a subset of different experiments performed using RUBBoS application in three cloud platforms. In the table, experiment refer to an execution of particular workload against a combination of hardware and software configurations. Typically, an execution in our domain takes one hour which is the aggregated duration of: reset time, start time, sleeping time, ramp-up time, running time, ramp-down time, stop time, and data copy time. Hence, as an example in Eumlab we have conducted approximately 8,000 hours of experiments, however, amount of engineering hours spent is significantly less due to test automation. In the table, nodes refer to the total number of machines we have used during our experiments. We calculated the number of nodes by multiplying number of experiments by the number of nodes for each experiment. Configurations mean the number of different software and hardware configurations that have been used by our experiments. Finally, the number of data points collected gives an illustration of the amount of configuration we have evaluated and amount of data we have collected.

Table 5: Number of Different Experiments used for the Study

Type	Emulab	EC2	Open Cirrus
Experiments	8124	1436	430
Nodes	95682	25848	4480
Configurations	342	86	23
Data points	3,210.6M	672.2M	2.3M

5.2.1 Notation

We use a notation #A-#T-#M-#D-ppp to represent each concrete configuration. #A denotes the no. of Web servers, #T for no. of Tomcat servers, #M for no. of SQL nodes and #D for no. of Data nodes in the experiment. A short string (ppp) denotes the platform (`ec2` for EC2, `em` for Emulab, and `oc` for Open Cirrus). For example, Figure 61 shows a `1-2-2-4-em` configuration with 1 Web server, 2 Tomcat servers, 2 SQL nodes and 4 data nodes in Emulab. In EC2 we add the instance (node) type at the end (small, large, exlarge, or cluster).

5.3 Browse-Only Comparison

In this section we present our experimental results using browse-only workload on three cloud platforms with various hardware configurations. In Section 5.3.1 we present performance and scalability characteristics on reference platforms (i.e., Emulab) and compare it with Open Cirrus, and Section 5.3.2 provides scalability characteristics and performance issues on EC2.

5.3.1 Performance on Reference Platforms

To compare and contrast the performance complications of commercial clouds against other alternative platforms it is essential to have a good reference, Emulab and Open Cirrus were used for this purpose. Methodically, we start with the smallest possible configuration and then gradually scale up based on the observed results.

More precisely, we started on Emulab with `1-2-2-2-em` configuration (the smallest

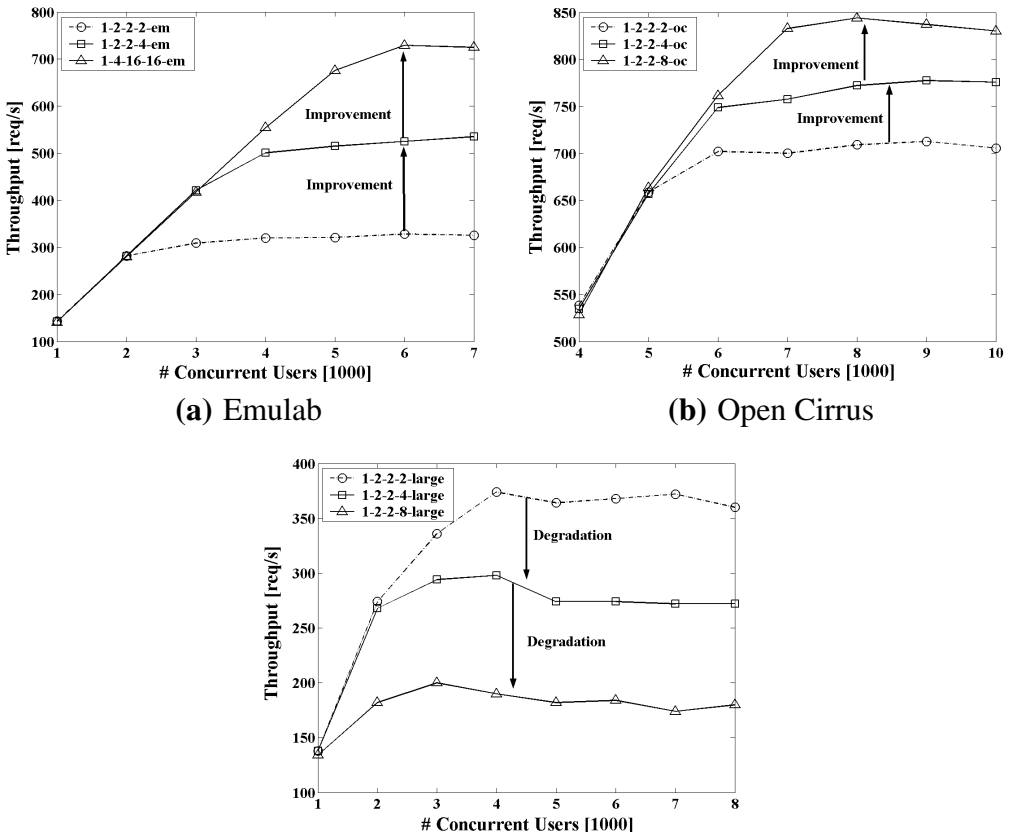
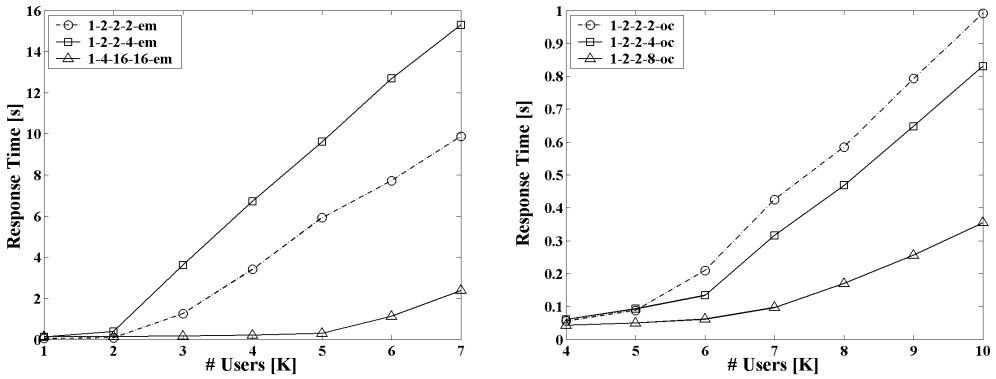


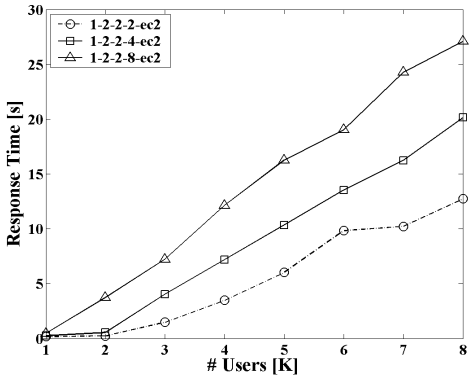
Figure 28: Horizontal Scalability: Read Only Throughput Analysis for three Clouds [Higher the better].

hardware configuration which can withstand 1000 workload); the configuration consists of two SQL nodes and two Data nodes, and we performed experiments from 1000 to 10,000 workloads (i.e., concurrent clients) with 1000 workload increments. As shown in Figure 28(a), for `1-2-2-2-em` configuration, throughput saturated at 3000 workloads. Detailed analysis of monitoring data shows that SQL node CPU utilization had become the bottleneck (see Figure 30(a)). To resolve it, we scale-out our experiment and introduce two additional Data nodes (`1-2-2-4-em` configuration) by partitioning the database into two groups. Running experiments with 4 Data nodes and 2 SQL nodes, we observed that knee point (i.e., throughput saturation point) shifted from workload of 3000 to 5000. Next, we increased the number of SQL nodes to 4 to get `1-2-4-4-em` configuration, likewise we continued this horizontal scalability process until we resolved the hardware



(a) Emulab

(b) Open Currus

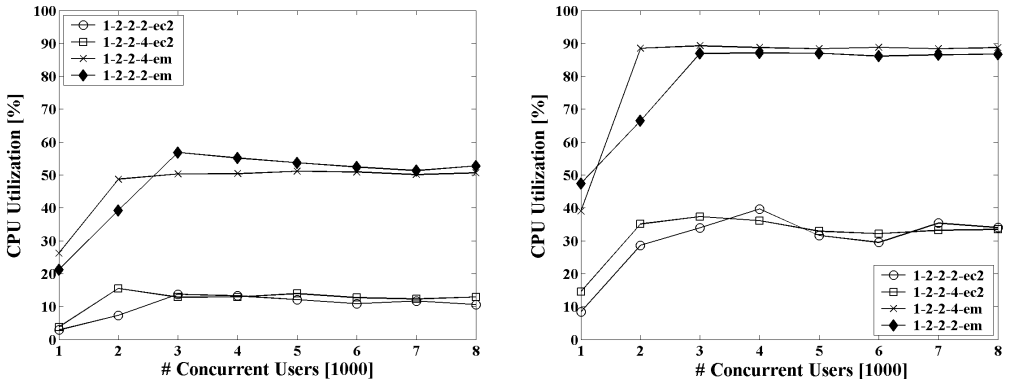


(c) Amazon EC2

Figure 29: Horizontal Scalability: Read Only Response Time Analysis for three Clouds [Lower the better].

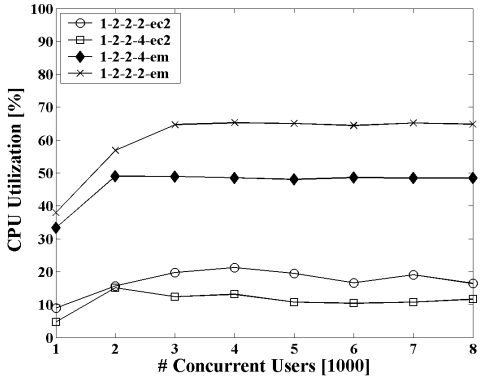
bottlenecks. Through our analysis we found `1-4-16-16-em` as the smallest configuration which can sustain our workload without system hardware resources being bottlenecked. Yet, `1-4-16-16-em` configuration shows software resource saturation, readers who are more interested in reading about software resource saturation refer to our previous work [334]. In summary, as shown in Figure 28(a) representative benchmark application scales well in Emulab.

In a web facing application, throughput is a simple indicator on how much workload that the application can handle. However, to understand actual user experiences it is necessary to analyze the response time characteristics (i.e., latency). Here we present end-to-end average response time values and our results are illustrated in Figure 29(a). As shown in the figure, `1-4-16-16-em` configuration has the lowest response time among all the configurations. It



(a) Tomcat CPU

(b) SQL CPU



(c) Data node CPU

Figure 30: Average CPU Utilization for `1-2-2-2` and `1-2-2-4` Configurations for Emulab and EC2.

gives very low average response time until the workload of 6000 and then started to increase, this behavior coincide with the observed throughput values (see Figure 28(a)). As shown in the figure other two configurations show sudden response time increase after workload of 2000 and 3000 respectively.

To better explain the observed performance characteristics on Emulab we have provided CPU utilizations for `1-2-2-2-em` and `1-2-2-4-em` configurations, Figure 30(a), 30(b) and 30(c) illustrate the CPU utilization in the Tomcat Server, the SQL nodes and the Data nodes respectively. As shown in the Figure 30(b), for both configurations SQL node CPU has become the bottleneck, and as a result we have a flat curve for throughput values (see Figure 28(a)). We extended bottlenecks analysis to other configurations and a high level summary of our results are shown in Figure 31(a). The figure shows each configuration and

observed average CPU utilization when system is saturated. The highlighted cells represent the bottlenecked servers for the configuration shown in x-axis.

We used the same procedures as in Emulab and performed a similar set of experiments on Open Cirrus cloud. As shown in Table 4 we used much powerful machine in Open Cirrus, thus as expected, we achieved comparatively higher throughput values, and more importantly, Open Cirrus also shows a good scalability. The observed throughput and response time values are illustrated in Figure 28(b) and Figure 29(b) respectively. As shown in the figure Open Cirrus also shows a good scalability and in fact it shows very low average response time values. It was mainly due to the better hardware we used for Open Cirrus and the Ethernet connectivity. In summary, both Emulab and Open Cirrus show good scalability characteristics, where observed average throughputs go up as number of nodes increase. Notably, in Open Cirrus we observed very low resource utilization (e.g., less than 30% CPU utilization), thus, we have omitted utilization graphs.

	1-2-2-2	1-2-2-4	1-2-4-4	1-2-8-8	1-4-4-8	1-4-8-8	1-4-16-16
DataNode	80	70	94	85	80	94	88
MySQL	95	100	95	80	95	87	60
Tomcat	75	60	92	100	50	65	70

(a) Read Only

	1-2-2-2	1-2-2-4	1-2-4-4	1-2-8-8	1-4-4-8	1-4-8-8	1-4-16-16
DataNode	98	95	96	88	94	82	60
MySQL	80	88	75	50	70	40	32
Tomcat	40	50	45	40	25	25	25

(b) Read Write

Figure 31: CPU Bottleneck Map for Emulab:(Each cell represents average CPU utilization when system is saturated and highlighted cells represent bottlenecked tier for the configuration specified in x-axis)

5.3.2 Amazon EC2 Performance

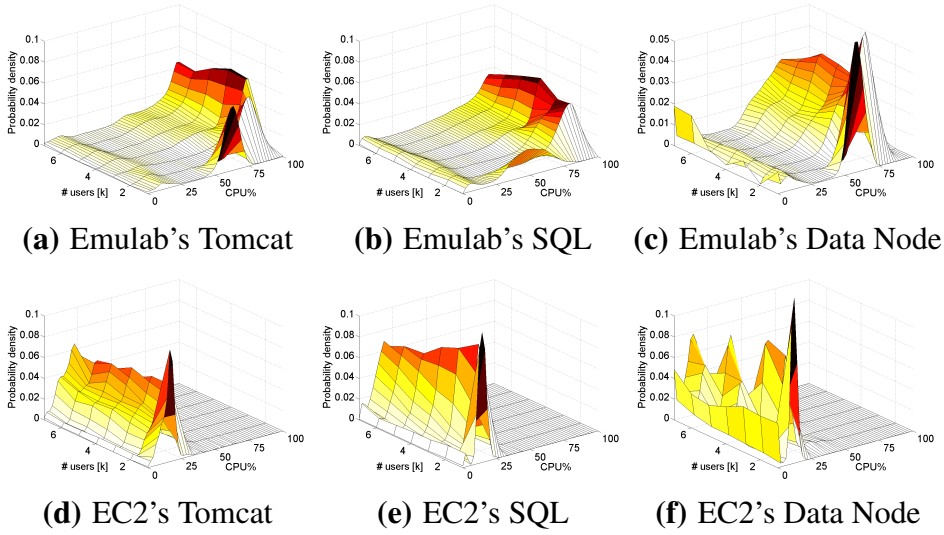
5.3.2.1 Horizontal Scale-Out (same nodes)

We used the same methodology as in Section 5.3.1, and performed a similar set of experiments in EC2. We kept as many experimental parameters untouched as possible (except platform specific parameters). Our initial expectation was to achieve better or similar results to those from Emulab since we used better hardware in EC2. Surprisingly, our assumptions were shown to be wrong when we observed quite different behavior in EC2. As shown in Figure 28(a) and (b) benchmark application scales well on both Emulab and Open Cirrus, increasing the number of Data nodes gives better performance; however, in EC2 increasing the number of Data nodes reduces the performance. As shown in Figure 28(c) scaling from `1-2-2-2-large` to `1-2-2-4-large` significantly reduced the throughput and moving from `1-2-2-4-large` to `1-2-2-8-large` further reduces the throughput. More specifically, the scalable application on Emulab and Open Cirrus shows poor scalability on EC2.

The observed response time values are shown in Figure 29(b), as shown in the figure after workload of 2000 users response time increase significantly. In addition, `1-2-2-2-large` configuration shows the lowest response time values and `1-2-2-8-large` shows highest response time values amongst three configurations. These response time values and their behavior support the observed throughout characteristics where `1-2-2-2-large` shows the best performance among the three. In fact, as compared to Emulab's response time values (Figure 29(a)) EC2 shows much higher response times(around two times higher). As mentioned below, Emulab's response time behavior was able to explain using the obvious resource monitoring data (e.g., CPU), in contrast, EC2 shows very low resource utilizations (far below saturation), yet considerably less average throughput values.

Our analysis of system monitoring data in EC2 shows interesting results. Figure 30(a), 30(b) and 30(c) illustrate CPU utilization for both Emulab and EC2. As we have already discussed, in Emulab the key limiting factor was CPU consumption (mostly saturated). As shown in Figure 31, horizontal scalability (both up and down) causes a CPU bottleneck shift from

one tier to another or a concurrent bottleneck [217] phenomenon in several tiers (see Figure 31(a)), however in EC2 we observed a completely different utilization pattern and found that all the server types have very low CPU utilization, yet significantly less performance compared to other two clouds.



(a) Emulab's Tomcat **(b)** Emulab's SQL **(c)** Emulab's Data Node

(d) EC2's Tomcat **(e)** EC2's SQL **(f)** EC2's Data Node

Figure 32: CPU Utilization for 4-4 Configuration.

In general, multi-tier workloads are bursty, hence, average value analysis could possibly hide important information. Our own experiences show the significance of time series analysis to better understand such complex systems and to observe any abnormal characteristics. As an illustration, we have presented kernel density analysis for resource utilization data in Figure 32. When drawing density graphs, we looked at the utilization distribution for each individual workload and then combined all the workloads to get the 3D figure. As shown in the figure, for Emulab experiments, most servers show high saturation near 90-95%, thus, it confirms for Emulab CPU saturation caused the observed performance issues. In contrast, kernel density analysis for EC2 shows similar distribution but with mean CPU saturation as 30%.

5.3.2.2 *Vertical Scalability (different nodes)*

Here we present our analysis results for vertical scalability on EC2. We started our analysis with `1-2-2-2-small` on small instances (similar hardware configurations as the PC3000s on Emulab) and observed comparatively low performance (as compared to Emulab's horizontal scalability), our results are shown in Figure 33. Node level data analysis revealed that for small instances, the Data node showed a large number disk access compared to that of large instances. There are two main reasons causing this issue: first, as Guohui *et al*. [332] described, in EC2 small instances always get 40% to 50% of the physical CPU sharing (i.e., they get only 40-50% CPU cycles from what they supposed to get), and second, the differences in memory configurations (Table 4). According to MySQL cluster documentation [23], small instances does not have sufficent memory to keep the complete RUBBoS database in memory, thus, it needs to use virtual memory a lot while processing the request.

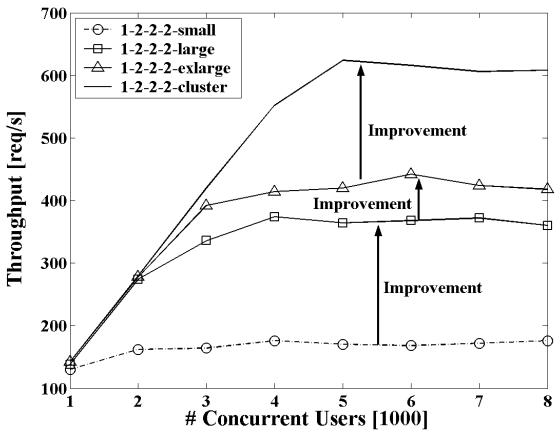


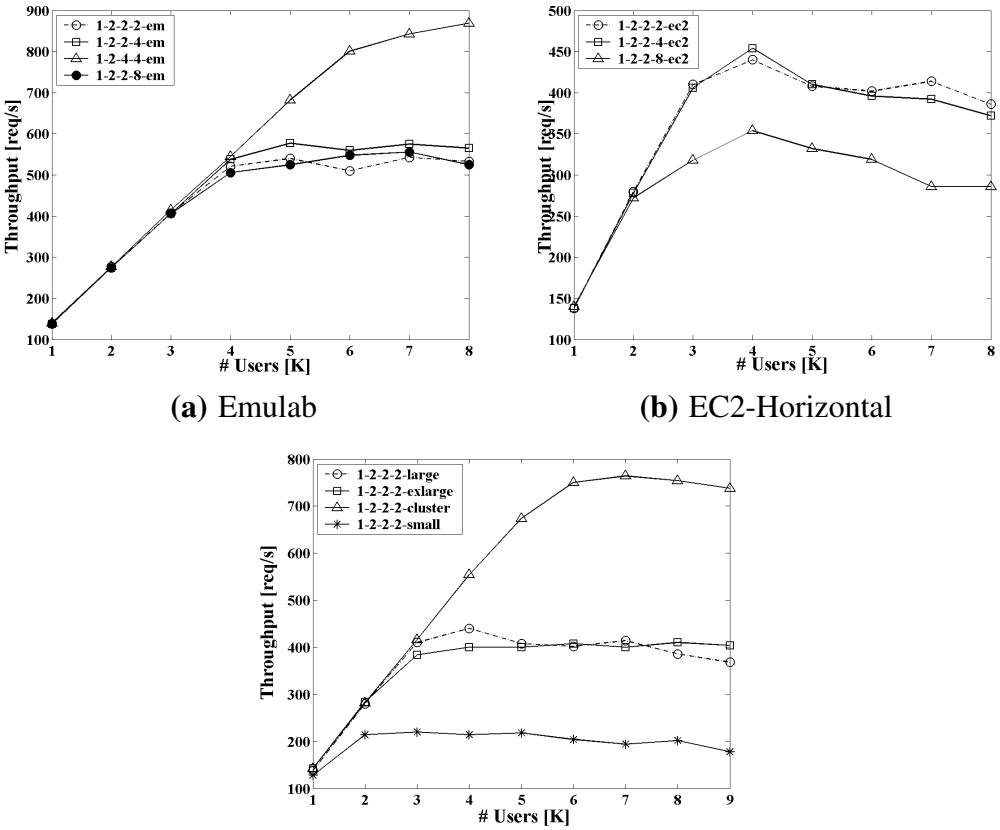
Figure 33: Throughput: EC2 Vertical Scalability

As a piratical solution to the above problem, we moved our experiments to large instances and followed the same procedure; likewise, we continued our process for the other EC2 instance types. As shown in Figure 33, EC2 shows good vertical scalability as compared to its horizontal scalability. As illustrated in the figure we achieved the highest throughput with cluster instances (cluster instances are the most powerful instances available at the time

we perform our experiments). Though, the figure shows better vertical scalability our profit analysis gives a different perspective(see section 5.5).

5.4 Read-Write Comparison

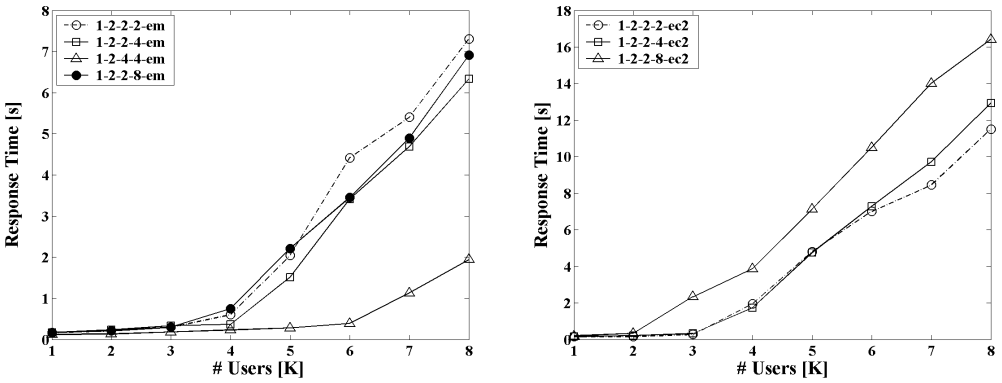
In this section we present our experiment results on read-write mix workload. In our analysis we use the same set of hardware configurations with the same software configuration settings. The main difference is the interaction types, in browse only scenarios we had 100% reads (with zero writes), in contrast, here we use 10% writes with 90% reads. In addition, browse only scanario uses only half of the interation types and read-write uses all the 24 interation types. As compared to browse only scenarios, a write request adds less work on application server and consists of at least one database update operation.



(a) Emulab **(b)** EC2-Horizontal

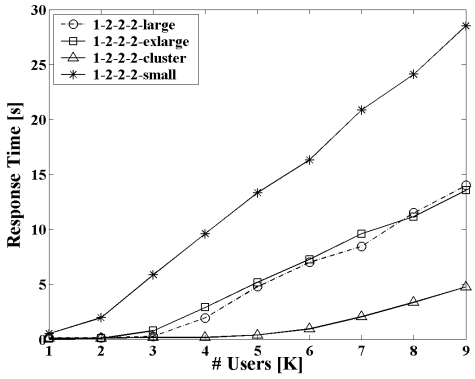
(c) EC2-Vertical

Figure 34: Throughput: Read-Write Scalability Analysis [Higher the better]



(a) Emulab

(b) EC2-Horizontal



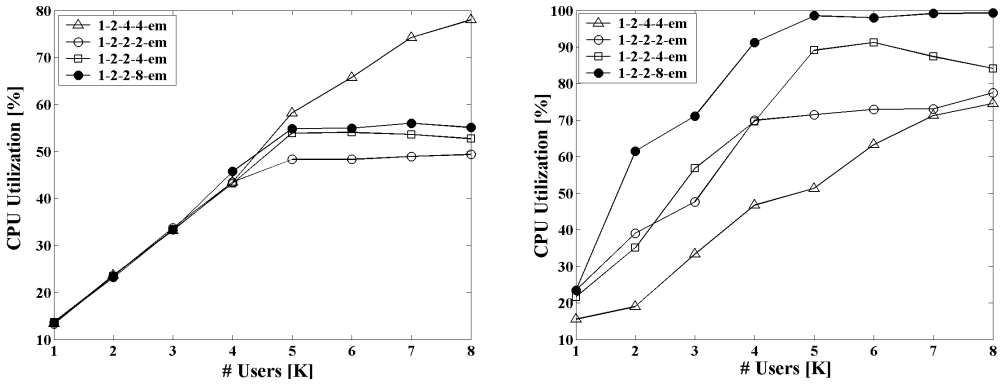
(c) EC2-Vertical

Figure 35: Average Response Time: Read-Write Scalability Analysis [Lower the better]

5.4.1 Performance on Reference Platform

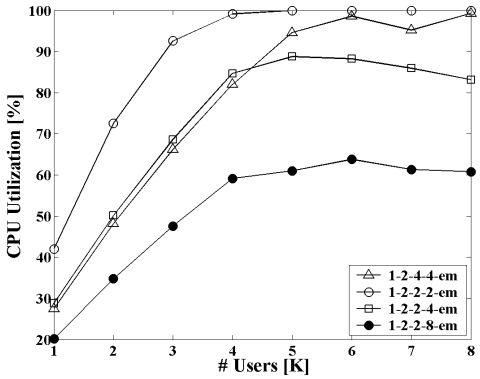
We used the same approach as in browse-only scenario (Section 5.3.1) and started with the smallest configuration and gradually increase the number of nodes in the configurations. As compared to browse-only, we observed relatively high average throughput values; in fact this is expected since we have mixture of reads and writes. Our results show that `1-2-2-2-em` and `1-2-2-4-em` configurations started to saturate at the workload of 4000. The observed throughput results are shown in Figure 34(a), in addition, calculated CPU utilization data for those configurations are shown in Figure 36. As sown in CPU utilization data, those two configurations have saturated mainly due to the Data node CPU saturation.

As in Section 5.3.1 we increased the number of Data nodes and executed a same set of experiments, and observed results are shown in Figure 34(a). As shown in the figure



(a) Tomcat CPU

(b) SQL CPU



(c) Data node CPU

Figure 36: Emulab: Read-Write Average CPU Utilization.

`1-2-4-4-em` configuration gave the best performance among all the configurations, yet it saturated at workload of 7000. Resource utilization data analysis shows that it was mainly due to CPU saturation in Data node. We added more nodes and got the `1-2-2-8-em`, yet it shows relatively low performance, in fact we observed SQL server to be the bottleneck for `1-2-2-8-em` configuration. Likewise, we evaluated many different configurations and observed throughput and response time values are shown in Figure 34(a) and Figure 35(a) respectively. The observed CPU utilization data are shown in Figure 36, and a high level view of bottlenecks and their shifting patterns are shown in Figure 31(b).

We also extended our analysis to average response time (at the client) for all the configurations discussed above. As we expected we observed relatively low average response time values as compared to that of browse only scenarios. The observed response time values are shown in Figure 35(a), as shown in the figure `1-2-4-4-em` gave the best response time

values among all the configurations. In fact, our browse only data analysis shows that the exact same configuration to have better performance.

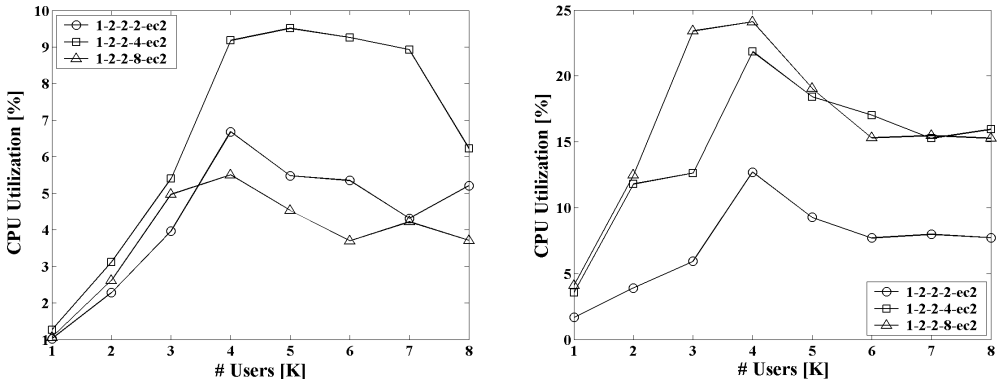
In summary, if the application consists of browse-only workload, then having many data nodes gives better performance, however, in read-write mix workload having more data nodes gives poor performance. Thus, to get better performance in average cases i.e., mixture of reads and writes one can use less number of data nodes.

5.4.2 Amazon EC2 - Horizontal Scalability

We used the same approach and extended our analysis to Amazon EC2, first we evaluated horizontal scalability. As compared to browse-only workloads, here we observed different behavior for `1-2-2-4-large` and `1-2-2-2-large` where both showed similar performance. In the case of browse only scenario, increasing number of Data nodes from 2 to 4 caused to reduce throughput significantly; however for read-write mix the behavior was different. Yet, increasing Data nodes further more causes to have a performance degradation. The observed throughput values are shown in Figure 34(b) and corresponding response time values are shown in Figure 35(b). As we discussed in browse-only scenarios, obvious resource utilization data (e.g., CPU) analysis failed to provide any useful information to find the performance issues, we observed the same phenomena in read-write workloads as well. The observed CPU utilization data for Tomcat, SQL and Data nodes are shown in Figure 37.

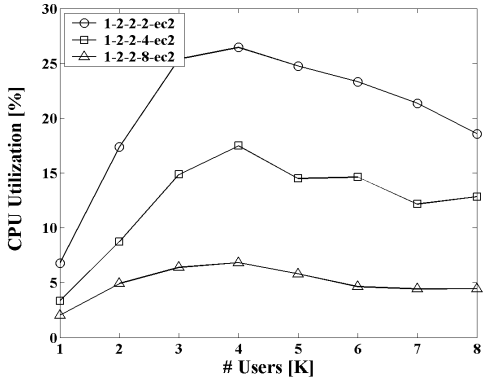
5.4.3 Amazon EC2 - Vertical Scalability

In the vertical scalability analysis we used similar configuration which we used for browse-only analysis and evaluated the scalability characteristics for different node types. The observed results are shown in Figure 34(c), as illustrated in the figure, maximum average throughput increased when moving from small instances to cluster instances. In particularly, we observed huge improvements from ex-large instances to cluster instances. Which is in fact two times higher than the values we observed for ex-large. The observed response time values are shown in Figure 35(c), as illustrated in the figure small instances shows higher



(a) Tomcat CPU

(b) SQL CPU



(c) Data node CPU

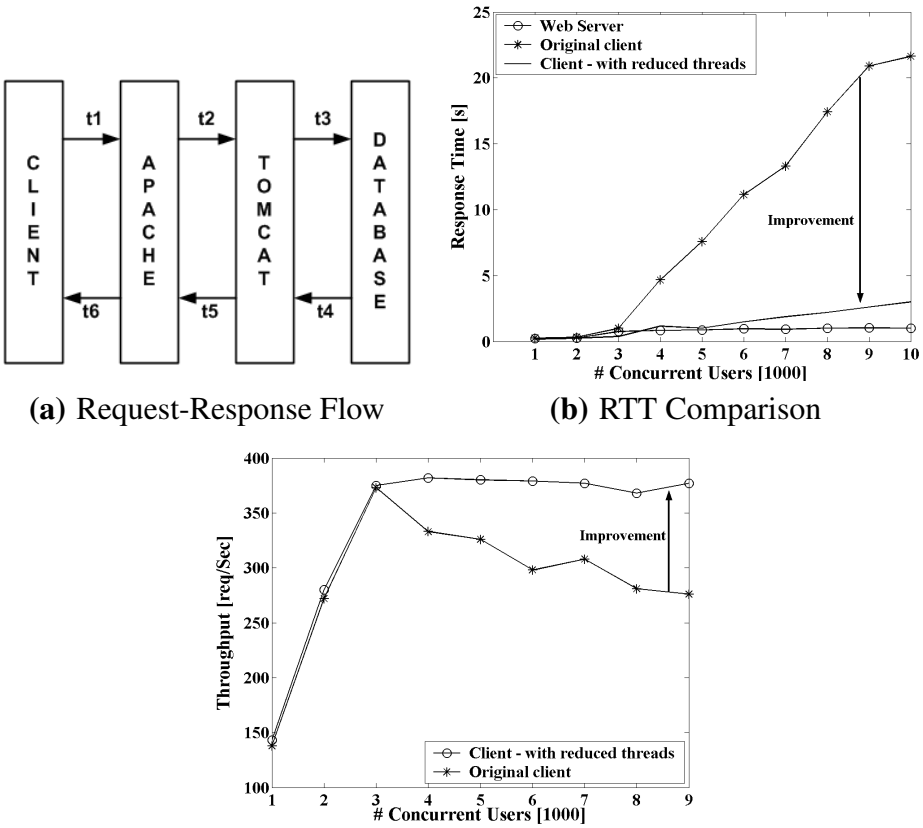
Figure 37: Amazon EC2: Read-Write Average CPU Utilization.

response time while cluster instances show very low response time. In fact, with cluster instances we were able to meet Emulab's performance numbers.

In summary, the RUBBoS benchmark shows better horizontal scalability on Emulab and Open Cirrus, better vertical scalability on EC2, and poor horizontal scalability on EC2. As illustrated in Figure 30(a), (b) and (c), the EC2 performance degradation was not caused by CPU utilization. For our experiments, we collected over 24 types of monitoring data (e.g., CPU, Memory, Disk read/write, network read/write, context switches, interrupts), and thus, we extended our analysis on other monitoring data and micro-benchmarks. We observed two key issues: overhead associated with concurrent threads (e.g., context switching and scheduling overhead) and network driver overhead (e.g., latency and queuing delays).

5.4.4 Concurrent-threading Costs in EC2

In this section, we provide the related analysis for EC2 by using the RUBBoS workload generator (a multi-thread workload driver) as a case study. Section 5.4.4.1 quantifies the context switching overhead, and Section 5.4.4.2 shows how to redesign the application to guard against the multi-threading issue.



(a) Request-Response Flow **(b)** RTT Comparison

(c) Throughput Comparison

Figure 38: Analysis for Multi-Threading Issues: (a) RUBBoS Request Path; (b) RTT comparison at Apache and Client; (c) Throughput—before and after reducing thread overhead.

Complex dependencies of multi-tier applications complicate the performance analysis process. As previously discussed (Section 5.3.2), the RUBBoS application saturated in EC2 due to effects of initially hidden resources utilization. Consequently, we extended our systematic analysis with micro benchmarks. Subsequently, we observed an interesting phenomenon when using multi-threading applications on EC2. To illustrate, we selected a RUBBoS client workload generator and measured the round trip time (RTT) for each

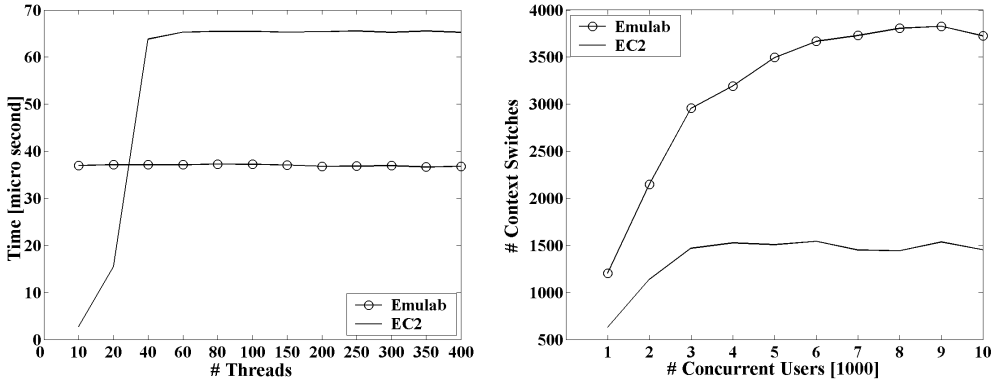
request for different workloads. We observed a sudden increase in RTT when the workload increases. Typically, these types of observations lead to assumptions of back-end server saturation; despite this, we observed the issue to be at the workload generator.

A typical request-response flow for a RUBBoS is shown in Figure 38(a). For each request, the client sees $\sum_{i=1}^{6} t_i$ (say RTT_1) as the RTT, where as RTT at Apache is $\sum_{i=2}^{5} t_i$ (say RTT_2). We recorded RTT for each individual request at the workload generator (i.e., RTT_1) and the Apache server (i.e., RTT_2). Next, we used two log files and calculated average RTT separately. The results are shown in Figure 38(b). In the figure, the original client represents the response time recorded at client and web server represents the response time recorded at Apache. As shown in the figure, for higher workloads, the calculated RTT at the client differs significantly from that of Apache. In contrast, for lower workloads, the RTT difference is negligible (e.g., workloads less than 3000). Figure 38(b) illustrates how the increase of workloads causes the recording of larger RTTs at the client.

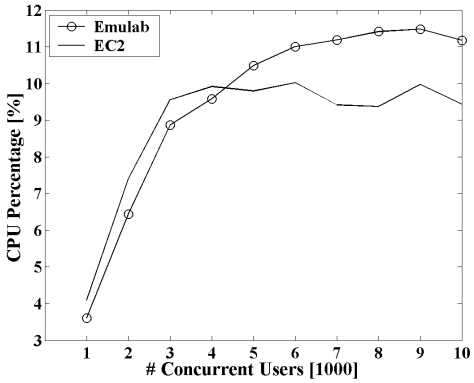
5.4.4.1 Context Switching Overhead

To further analyze the observed phenomenon we used LMbench [28]. We started with the default settings and performed all supported tests on Emulab and EC2. We found surprising results for context switching overhead. It recorded the overhead in EC2 to be twice as high as in Emulab. Figure 39(a) shows the time for a single switch when varying the number of threads. As generally accepted, measuring context switching time with high accuracy is a difficult task. Nevertheless, analysis shows that for a few number of threads (< 40) EC2 takes less time for a switch compared to Emulab; in contrast, with a greater number of threads, EC2 takes considerably longer time.

In RUBBoS, each client gets a large number of threads which is in fact equal to the workload; therefore, context switching becomes a key issue. To illustrate the significance, we calculated the average number of context switches when running the benchmark. Figure 39(b) shows the calculated averages for both EC2 and Emulab. As shown in the figure,



(a) Time per switch (LMBench) **(b)** Context switches per second



(c) Fraction of CPU spent on switching

Figure 39: Comparative Context Switching Analysis for Emulab and EC2.

Emulab has a significantly higher number of context switches compared to that of EC2. Moreover, combining Figure 26(a), Figure 26(c) and Figure 39(b) we can observe a positive correlation between the throughput and the number of context switches.

To analyze context switching overhead from the resource perspective, we calculated the overhead as CPU percentages. In our experiments we measured the number of switches at 1s granularity (see Figure 39(b)), using LMBench we approximated the time for a single context switch (against no. of threads), and finally combining the two values we estimated the context switching overhead for an experiment cycle. Figure 39(c) illustrates the calculated overhead. As shown in the figure, for the workloads which are higher than 3000, context switching uses 10% of the CPU resources for both Emulab and EC2. Note that the overhead is much more significant in EC2 because the number of context switches and the throughput are significantly lower while having similar overhead.

5.4.4.2 *Proposed Candidate Approach*

The most trivial solution in commercial clouds is to rent more nodes so that each gets less users (threads). This helps to reduce the overhead caused by concurrent threads (see Section 5.4.4.1). While this solution is acceptable for occasional users that run an application only a few times, for long term users who run their applications often or continuously, this may not be the most economical.

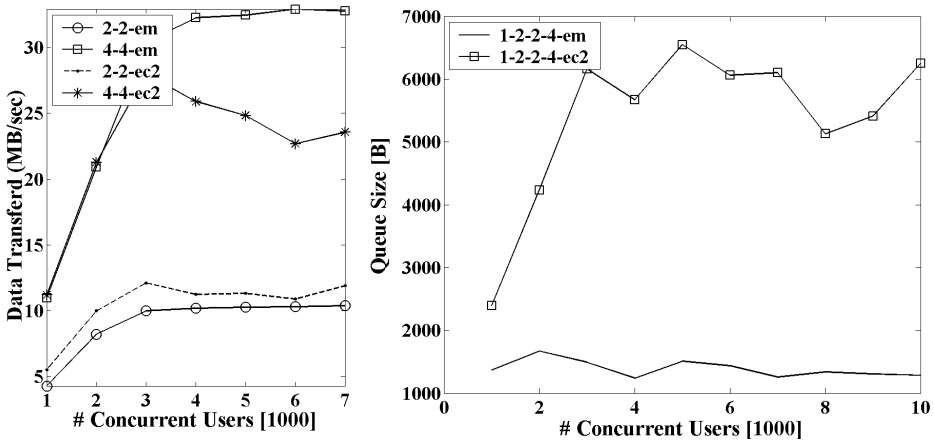
On the other hand, it is possible to redesign the application to mitigate complexities associated with clouds. For example, we can reduce the number of threads and increase the amount of work done by a single thread. We in fact followed this approach and modified the RUBBoS workload generator to generate 7 times more messages (with the same number of threads) by reducing the average client think time (e.g., reducing think time from 7s to 1s). This solution could deviate from the actual user behavior, yet it helps us to overcome the multi-threading issues.

Our solution shows a significant improvement, both in RTT (Figure 38(b)) and throughput (Figure 38(c)). As shown in the figure, with our solution we were able to bring RTT_1 much closer to RTT_2. In addition, as shown in Figure 38(c) our solution produces a significantly higher throughput. We confirmed our hypothesis by calculating the average RTT using Apache logs and the client logs. With the thread reduction, the recorded response time at client is much closer to that of the Apache. The remaining difference can be explained with the queuing effects that naturally take place between the two tiers. While this solution is practical in many cases, it depends on the nature of the application whether the desired concurrency level can still be reached in this manner.

5.4.5 Network Driver Overhead

We increased the number of client nodes to 10 and resolved the client issue, and subsequently shifted the bottleneck to the back-end. Section 5.4.5.1 shows the network traffic increase through database partitioning and subsequent network transmission overhead on EC2. In

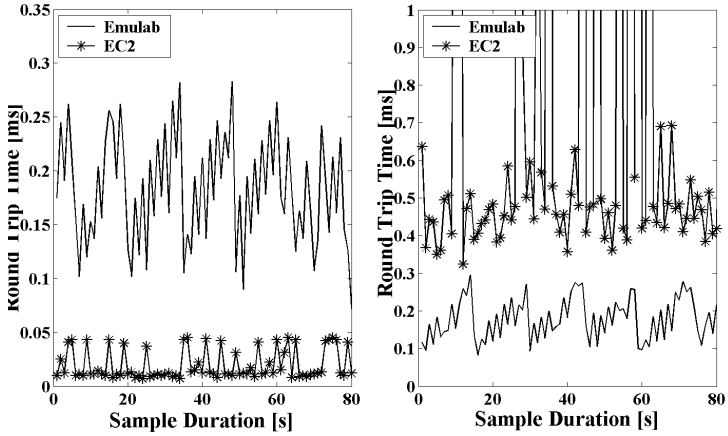
Section 5.4.5.2 we provide our solutions to achieve higher performance in public clouds.



(a) Network Traffic Recored at the Data Node

(b) Data Node Sending Queue Length

Figure 40: Comparison of Network Driver Overhead.



(a) Ping RTT without load (EC2 is lower)

(b) Ping RTT with load (Emulab is lower)

Figure 41: Analysis of Network Driver Overhead and Characteristics for Emulab and EC2.

5.4.5.1 Analysis of Network Driver Overhead

We extended our analysis to the network layer, and our results revealed two important findings. First, increasing the number of Data nodes (partitioning the database) caused a significant increase in the network traffic generated at the Data nodes (see Figure 40 (a)). Moving from 2 Data nodes to 4 Data nodes doubles the network traffic. Second, transmission

queue sizes for EC2 were significantly higher compared to that of Emulab (see Figure 40 (b)). Nevertheless, for the `1-2-2-2` configuration, both Emulab and EC2 shows similar network traffic patterns and also somewhat similar throughput values; however, `1-2-2-4` shows different behaviors both in network traffic and throughput (Figure 26(a) and (c)).

We selected `1-2-2-4-em` and `1-2-2-4-ec2` and used a number of network analysis tools to observe network behavior. Through our data, we observed a interesting behavior in the Data nodes. As shown in Figure 40(b), we observed that the sending queue size (at the network layer) of the Data node in EC2 is much higher than in Emulab. In contrast, the receiving sides of both EC2 and Emulab show similar characteristic, and both have negligible queue sizes.

In a RUBBoS application with MySQL Cluster, the Data node is the first server which generates a large amount of data in the request propagation pipeline. As shown in Figure 38(a), initially, users send an HTTP request to Apache HTTPd, and then Apache forwards the request to Tomcat. Tomcat processes the request and generates SQL statements that are sent to the SQL servers. The SQL servers send the query to the Data nodes, which then processes these queries and generates results. In this process, especially in the read-only scenario, the message sizes of the generated output are significantly higher than the incoming message (SQL query vs. a result set). Therefore, when the message rate is high, the Data node generates more data; however, when there is a problem at the transmitting side, the Data node cannot send the data as required, which then results in a long queue at the network buffers. Next, in the other tiers (e.g., Tomcat and Apache HTTPd), the connections start to wait as well. Therefore, the overall message transmission rate reduces. Eventually this behavior affects the entire system performance.

To further analyze observed hypothesis, we used NetPIPE [18] and measured the achievable bandwidth. NetPIPE is a protocol-independent performance tool that visually represents the network performance under a variety of conditions. Our results show that the two systems have similar bandwidth, thus, confirming that the sending buffer issue is not caused

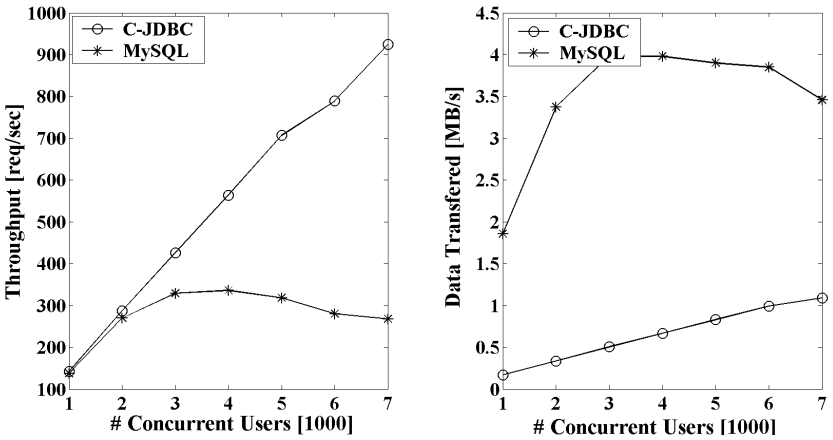
by network bandwidth. We then used the `Unix Ping (ICMP)` program and measured the RTT between the two Data nodes. First, we evaluated ping RTT without generating any application load (i.e., without generating any RUBBoS client requests), thus, making `ping` program the only process running on both the nodes. Second, we evaluated RTT while generating a representative application load (i.e., while running RUBBoS). We observed a very interesting behavior. When there is no load on the node, EC2 shows significantly less RTT compared (approximately 20 times less) to Emulab (see Figure 41(a)). In contrast, when running with the RUBBoS workload, EC2 shows very large RTT that varies between 0.4 ms to 30ms (see Figure 41(b)), but Emulab shows similar results for both cases (average of 0.2 ms). The first scenario simply means that the two nodes have good network connectivity as compared to that of Emulab; however, in EC2 when the load on the network driver increases, RTT goes up significantly whereas Emulab maintains the same RTT regardless of the load. This provides additional evidence to confirm network driver overhead in EC2 and potential performance degradation.

5.4.5.2 Proposed Candidate Approach

As we discussed in Section 5.4.5, due to this network overhead, the scalable software system is unable to scale in EC2. In general, MySQL Cluster is a sophisticated but heavily used database middleware in industry. It has a number of advantages including availability, reliability, and scalability compared to other alternatives. Unfortunately, due to higher network traffic generation of MySQL cluster, the complete application shows poor scalability.

To illustrate the significance and to provide an alternate solution, we used the C-JDBC [105] middleware and performed the same set of experiments as in Section 5.3.1 on EC2. The observed throughput values are shown in Figure 42(a). As shown in the figure C-JDBC shows very good scalability and achieves very high throughput. Next, we measured the amount of data generated at the database tier by the two approaches, and our results are shown in Figure 42(b). As demonstrated in the figure, C-JDBC generates a significantly

smaller amount of data compared to MySQL Cluster. Consequently, the middleware results in lower pressure on the network, which causes better performance. In contrast, MySQL Cluster produced a large network traffic and higher pressure on the network driver.



(a) Throughput improvement **(b)** Database tier—total traffic

Figure 42: Performance Comparison C-JDBC vs. MySQL Cluster.

5.5 Profit Analysis

We inserted the experimental data into a simple profit model to analyze the economical implications of observed findings. More precisely, we used a previously developed profit model by combining the application provider's revenue and the infrastructure cost [213]. Profit can be defined as provider revenue minus the infrastructure cost for providing the service. In commercial clouds it is straightforward to define the cost model for infrastructure since commercial clouds provide such a charging model per definition. In this case study, we used the EC2 pricing scheme to calculate the running cost of the application. In contrast, finding a realistic provider's revenue model is a much more complex task. We assume the the SLA document contains information of the the provider's revenue model, which helps to determine the earnings of the provider for SLA compliance as well as the penalties in case of agreement violations.

We have previously defined the provider's revenue as a function of average response time [213]. Here we simply apply our model to the configurations discussed in the chapter

Table 6: Profit Analysis: SLA Model

Response Time Interval	Revenue/Penalty
[0s, 1s]	0.0033 cent
(1s, 2s]	0.0027 cent
(2s, 3s]	0.0020 cent
(3s, 4s]	0.0013 cent
(4s, 5s]	0.0007 cent
≥ 5s	-0.0033 cent

to get an economical perspective. In our model we calculated the service provides revenue/penalty for browse-only and read-write separately. Concretely, for each hardware configuration we run the experiments for 10 hours with different workloads (1000 to 8000) and then categorized each individual request based on the end-to-end response time. We used six categories as shown in Table 6 and then we calculated number of requests in each category. Finally, we used the associated SLA model to calculated the revenue/penalty values for each category and our results are shown in Table 7. We used Amazon EC2 pricing scheme for on demand instances to calculate infrastructure cost, where we simply used the number and types of nodes and then multiply by the EC2 hourly charge. As mentioned before we executed our experiments for 10 hours, so infrastructure cost shown in Table 7 represent the infrastructure cost for complete experiment cycle.

Table 7: Profit Analysis: Configuration Cost vs. Profit/Loss

Configuration	Infra Cost ($)	Browse-Only		Read-Write	
		Revenue /Penalty	Profit /Loss ($)	Revenue /Penalty	Profit /Loss
1-2-2-2-cluster	40.40	127.89	87.49	168.02	127.62
1-2-2-4-exlarge	68	72.80	4.80	136.32	68.32
1-2-2-2-exlarge	54.40	56.24	1.84	66.90	12.50
1-2-2-2-large	27.20	53.96	26.76	76.32	49.12
1-2-2-4-large	34	32.34	-1.66	68.28	34.28
1-2-2-2-small	6.80	5.35	-1.45	20.11	13.30
1-2-2-4-small	8.50	2.46	-6.04	28.00	19.50

The calculated revenue/penalty and profit/loss values are shown in Table 7. As shown

in the table for browse-only workloads use of small instance results in huge lost (in a long run). In addition, as we previously discussed when using large instances to deploy the application, it is much profitable to use less number of Data nodes, as shown in the table `1-2-2-4-large` results in $1.66 loss while `1-2-2-2-large` gives $26.76 profit. More importantly, `1-2-2-2-large` is much profitable than both `1-2-2-2-exlarge` and `1-2-2-4-exlarge`, so it is necessary to understand application well before choosing the instance type. Among all the configurations, `1-2-2-2-cluster` configuration gives highest profit. We observed somewhat different results for read-write scenarios, as shown in the table each hardware configuration seems to be profitable and `1-2-2-2-cluster` gave the highest profit. Even with read-write `1-2-2-2-large` is much profitable than `1-2-2-2-exlarge`. As shown in the table, selecting the instance types and number of nodes are greatly affected by the application and its transaction types. Nevertheless, `1-2-2-2-cluster` gives highest profit regardless of the invocation model.

5.6 Related Work

Traditionally, performance analysis in IT systems builds models based on expert knowledge and uses a small set of experimental data to parameterize them [172, 204, 344]. The most popular representative of such models is queuing theory. Queuing networks have been widely applied in many performance prediction methodologies [298, 310]. These approaches are often constrained because of their rigid assumptions when handling multi-tier systems due to the complex dependencies. As an illustration of significant characteristics that are hard to capture with traditional analysis, consider the significance of context switching—as shown in our work—towards system performance when a large number of threads is involved.

While the increasing popularity of cloud computing has spawned very interesting research on modern clouds, to the best of our knowledge, no previous work has evaluated IaaS clouds using complex multi-tier systems to find performance and scalability issues. Jim et

al. presented a method for achieving optimization in clouds by using performance models in the development, deployment, and operation of applications that run in the cloud [202]. They illustrated the architecture of the cloud, the services offered by the cloud to support optimization, and the methodology used by developers to enable runtime optimization of the clouds. Thomas et al. analyzed the cloud for fundamental risks that may arise from sharing physical infrastructure between mutually distrustful users, even when their actions are isolated through machine virtualization [267]. Ward et al. discussed the issues and risk associated with migrating workloads to clouds, and the authors also proposed an automated framework for "smooth" migration to cloud [338]. In contrast, we studied the potential scalability and performance issues after migration.

Mayur et al. evaluated Amazon S3 as a black box and formulated recommendations for integrating S3 with science applications and for designing future storage utilities targeting this class of applications [249]. They discussed how costs can be reduced by exploiting data usage and application characteristics to improve performance and, more importantly, by introducing user managed collaborative caching in the system. Despite the apparent differences, this work has some important similarities to our work: first, both approaches evaluate commercial clouds; second, both papers use real applications and not simulations; third, both approaches derive recommendations to address the uncovered issues.

Guohui et al. have deeply analyzed the network overhead on EC2 and presented a quantitative study on end-to-end network performance among different EC2 instances [332]. Furthermore, Walker has presented a performance analysis and shown that a performance gap exists between performing HPC computations on a traditional scientific cluster and on an EC2 provisioned scientific clusters [331]. Similar to our findings, his experiments also revealed the network as one of the key overhead sources.

As part of our analysis, we have shown the significance of virtualization and associated network overhead. Padma et al. have looked at both the transmit and receive aspects of this workload. Concretely, they analyzed virtualization overheads and found that in both cases

(i.e., transmit and receive) the limitations of VM scaling are due to dom0 bottlenecks for servicing interrupts [58]. Koh et al. also analyzed the I/O overhead caused by virtualization and proposed an outsourcing approach to improve the performance [193].

5.7 Summary

In this chapter we presented an experimental analysis of performance and scalability when multi-tier applications are migrated to IaaS clouds. We employed the RUBBoS benchmark to measure the performance on Emulab, Open Cirus, and EC2. Especially the comparison of EC2 and Emulab yielded some surprising results. In fact, the best-performing configuration in Emulab became the worst-performing configuration in EC2 due to a combination of several factors. Moreover, in EC2 the network sending buffers limited the overall system performance. For computation of intransitive workloads, a higher number of concurrent threads performed better in EC2 while for network based workloads, high threading numbers in EC2 showed a significantly lower performance. Our data also exemplified the significance of context switching overheads. We provided a set of suitable candidate solutions to overcome the observed performance problems.

More generally, this work enhances the understanding of the risks and rewards when migrating multi-tier application workloads into clouds and shows that clouds will require a variety of further experimental analysis to be fully understood and accepted as a mature technological alternative. In addition, experiment results show the neediness of refactoring multi-tier applications before they are deployed on commercial clouds especially when one has to satisfy SLAs.

CHAPTER VI

VARIATIONS IN APPLICATION PERFORMANCE AND SCALABILITY IN MODERN HYPERVISORS

Through large-scale experimental analysis with the Expertus automated experiment management infrastructure, we aim to show the impact different hypervisors have on two types multi-tier workloads. Concretely, we use a commercial hypervisor, Xen, and KVM to run two representative n-tier macro-benchmarks (Cloudstone and RUBBoS). We observe significant performance variations among the three hypervisors. For instance, Xen outperforms the commercial hypervisor by 75% on the read-write RUBBoS workload, and the commercial hypervisor outperforms Xen by over 10% on the Cloudstone workload. The benefits of hypervisors are profound, and as shown in this chapter, not all hypervisors are equal. Our experiments have shown that Xen is a good deployment choice for RUBBoS workloads, which consume a lot of CPU and disk; whereas, the commercial workload is better for Cloudstone workloads. The workload performance differences shown among the hypervisors suggest that further analysis and consideration of hypervisors is needed before deploying applications into cloud environments.

6.1 Introduction

Cloud computing has become possible due to a large number of underlying technologies, services, infrastructures and business models. Among them, virtualization is the key enabler i.e., the process of abstracting computing resources such that multiple operating system and application images can share a single physical server (hardware and system resources). This is typically achieved through a hypervisor or a Virtual Machine Monitor (VMM), which lies in between the hardware and the operating system allowing multiple operating systems,

which are termed as "guests", to run concurrently on a host computer. The hypervisor presents a virtual operating platform to the guest operating systems, and it manages the execution of the guest operating systems. Many different hypervisors (both open source and commercial) exist today, each with their own advantages and disadvantages. Regardless of which one, they introduce a large number of new and challenging research questions.

In this chapter, we use a distinct set of representative workloads to try to provide an in-depth performance and scalability analysis of two commonly used open source hypervisors (i.e., XEN and KVM) and a commercial hypervisor (referred to as CVM). Concretely, we selected two representative n-tier macro benchmark applications (RUBBoS [25] and Cloudstone [294]). These experiments are generated and executed automatically using the automated framework discussed in Chapter 3. We use automated experiment management tools, which setup, execute, monitor, and analyze large-scale application deployment scenarios. Through our experiments, we aim to answer following questions:

- How different hypervisors respond to different workloads?
- How well different hypervisors respond to increasing workload?

Relying on large-scale experiment measurements, we observed significant performance variations among three hypervisors. More precisely, Xen outperforms the commercial hypervisor by 75% on the read-write RUBBoS workload and the commercial hypervisor outperforms Xen by over 10% on the Cloudstone workload. Obviously, the results reported in this chapter are just a snapshot of the current state-of-the-art. The contribution is to establish a framework that allows vendors to gradually improve their service and allows users to compare products. More generally, this study shows that virtualization technologies are a relatively immature technology, and a significant amount of additional experimental analysis is necessary in order to establish their suitability for mission-critical applications.

The remainder of this chapter is structured as follows. Section 6.2 provides an overview

of the benchmark application, the experimental setup, and the tools used during the experiments. In Section 6.3 we study the performance difference for three hypervisors using n-tier workloads and in Section 6.4 we study how performance varies when increasing the number of virtual cores given to the virtual machines. Related work is summarized in Section 6.5, and Section 6.6 summarizes the chapter.

6.2 Background

This section provides a brief introduction to three types of hypervisor we used for our experiments. Introduce benchmark applications and reason for selecting them. Experiment platform, software stack, versions. How we run the experiments - automated framework, data collection and analysis. The benchmarks applications we used for the study are detailed in Appendix A.1 and Appendix A.2 respectively, and a brief description to the database middleware is given in Appendix A.3.

6.2.1 Kernel-based Virtual Machine (KVM)

KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, `kvm.ko`, which provides the core virtualization infrastructure, and a processor specific module, `kvm-intel.ko` or `kvm-amd.ko`. KVM requires a modified version of `qemu`, a well-known virtualization software. KVM supports a large number of x86 and x86_64 architecture guest operating systems, including Windows, Linux and FreeBSD.

6.2.2 XEN

Xen is an open source virtual machine monitor which is structured with the Xen hypervisor as the lowest and most privileged layer. Above this layer are located one or more guest operating systems, which the hypervisor schedules across the physical CPUs. Xen can work both in para-virtualized or HVM mode; in the first the guest operating system must be modified to be executed. Through paravirtualization, Xen can achieve very high performance. The

HVM mode offers new instructions to support direct calls by a para-virtualized guest/driver into the hypervisor, typically used for I/O or other so-called hyper calls.

6.2.3 CVM - Commercial Virtual Machine Monitor

There are many different commercial virtualization technologies each of which aiming to provide similar or better performance and features as open source hypervisors. However, most of them have licensing and copy rights issues which prevent publications of performance and comparison data. Thus, in our study we selected one of the commonly used commercial hypervisor (hereafter CVM) and compare the performance with both XEN and KVM.

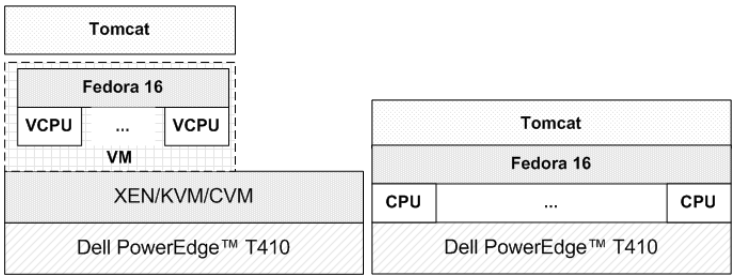
Table 8: Hardware and VM Configuration.

Physical Machine	
Processor	2 X Intel(R) Xeon(R) @ 2.27GHz (Quad)
Memory	16GB
Cache	2 X 8MB
Operating System	Debian Linux/Fedora 16
Virtual Machine	
Virtual CPUs	[Vary from 1 to 8]
Memory	4GB
Operating System	Fedora 16

6.2.4 Overview of Experiments

The experiments discussed in this chapter were run in three hypervisors and native hardware using number of different macro and micro benchmarks. To focus this study on the differences among three platforms, we allocated each server to a dedicated physical node. Specifically, the sharing of physical resources (e.g., CPU) in a node is among the threads within each server, not between servers. For each concrete hardware configuration we observed the most appropriate software configuration (e.g., number of threads and thread pool sizes) by using the approach mentioned in our previous work [334]. Table 8 provides

an overview of physical and virtual configurations we use for our experiments. To have a fair judgment, we selected exact same virtual machine configurations across all three hypervisors (e.g., OS, Memory, disk). A typical application deployment in a virtual machine and in a physical machine is shown in Figure 43.



(a) VM Deployment **(b)** Native Deployment

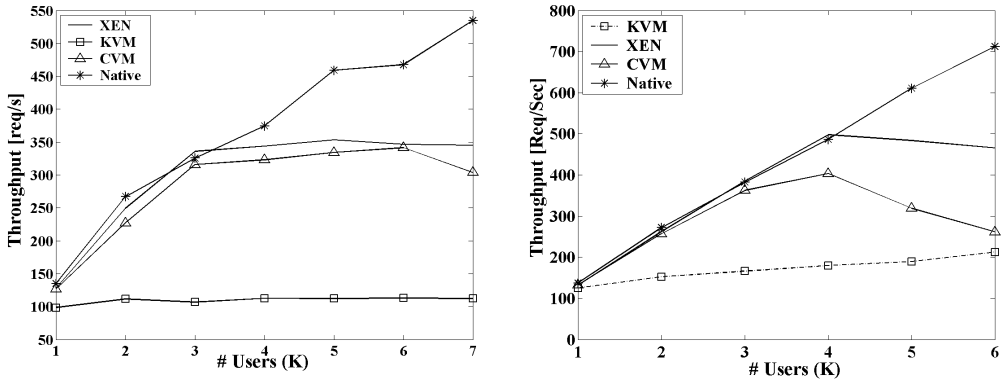
Figure 43: Illustration of Tomcat Deployment: Virtual Machine vs. Native Hardware.

Each of our macro-benchmark experiment trials consisted of three periods: 8-minute ramp-up, a 12-minute run period, and a 30-second ramp-down. Performance measurements (e.g., CPU or network bandwidth utilization) were taken during the Run period using lightweight system monitoring utilities (e.g., `dstat` and `sar`) with a granularity of one second. While the macro-benchmark data gave us the interesting bottleneck phenomena described in this chapter, we often need more detailed data to confirm concrete hypotheses on the non-trivial causes of such phenomena.

6.3 Study with n-Tier workloads

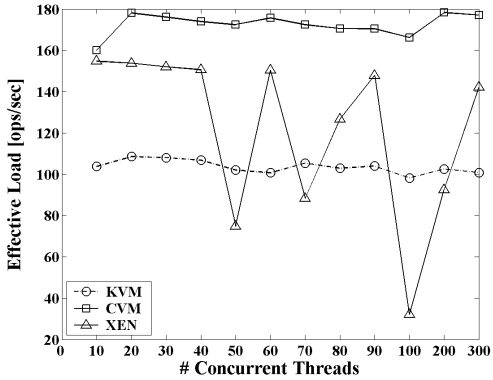
In this section we discuss and compare the observed performance results on three hypervisors and on native hardware by using two n-tier benchmark applications. First, we discuss performance using RUBBoS benchmark, and then performance with CloudStone. We also explain observed performance phenomena through detailed resource utilization data.

The goal of our study is to obtain a fair performance comparison among three hyperviosrs and to validate the results with native hardware. Hence, for all our experiments (regardless of the hypervisor) we selected an identical virtual machine configuration (i.e., CPU, Memory,



(a) RUBBOS - Read Only

(b) RUBBOS - Read Write



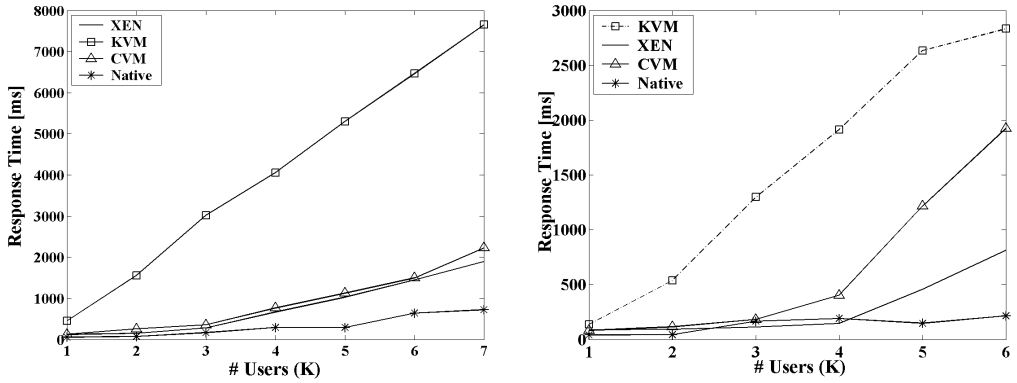
(c) CloudStone

Figure 44: Throughput Comparison for RUBBoS Read-only, RUBBoS Read-write and Cloudstone. Among Hypervisors, XEN Outperforms while KVM is the Lowest.

Disk and Operating System) and subjected to an identical workload. More specifically, for all the hypervisor studies we generate workloads from a fixed set of physical machines. Each of our experiments, we have used dedicated deployment i.e., only one VM per physical host. We used that approach to reduce the interference from other virtual machines. In addition, to observe the performance characteristics on native hardware, we disable six cores and brought it to similar configuration as virtual machines, and then repeated the same procedure on native hardware as well.

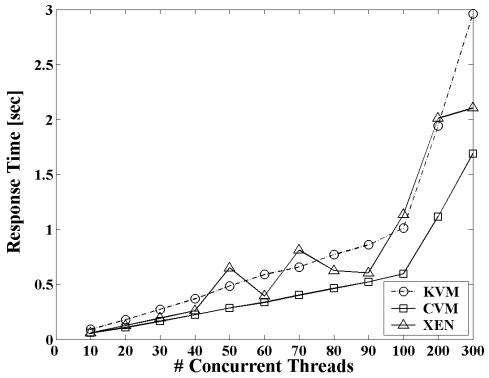
6.3.1 RUBBoS - Read Only

In our study we first deployed RUBBoS application with MySQL cluster database middleware and executed workload from 1000 concurrent users to 7000 concurrent users (with step



(a) RUBBOS - Read Only

(b) RUBBOS - Read Write



(c) CloudStone

Figure 45: Response Time Comparison for RUBBoS Read-only, RUBBoS Read-write and Cloudstone.

of 1000 users) on all the three hypervisors. The observed throughput results and response time values are shown in Figure 44(a) and Figure 45(a), respectively.

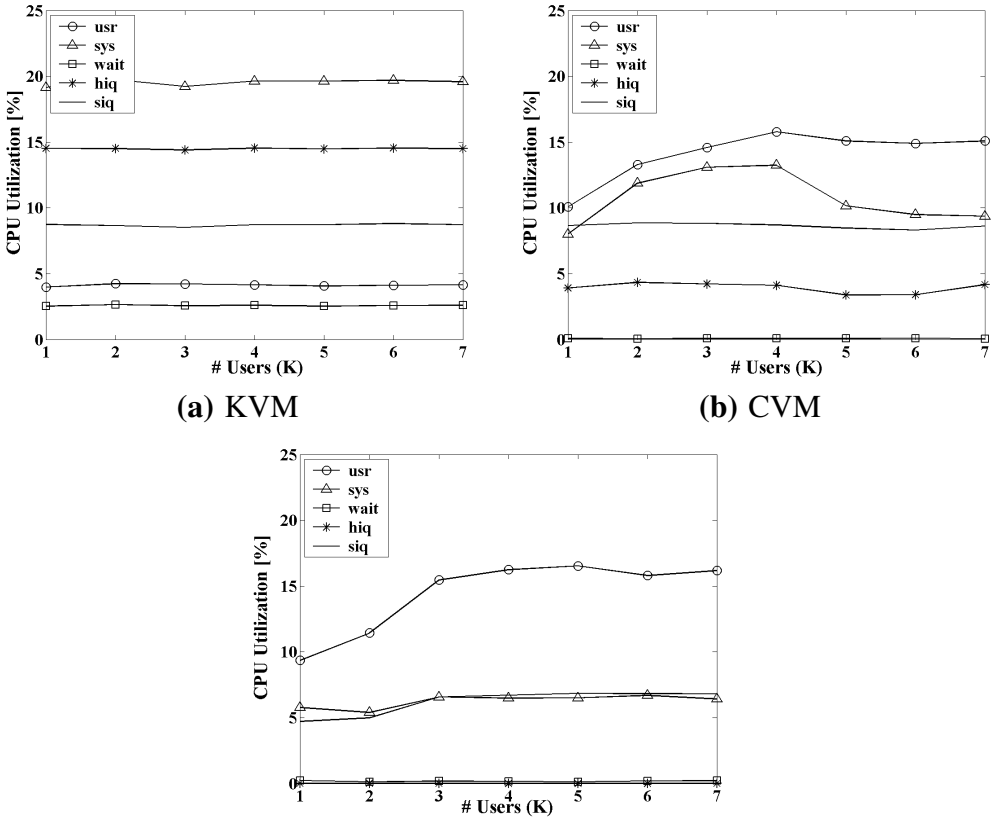
As illustrated in two figures (Figure 44(a) and Figure 45(a)), native hardware gave the best performance, which is in fact the expected behavior. Among three hypervisors, XEN and CVM show similar performance characteristics (both response time and throughput), in contrast, KVM shows significantly less throughput and comparatively high response time. As shown in the Figure 44(a), throughput on KVM remains constant against increasing workload; this is a clear indication of system saturation. Additionally, both XEN and CVM started to saturate at the workload of 3000 concurrent users while native hardware continued to perform better.

6.3.2 RUBBoS - Read Write

We then used the same approach as in read-only scenario and extended our experiments to read-write mix workload, the goal was to introduce significant amount of disk writes. Due to the nature of transactions types in RUBBoS, we expected to see relatively higher throughputs values compared to that of read-only. The observed results (i.e., throughput and response time values) are shown Figure 44(b) and Figure 45(b), respectively. We observed a noticeable difference for CVM, where for read-only workloads both CVM and XEN show similar results (throughput and response time), in contrast, with read-write XEN shows much better performance compared to CVM. For example, the throughput difference between CVM and XEN for 6000 workload was 78%, and the same workload CVM show 200% higher response time than XEN. KVM behavior reamins unchaned when moving from read-only to read-write mix workloads, and in fact both the cases it shows very poor performance.

To explain the observed performance differences in read-only and read-write scenarios, we extended our analysis to 24 different resource utilization data (e.g., CPU, Memory, IO, interrupts, context switches) on all the servers. Due to the space constraints, we have selected one with most interesting results for our discussion. In RUBBoS deployment (see Figure 61), data node is the last component in the request pipeline and does most disk/network read and writes. Hence, its resource utilization determines overall system performance. Thus, we looked at CPU utilization, in particularly CPU breakdowns (i.e., `usr`, `sys`, `wait`, `siq` and `hiq`) from which we were able to explain the observed performance difference.

The observed CPU utilization for KVM, CVM and XEN for read-only workloads are illustrated in Figure 46 (a), (b) and (c), respectively. As shown in the figure, both CVM and XEN has their highest utilization for `usr` component, while KVM has its highest for `sys`. In addition, KVM spends more CPU cycles for software interrupts (`siq`), hardware interrupts (`hiq`) and IO `waits`, and CVM spends similar amount of CPU cycles for software interrupts. Noticeably, XEN spends less CPU cycles on software interrupts while (as expected) it



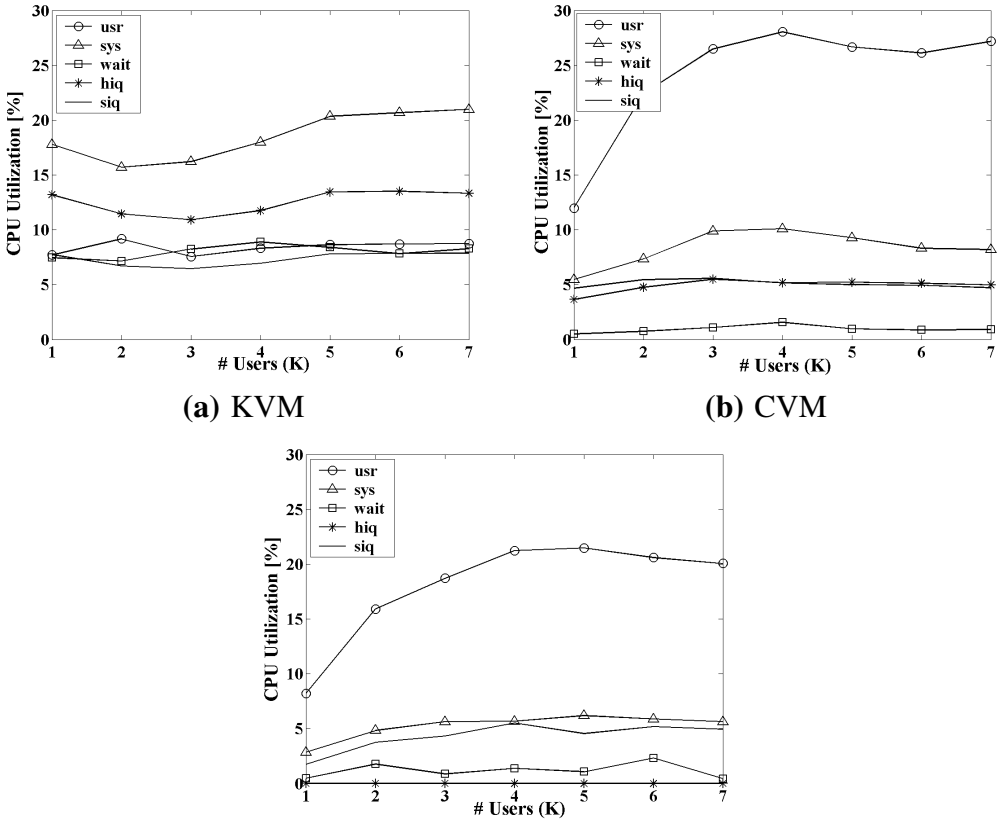
(a) KVM **(b)** CVM

(c) XEN

Figure 46: CPU Utilization - Read Only.

spends zero CPU cycles on hardware interrupts. Among all three hypervisors, KVM has considerably higher CPU `wait`, while XEN has lowest aggregated CPU utilization. As shown in Figure 44(a), both XEN and CVM gave similar throughput values, yet, CVM spends more CPU cycles to achieve the same throughput as compared to XEN.

We then extended our analysis to read-write data, our results for KVM, CVM and XEN are shown in Figure 47 (a), (b) and (c), respectively. There we observed three noticeable differences 1) CPU `wait` changes for KVM, 2) CPU `usr` behavior changes for CVM and 3) increase of utilization difference between `usr` and `sys` for both XEN and CVM. For read-only workloads KVM shows around 3% utilization for CPU `wait`, in contrast that value went to 8% for read-write workloads. The difference for XEN and CVM were negligiable. As shown in Figure 44(b), XEN gave much better perormance as compared to CVM, yet CPU utilization data for `usr` shows the opposite. Where both XEN and CVM showed



(a) KVM **(b)** CVM

(c) XEN

Figure 47: CPU Utilization - Read Write.

increase of `usr` component, while CVM has around 25% when XEN as only 20%. As shown in Figure 46(b), CVM has very small gap bwteen its `usr` and `sys` component for read-only workloads, however as shown in As shown in Figure 47(b) the difference become significantly large. Interestingly, we obsrved somewhat similar results for XEN as well.

In summary, we observed relatively less performance in KVM due to more CPU cycles have been spend for `sys` and very less CPU cycles for `usr` (i.e., to run user application), while XEN does a much better job on CPU handling. As throughput values shows, CVM behave moderately. Further analysis on interrupts handling data shows that XEN has the highest interrupts handling rate (interrupts per second) while CVM has the lowest.

6.3.3 Comparison with Cloudstone

In previous two studies we used the same benchmark application and changed only the database middleware. However, cloud platforms (a.k.a hypervisors) are used to deploy vast variety of applications. Hence, to better understand the behavior we selected CloudStone, a new benchmark with completely different workload characteristics and interaction patterns. We used the same set of virtual machines and follow similar procedures as previous cases i.e., measure the performance while increasing the workload.

One of the noticeable difference between RUBBoS and Cloudstone is their workload generators, where RUUBoS uses a closed system while Cloudstone (Rain) uses an open system. Thus, in Cloudstone it records the number of effective request (completed successfully) vs. offered workload. The observed value for effective load when increasing number of thread is shown in Figure 44(c) and observed response time values are shown in Figure 45(c). As we observed in two previous scenarios, XEN and CVM out performed KVM, yet we observed an interesting phenomenon with XEN which we will discuss later in this chapter.

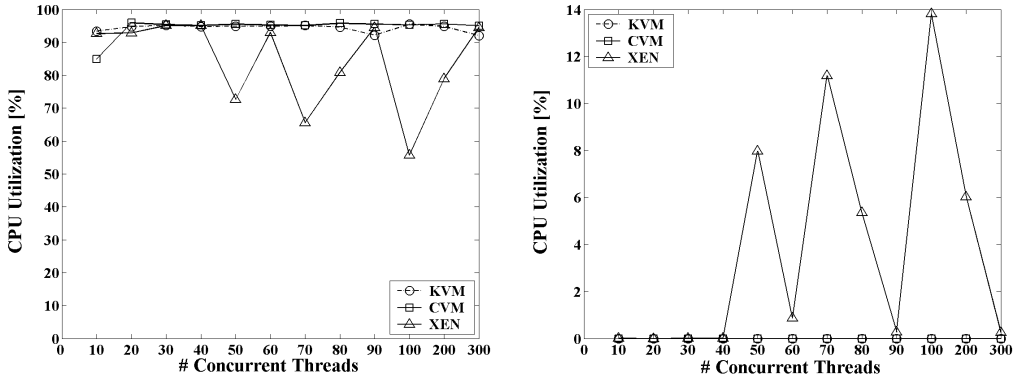
6.3.4 Cloudstone Performance Issues in XEN

As we discussed, in Figure 44(c) we observed an interesting effective load characteristics in XEN. As shown in the figure, KVM and CVM show more stable curve for the effective load. To explain the observed phenomenon we extended our analysis to different resource monitoring data (i.e., disk IO, network IO, CPU, memory, interrupts and context switches). Notably, through our analysis we observed a matching pattern in the CPU utilization, which is illustrated in Figure 48(a). Nevertheless, average CPU utilization graph alone does not provide better insight to the problem, thus, we analyzed individual CPU utilization (i.e., `sys`, `user`, `idle` etc. . .).

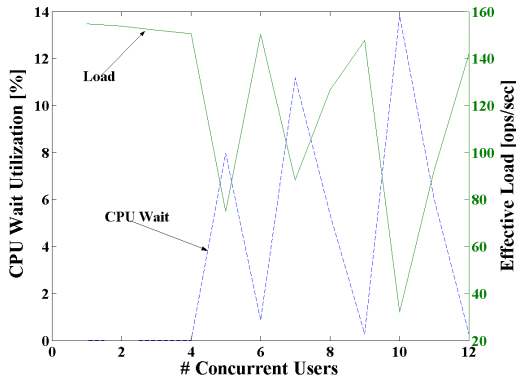
Through our analysis we observed a surprising results, where while both KVM and CVM has almost zero CPU utilization for `wait`, XEN shows higher utilizations for some workloads. We then use effective load and CPU `wait` distribution and inserted both into

a correlation model. Through the model we observed a complete negative correlation between CPU `wait` and effective load. The observed results for the correlation is shown in Figure 48(c), while Figure 48 illustrates CPU wait for three hypervisors.

In summary, performance comparison results on three hypervisors using n-tier workloads shows interesting results. More precisely, the observed performance on three hypervisors were highly sensitive to the application workloads. Nevertheless, both the open source hypervisor (i.e., XEN) and the commercial hypervisor (i.e., CVM) show similar results and outperformed KVM.



(a) CPU Utilization **(b)** CPU wait Time



(c) Negative Correlation with Load

Figure 48: CloudStone issue.

6.4 Processor Scalability

One of the biggest advantages of today's cloud is its infinite scalability; in fact there are two types of scalabilities namely `vertical` and `horizontal` scalability. In vertical scalability,

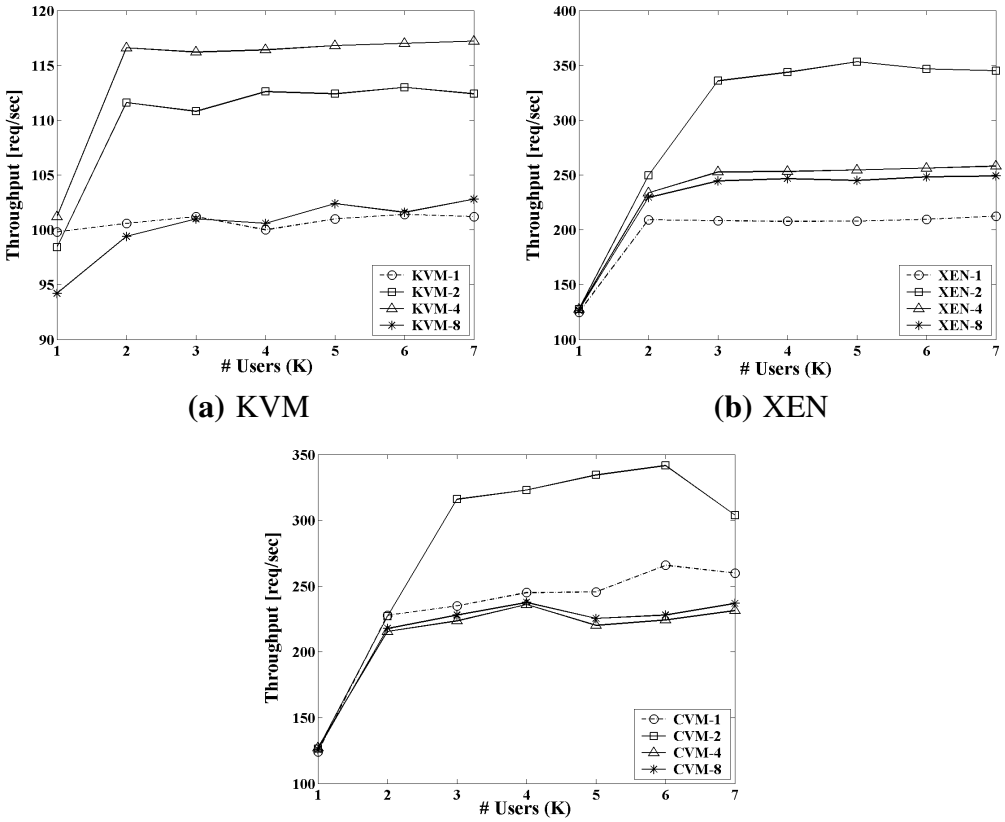
user moves from a lower configuration (e.g., 2GB) to a higher configuration (e.g., 4GB), and in contrast with horizontal scalability users moves from a less number of nodes to a large number of nodes. In our previous studies, we have experimentally evaluated the performance variations in public clouds [174]. Here we study, how three hypervisors respond to vertical scalability (i.e., increasing number of virtual cores given to the virtual machine). More concretely, in our study we used RUBBoS benchmark, and analyze its performance characteristics by changing number of VCPUs given to each virtual machine.

In our approach, we used identical VMs for all the servers (e.g., Tomcat, Apache, MySQL) and started with the smallest CPU configuration i.e., one virtual CPU and increase up to eight VCPUs (maximum number of physical cores) and measure the application performance. As shown in Figure 49 all the three hypervisors show similar characteristics when moving from `1 VCPU` to `2 VCPUs`, where we observed performance improvements. However, as shown in Figure 49, three hypervisors show different behavior when moving from `2 VCPUs` to `4 VCPUs` and then to `8 VCPUs`.

As illustrated in Figure 49(a), KVM shows performance improvement when moving from `2 VCPUs` to `4 VCPUs`, in contrast we observed performance degradation in both XEN and CVM (see Figure 49(b) and (c)). We observed a noticeable performance characteristics in KVM when moving from `4 VCPUs` to `8 VCPUs`, however XEN and CVM were shown to behave same for both `4 VCPUs` and `8 VCPUs`. The observed response time values are shown in Figure 50, and in fact response time values also show the same phenomenon. To better explain the observed phenomenon, we extended our analysis to individual CPU and CPU breakdowns (i.e., `sys`, `user`, `wait`).

6.4.1 Individual Core Analysis

To better explain the observed behavior when increasing the number of VCPUs (i.e., vertical scalability) we extended our analysis to individual CPU for each hypervisor. More specifically, we calculated the average CPU utilization for each individual VCPU and compare



(a) KVM **(b)** XEN

(c) CVM

Figure 49: Number of VCPUs vs. Average Throughput.

them with each other hypervisor. We first started with `2 VCPUs` and extended up to `8 VCPUs` i.e., to cover all the three scenarios discussed in previous section. The observed CPU utilization values for Apache server with `2 VCPUs` case for KVM, XEN and CVM are shown in Figure 51(a), (b) and (c), respectively. As shown in the figures KVM shows highest utilization and both the VCPUs have some degree of utilization. In contrast, XEN and CVM shows utilization for only one VCPU and other one remain almost zero.

The utilization pattern remain unchanged even when moving from `2 VCPUs` to `4 VCPUs`, more specifically both XEN and CVM had used only one CPU while KVM used all the four VCPUs, but only one reported to have a significantly higher utilization. The observed results are for KVM, XEN, CVM are illustrated in Figure 52(a), (b) and (c), respectively.

We observed interesting results when moving from `4 VCPUs` to `8 VCPUs`, where KVM behavior unchanged, yet, XEN and CVM show considerably different utilization patterns.

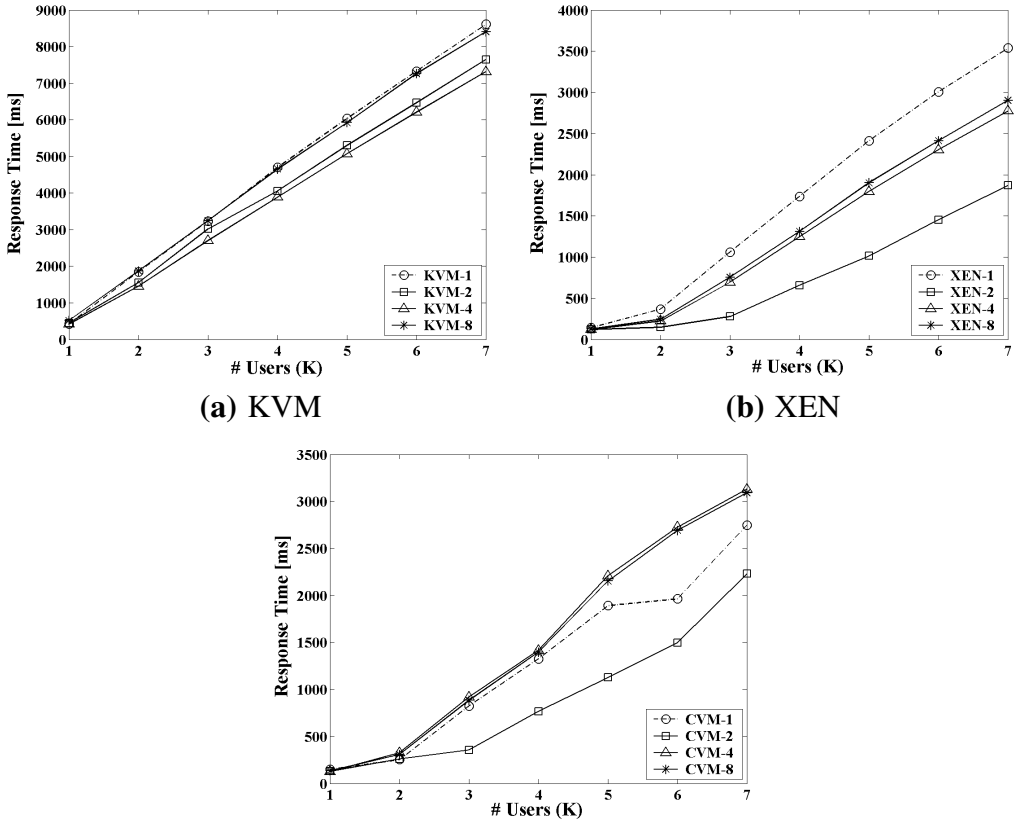
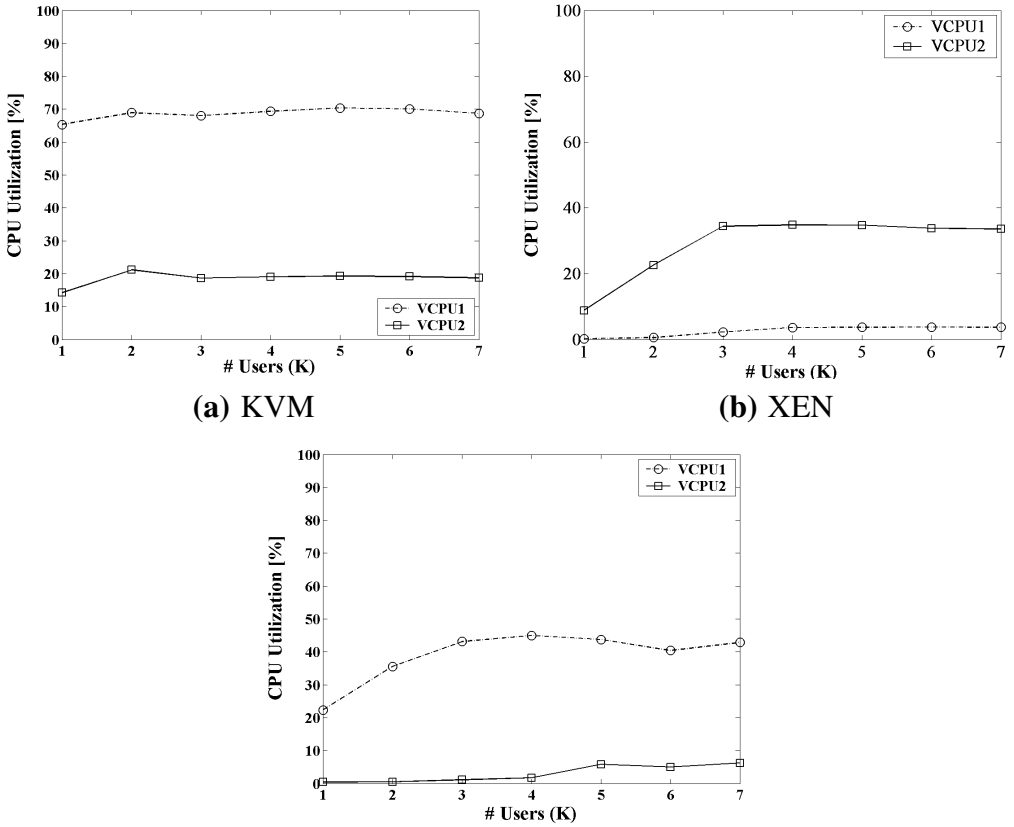


Figure 50: Number of VCPUs vs. Average Response Time.

The observed results are shown in Figure 53(a), (b) and (c). As shown in two latter figures all the cores show some degree of utilization (and high degree of fluctuation), more importantly non-uniform pattern across different workloads. These types of utilization patterns lead to have higher scheduling cost and which then results in lower performance.

Our previous studies have shown how average cannot be the average for large-scale systems [333], thus we inserted observed utilization data to a kernel density model. In our model, for each second (i.e., one second granularity) we chose maximum utilization from all the 8 CPUs as the utilization for that given second. Then we calculate max utilization for whole experiment and draw the kernel density graphs for each workload. We applied the same technique to all the three hypervisors and observed results are shown in Figure 54(a), (b) and (c). As shown in Figure 54 (a) KVM shown a `uni`-model distribution while both XEN and CVM show a `bi`-model distribution. The interesting observation is that both XEN



(a) KVM **(b)** XEN

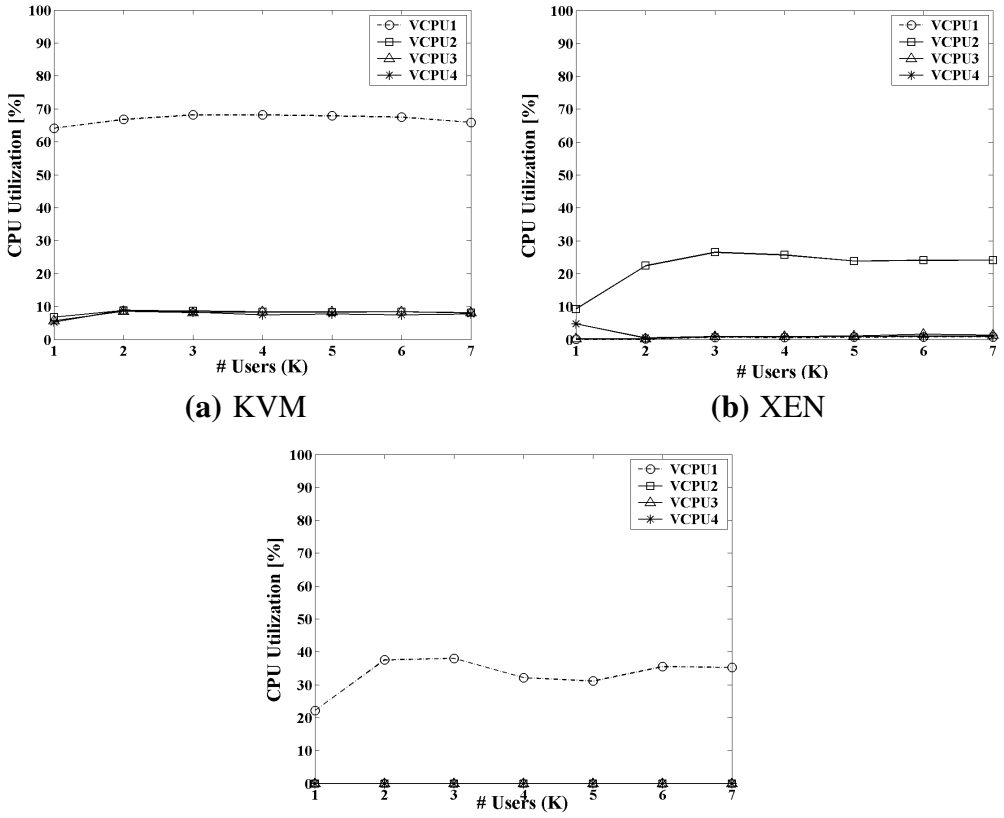
(c) CVM

Figure 51: Individual CPU Utilization for Apache (for 2 VCPUs).

and CVM seem to have at least one CPU at higher utilization level (i.e., bottlenecked) and causing whole system to have lower throughputs.

6.4.2 Analysis of CPU Breakdowns

In general, most resource monitoring tools (e.g., `sar`, `vmstat`, `dstat`, `top`) provide a way to monitor individual CPU utilization in multi-core environment as well as CPU utilization breakdown for each individual CPU. CPU breakdowns typically consists of `user`, `system`, `idle`, `software interrupts`, `hardware interrupts`, `wait` and `nice`, each of which have significant impact on different applications. In a typical experiment, CPU utilization means `(100 - idle)`, which is in fact provides a good indicator to understand the system. However, our results show that to explain performance impact of hypervisors it is required to look at the CPU at finer granularity.

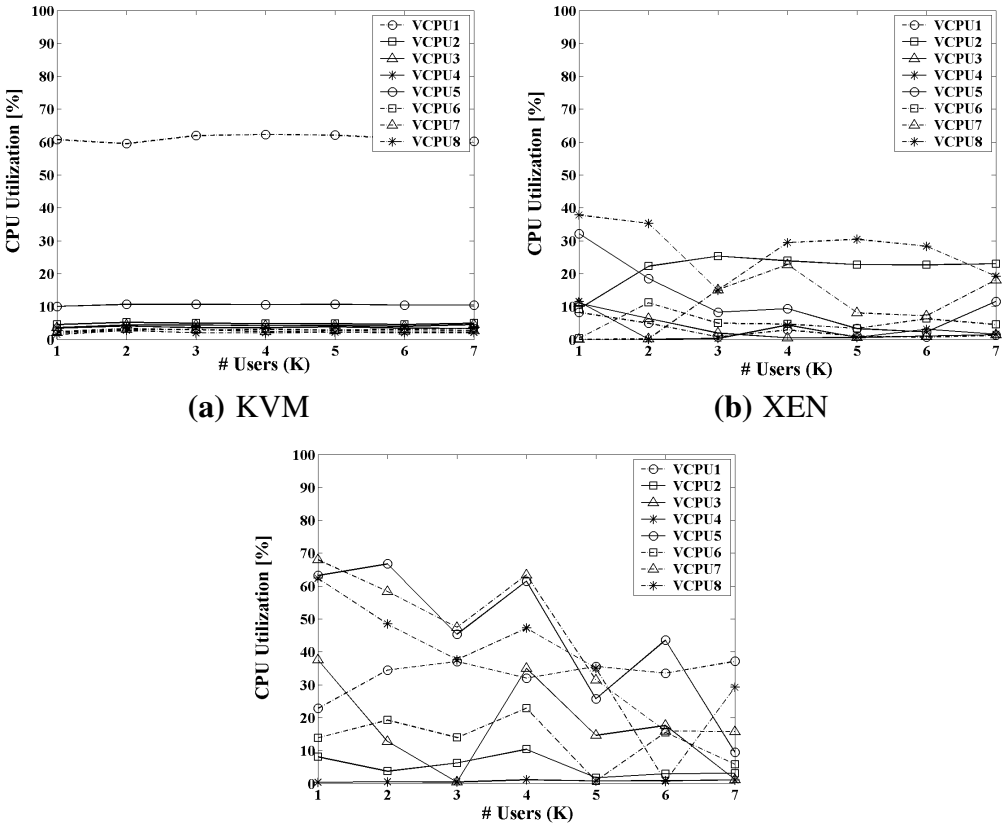


(a) KVM **(b)** XEN

(c) CVM

Figure 52: Individual CPU Utilization for Apache (for 4 VCPUs).

We used our monitoring data and analyzed the CPU utilization data for Apache server at individual components level. Our results show interesting results for hardware interrupts, software interrupts and system. The observed results for three hypervisors are shown in Figure 55. As shown in the figure KVM shows the highest utilization for all three (yet having significantly fewer throughputs), more concretely regardless of the workload KVM has over 80% utilization for these three CPU components. As expected XEN shows zero utilization for hardware interrupts. As a summary, KVM spends significant amount of CPU cycles for hardware and software interrupts which is considerably higher than other two hypervisors, and that is said be a one of the contibuotor to poor performance in KVM.



(a) KVM **(b)** XEN

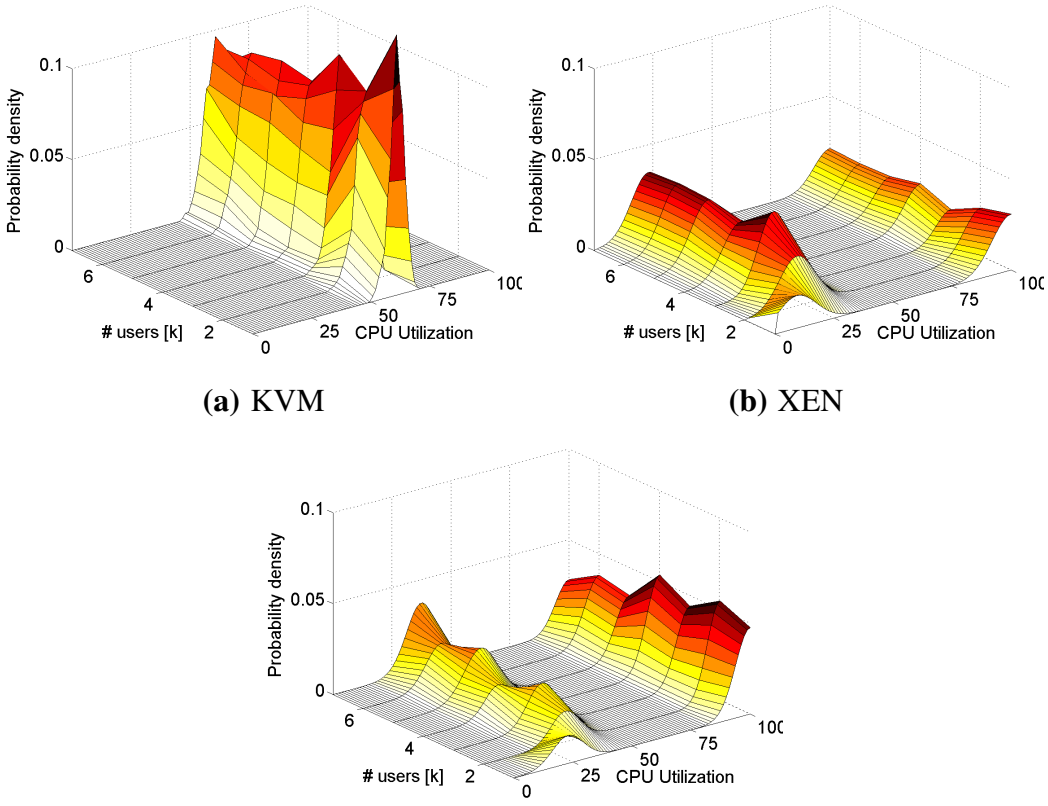
(c) CVM

Figure 53: Individual CPU Utilization for Apache (for 8 VCPUs).

6.5 Related Work

Traditionally, performance analysis in IT systems builds models based on expert knowledge and uses a small set of experimental data to parameterize them [172, 204, 344]. The most popular representative of such models is queuing theory. Queuing networks have been widely applied in many performance prediction methodologies [298, 310]. These approaches are often constrained because of their rigid assumptions when handling n-tier systems due to the complex dependencies. As an illustration of significant characteristics that are hard to capture with traditional analysis, consider the significance of context switching towards system performance when a large number of threads is involved [174].

While the increasing popularity of virtualization and cloud computing has spawned very interesting research on private and public clouds. Barham et al. [70] benchmark Xen against



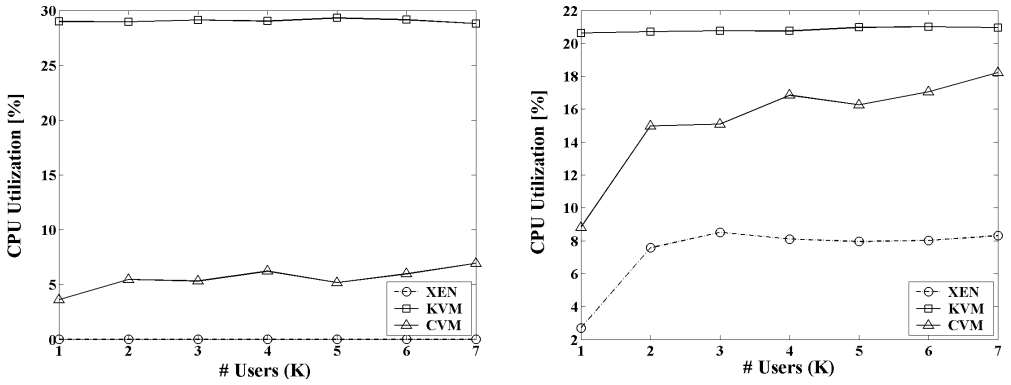
(a) KVM **(b)** XEN

(c) CVM

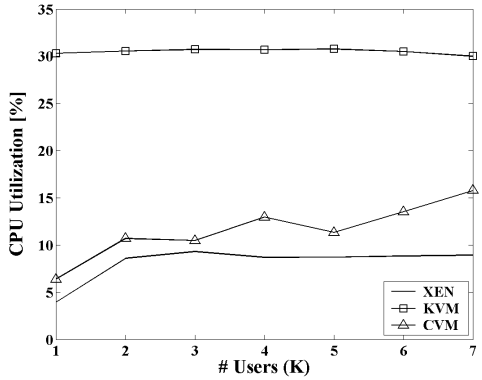
Figure 54: CPU Densities (for 8 VCPUs) [At each second maximum utilization from among all the eight core is taken].

VMware Workstation and User-Mode Linux, they show that Xen outperform them on a range of microbenchmarks and system-wide tests. Clark et al. [115] repeat this performance analysis of Xen in [70] and confirm the high performance of Xen claimed in [70]. Also they compare Xen on x86 with IBM zServer and find that the former has a better performance than the latter. Interestingly, in our study we observed complex results, where different hypervisor show different performance characteristics for varying workloads.

In an another study, Padala et al. [48] concentrate on comparing Xen's and OpenVZ's performance when used for consolidating multi-tiered applications. Their experimental results show that Xen incurs higher overhead than OpenVZ does and average response time can increase by over 400% in Xen and only 100% in OpenVZ as the number of application instances grows from one to four. This can be explained by looking at L2 cache misses; Xen



(a) Hardware Interrupts **(b)** Software Interrupts



(c) System

Figure 55: CPU Utilization Breakdowns for Apache

has higher L2 cache misses than OpenVZ.

Deshane et al. [96] focus on three aspects of benchmarking Xen and KVM: overall performance, performance isolation, and scalability. They illustrate that Xen has excellent scalability while KVM has substantial problems with guests crashing when hosts more than 4 guests. KVM outperforms Xen in isolation. In overall performance test, Xen has a better performance than KVM on a kernel compile test while KVM KVM outperforms Xen on I/O-intensive tests. Camargos et al. [197] analyze the performance and scalability of six virtualization technologies (KQEMU, KVM, Linux-VServer, OpenVZ, VirtualBox and Xen) for Linux. They find Linux-Server delivers little or even no overhead in all test. In all these experiments, Xen, KVM and virtualBox perform well. The experimental result of OpenVZ is disappointing and they suggest that KQEUM should be avoided in production systems.

Several studies focus on building models to predict the I/O performance of the VMs in

consolidation [197]. Based on parameters such as mean queue-length of requests, proposes three classes of linear prediction models for request throughputs, mix of read/write requests, and response times in consolidation. The model in [197] evaluates a queuing network with fair share scheduling using trace-driven simulation. Yet, our experiences on performing large-scale studies suggested the challenges associated with performance prediction using those approaches, mainly because those approaches unable to accommodate complex system characteristics (e.g., garbage collection, context switching).

6.6 Summary

In this chapter we presented an experimental analysis of performance and scalability when n-tier applications are migrated to IaaS clouds. We employed the RUBBoS benchmark to measure the performance on Emulab, Open Cirus, and EC2. Especially the comparison of EC2 and Emulab yielded some surprising results. In fact, the best-performing configuration in Emulab became the worst-performing configuration in EC2 due to a combination of several factors. Moreover, in EC2 the network sending buffers limited the overall system performance. For computation of intransitive workloads, a higher number of concurrent threads performed better in EC2 while for network based workloads, high threading numbers in EC2 showed a significantly lower performance. Our data also exemplified the significance of context switching overheads. We provided a set of suitable candidate solutions to overcome the observed performance problems.

More generally, this work enhances the understanding of the risks and rewards when migrating n-tier application workloads into clouds and shows that clouds will require a variety of further experimental analysis to be fully understood and accepted as a mature technological alternative. In addition, experiment results show the neediness of refactoring n-tier applications before they are deployed on commercial clouds especially when one has to satisfy production SLAs.

CHAPTER VII

PERFORMANCE AWARE VIRTUAL MACHINE PLACEMENT

The increasing popularity of modern virtualization-based datacenters continues to motivate both industry and academia to provide answers to a large variety of new and challenging questions. In this chapter, we focus on and aim to answer one such question: how to improve performance and availability of services hosted on IaaS clouds. Our system, Structural Constraint-Aware Virtual machine Placement (SCAVP), supports three types of constraints: demand, communication and availability. We formulate SCAVP as an optimization problem and show its hardness. We design a hierarchical placement approach with four approximation algorithms to efficiently solve the SCAVP problem for large problem sizes. We provide a formal model for the application (to better understand structural constraints) and the datacenter (to effectively capture capabilities), and use the two models as inputs to the placement problem. We evaluate SCAVP in a simulated environment to illustrate the efficiency and importance of the proposed approach. In our approach, we deploy the application and measure the performance using the automated performance management infrastructure. Finally, we use the collected performance data to build the application model, which we then use during VM placement.

7.1 Introduction

Virtual machine (VM) placement has become a crucial issue in modern virtualization-based datacenters, and it has attracted significant attention recently [9, 12, 30, 79, 94, 106, 131, 153, 201, 278, 285, 302]. However, due to the associated complexity, most existing techniques focus on the placement of individual VMs and ignore the structural information of the application [9, 12, 30]. Yet, many services deployed in a cloud consist of several VMs, which may be clustered for availability or used for different tiers of a multi-tier application. The

performance and availability of such applications greatly depend on how its components (that is to say, the VMs) are placed relative to one another in the datacenter.

In this chapter, we present structural constraint-aware VM placement (SCAVP) to improve the performance and availability of services deployed in Infrastructure as a Service (IaaS) clouds. The proposed approach focuses primarily on the initial VM placement, and it uses the automated performance measurement data to understand the communication and performance characteristics of the application. In SCAVP, we consider an application (we use 'application' and 'service' interchangeably) as a cluster of ***related*** VMs, rather than a set of individual VMs. By augmenting the placement system with awareness of the availability and communication requirements (such systems are typically already aware of demand requirements), we can improve the overall performance and availability of the application.

When deploying an application, SCAVP attempts to satisfy three types of constraints: *demand*, *availability*, and *communication*. Demand satisfaction means providing sufficient resources (e.g., CPU, # of CPU cores, memory) to the service for it to meet its service level agreement (SLA). Observing availability constraints increase the overall availability of the application, perhaps by deploying VMs across different isolation levels of the datacenter. Finally, communication constraints permit us to minimize overall communication costs by assigning VMs with large, mutual communication costs to VMs in close proximity to one another. Minimizing communication cost has significant economic benefits for modern commercial clouds. For example, in Amazon EC2, the cost of data transfer to and from the Internet is \$0.08-\$0.15 per GB, whereas data transfer within the region costs only \$0.01 per GB. Thus, placing VMs intelligently so as to reduce communication costs not only enhances performance but also the pecuniary cost of running the system.

We formulate SCAVP as an optimization problem and show hardness; more concretely, we show that SCAVP consists of four sub-optimization problems. We propose a hierarchical placement approach that efficiently solves the VM placement problem for large problem sizes. Methodologically, we use Vespa [302]—an existing conventional placement approach—and

extend it to support an application's structural information. To better analyze this structural information, we formally model the application, and to effectively capture capabilities we provide a formal model for the datacenter. We then use the two models as inputs to the placement problem. Finally, we evaluate SCAVP in a simulated environment to illustrate the efficiency and importance of the proposed approach.

The remainder of this chapter is structured as follows. In Section 7.2, we formally model the datacenter and application, and we formalize the placement problem. Our hierarchical placement approach is discussed in Section 7.3, and in Section 7.4 we present our algorithms and discuss the complexity. Section 7.5 provides our experimental results. Related work is summarized in Section 7.6, and finally, we provide a chapter summary in Section 7.7.

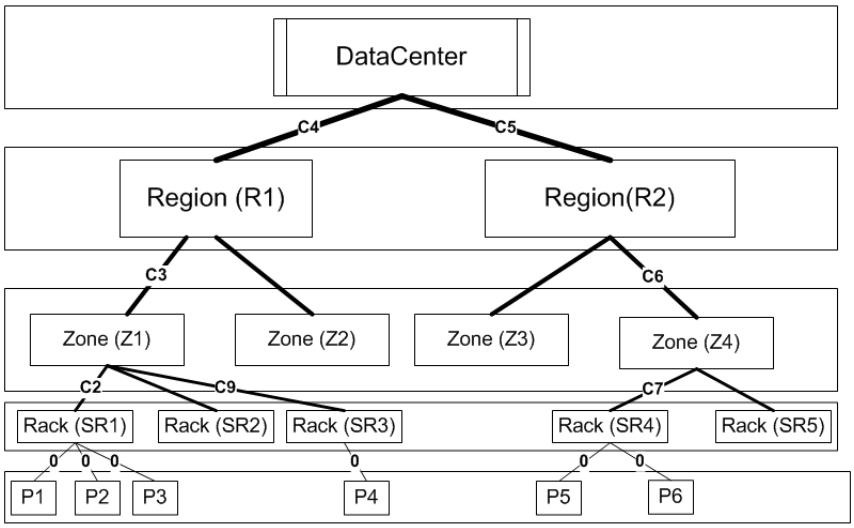
7.2 Problem Formulation

The core of the VM placement problem is to create a placement plan which dictates where VMs should be deployed (that is to say, to which physical machines (PMs)) in order to satisfy various constraints. For brevity, we simply refer to "VM placement" as "placement" in the rest of the chapter. The inputs to a placement problem are an application (i.e., a set of related VMs) and a datacenter (i.e., a set of PMs).

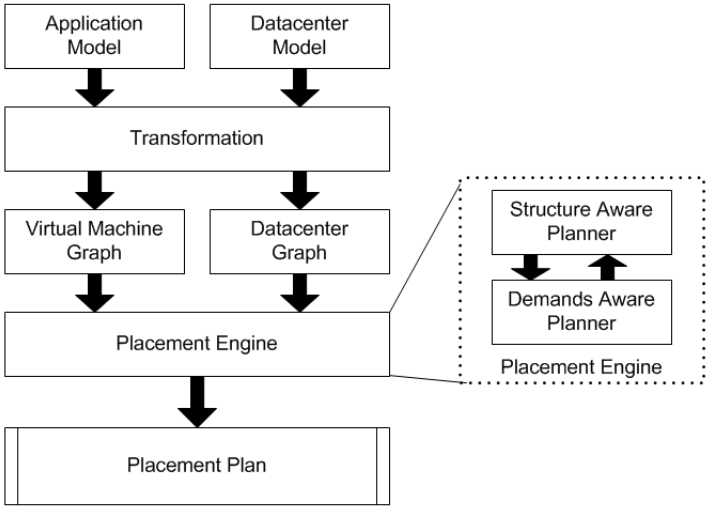
Placement can be further subdivided into 'initial VM placement' and 'runtime VM placement'. The initial VM placement problem is to find the most suitable PMs to deploy the application (i.e., VMs) for the very first time; in contrast, the runtime VM placement problem is to manage the system after initial deployment, including migrating VMs from one PM to another in order to adjust system utilization, improve performance, and reduce the power cost (amongst other possible goals). Without loss of generality, in this chapter we focus on creating an efficient initial placement plan by satisfying the following three structural constraints:

Demand constraint

is defined as the lower bound of resource allocations that each VM requires for the



(a) Hierarchical datacenter model



(b) Hierarchical placement process

Figure 56: System Architecture - Structural Constraint-aware VM Placement

service to meet its SLA. Each VM is given a set of projected resource demands (e.g., CPU, memory, #cores); the placement plan should provide the requested demands.

Availability constraint

is to improve the overall availability of the service. The availability of a service is expressed as a combination of anti-collocation/collocation constraints; the placement plan MUST respect these constraints and produce a plan with maximum service availability.

Communication constraint

is a user-assigned value that represents the communication requirement between two VMs such that the application functions. The value is higher when the communication between the pair of VMs requires more bandwidth or lower latency. Moreover, full constraint satisfaction requires also minimizing the overall communication cost.

7.2.1 Datacenter Modeling

We model the datacenter as a tree structure with arbitrary depth, and we use both physical network topology (e.g., switches, routers) and logical groupings of the datacenter (e.g., power zones, security zones, user defined zones) as key parameters. A typical datacenter can be modeled using three or four levels—an example is shown in Fig 56(a).

The bottom-most level in the tree is the *Server racks* (e.g., SR1, SR2); we then group a set of server racks into a *Zone*. A zone can be a security zone, a power zone or a zone defined through some other means. The communication cost between two PMs in different zones is non-zero, and can vary from one PM pair to another. Similarly we group a set of zones to a *Region*; regions may, for example, be geographically distributed and more isolated from one another and therefore incur significantly higher communication costs.

We assume that the communication cost between any two PMs in the same rack is zero, and that communication costs increase as we move up the hierarchy (i.e., `0 < C2 < C3 < C4` for Fig 56(a)). The communication cost between two PMs in two server racks can be calculated either by counting the switches between the racks or by measuring the end-to-end communication cost (e.g., latency or bandwidth) between the racks. For example, in Fig 56(a) the communication cost between *P1* and *P4* becomes `C2 + C9`.

Next, we formalize different entities in the datacenter model:

Definition 7 *A physical machine (PM) exists only at the base of the datacenter hierarchy and is given by* $P = (N, \vec{PC}, SR)$*, where* N *is the name (an arbitrary value unique within* SR*),* $\vec{PC}$ *is a capability vector, and* SR *is the parent server rack.*

Definition 8 *A capability is a property of a PM and is given by PC = (N, V), where* N *is the name (e.g., CPU speed, memory) and* V *is the value (e.g., 1.7 GHz, 4GB).*

Definition 9 *Slot size* S *of an entity* E *is defined as the maximum number of VMs that can be placed on* E*, furthermore,* S *is calculated using the aggregated capabilities of* E*'s children.*

Definition 10 *Server Rack is given by SR = (N, S, {P}, C), where* N *is the name,* S *is the slot size,* {P} *is the set of all PMs in SR, and* C *is the communication cost to the parent node. Moreover,* $\forall p \in \{P\}$ *is homogeneous with respect to* P*'s type (e.g., operating system) and communication cost between any two PMs in* SR *is zero.*

Definition 11 *The* i^{th} *intermediate level of a datacenter is given by* $L_i = (N, C_i, L_{i+1}, S)$*, where* N *is the name,* C_i *is the communication cost to parent node,* L_{i+1} *is the set of children nodes, and* S *is the slot size at level* i*. If* $i = 0$*, then* L_i *represents the complete datacenter* D*.*

7.2.2 Application Modeling

The main objectives of application modeling are to effectively convert domain specific concepts (e.g., high available cluster, primary-backup cluster) into structural constraints among application components and to permit a better understanding of resource demands. A typical application consists of sets of servers (e.g., Web servers, Application Servers), each of which has constraints and demands. In our model, each individual server is represented as a separate VM, for example, an application with two Web servers and three Application servers is represented as five VMs. More concretely, the dependency, constraints, and resource requirements of servers are transformed to corresponding VMs. Next we formally define different entities used in the application model.

Definition 12 *A virtual machine (VM) is given by* $V = (N, \vec{R})$*, where* N *is the name and* $\vec{R}$ *is the set of projected resource demands. Furthermore,* $\vec{R}$ *MUST be a subset of the capability vector* PC*.*

Definition 13 *A structural constraint* SC *is a link between a pair of VMs: each link MUST be associated with at least one constraint. A structural constraint can be either an availability constraint or a communication constraint.*

Definition 14 *A communication constraint is given by CC = (S, T, C), where* C *is the communication cost between VM* S *and VM* T. *Notice that when cost is zero, the VMs are considered as non-communicating.*

Definition 15 *Availability constraints are a tuple A = (S, T, L, B), where* S *is the source VM,* T *is the target VM,* L *is a level in the datacenter hierarchy ($L \in \cup L_i$), and* B *is the constraint type. Constraint type (i.e.,* B*) becomes anti-collocation if* S *and* T *cannot to be placed on the same intermediate at level* L*, else it becomes a collocation. Note, collocation and anti-collocation have the following properties:*

- Collocation is transitive: if $A \rightarrow B$, and $B \rightarrow C$, then $A \rightarrow C$
- Anti-collocation is not transitive: if $A \rightarrow B$, and $B \rightarrow C$, this does not imply $A \rightarrow C$

Definition 16 *An application is given by an un-directed graph A = ({V}, {SC}), where* {V} *is a set of VMs and* {SC} *is set of structural constraints that connects VMs with each other.*

Problem Statement: For a given arbitrary application *A=({V}, {SC})* and a datacenter *D*, find A's VM placement plan on *D* whereby the VMs in *{V}* are assigned to PMs in *D*. The created placement plan should provide projected resource demands ($\sum \vec{R}$), satisfy structural constraints ($\sum SC$), and minimize the total communication cost ($\sum CC$).

7.3 Placement Process

We have designed a hierarchical placement approach to find the placement plan for a given application on a given datacenter, and our approach is illustrated in Fig 56(b). Firstly, we model both the datacenter and the application, as described above. Secondly, the application model is transformed into a VM graph and the datacenter specification is transformed to a

datacenter graph. Finally, these two graphs are provided to the placement engine as input parameters.

The placement engine is the core of SCAVP and consists of two key sub-planners: the *Structure-Aware Planner (SAP)*, and the *Demands-Aware Planner (DAP)*—the *SAP* handles the structural constraints of the application and the *DAP* handles the resource constraints.

The *SAP* uses the structural constraints of the application and produces a candidate placement plan, which the *DAP* then refines. Methodologically, SCAVP uses a *divide-and-conquer* approach to create the placement plan. In the *divide* stage, the application graph is split into a set of smaller VM groups, and the group size is determined by the slot size at the server rack level. In this process, only the structural constraints are used: the demand constraints are ignored. Next, for each VM group, the *SAP* finds a candidate server rack and refines it using the *DAP*. In this process the *DAP* only uses demand constraints and ignores the other two constraints.

The *DAP* is a conventional *bin-packing* placement engine [302]. It uses individual and aggregated demand constraints of VMs and capabilities of PMs and determines whether the given sets of VMs can be placed on the given sets of PMs to provide the projected demands. The *SAP* is notified when the set of PMs cannot provide the projected demand. This causes the *SAP* to find another candidate server rack (i.e., new sets of PMs) and refine it with the *DAP*. This process continues until it finds a suitable server rack for each VM group.

Finally, in the *conquer* stage, the *SAP* combines each VM group to server rack mapping and produces a complete plan. Deployment toolkits can use the produced plan to deploy VMs to the actual datacenter. The overall process of our approach is outlined below and illustrated in Fig 56(b).

1. Divide the complex VM graph into a set of smaller VM groups. The created groups should maximize intra-communication cost and minimize inter-communication cost. Moreover, groups should satisfy the availability and group's constraints (e.g., slot size of the group).

2. Find the suitable server rack for each VM group, such that the mapping guarantees the satisfaction of anti-collocation and collocation constraints at each level of the datacenter hierarchy.

3. Refine the VM groups to server rack assignment using the *DAP*, and find the most suitable PM for each VM.

Our placement problem can be classified as an optimization problem, which in fact consists of four *NP*-class problems. For example, finding the problem solvability is also an *NP*-hard problem. We find the solvability of the problem using availability constraints, more specifically by using anti-collocation.

The anti-collocation constraint provides a lower bound to the resource requirements; for example, having an anti-collocation constraint between two VMs at the server rack level requires having at least two server racks, one for each VM, to respect the constraints. Thus, solvability is evaluating number of servers required to support the availability constraints. We can show the complexity of the problem by converting it to a graph coloring problem. In graph coloring the intuition is to color each vertex so that no two adjacent vertices have the same color: the minimum number of colors required to color such a graph is called a *chromatic number*. In anti-collocation the requirement is not to put two VMs, which have the anti-collocation constraint, into the same VM group. Thus, the minimum number of server racks to support the anti-collocation constraints at the server rack level can be considered as a *chromatic number*. If the number of available server racks is less than the *chromatic number*, then we cannot solve the problem.

7.4 Algorithms and Complexity

The algorithms presented here can be used for a datacenter with an arbitrary number of levels; however, for simplicity we discuss our algorithms by using the example datacenter model shown in Fig 56(a).

7.4.1 Constraint-aware VM Grouping (bottom-up)

The intuition of constraint-aware grouping is to respect the structural constraints of the application and to create a set of small groups both to minimize the overall communication cost and to enforce availability constraints. SCAVP employs a bottom-up approach, in which it uses the VM graph to build a logical hierarchy similar to that of datacenter: for example, if the datacenter has three intermediate levels then the created VM hierarchy also has three levels.

SCAVP uses the *MinMaxVMGrouping* algorithm to create the VM hierarchy. It starts with the VM level and creates a set of VM groups (VMGs) to enforce the structural constraints at bottom-most level (i.e., server rack). Then, it uses previously created VMGs and creates a new set of VMGs for the next level in the hierarchy (i.e., zone according to Fig 56(a)). Likewise it recursively uses the algorithm and continues the process until it creates a hierarchy similar to the input datacenter.

For each level (including VM) it transforms communication aware grouping to a *K-Min-Max-cut* problem [283]. In *K-Min-Max-cut*, the requirement is to cut the graph into a set of K cluster so that the inter-cluster cost is minimized and intra-cluster cost is maximized. In SCAVP, the requirement is also to create a set of VM groups (no constraints on K) so that communication cost between any two groups is minimized. Moreover, *K-Min-Max-cut* is an *NP*-hard problem; hence, only approximate algorithms are possible. We use a greedy algorithm with a heuristic: the heuristic is to merge two VMs with highest communication cost together to form a group (unless constraints are violated).

7.4.1.1 Formal Problem

Assume $V=\{v_1, v_2, \cdots, v_n\}$ is a set of VMs, and let $l(u, v)$ denote the communication cost between VM v and VM u, and assume s as the slot size. If the number of groups created is $\mathtt{k}$, then,

$$\lceil n/s \rceil \leq k < n$$

Total communication cost within a group (where $\cup$ is the set of all VMs in the group)

$$l(v, \cup) = \sum_{u \in \cup} l(v, u)$$

$$Total\ com\ cost\ (when\ k = n) = \sum_{i<j} l(v_i, u_j)$$

$$Total\ com\ cost\ (when\ k \neq n) = \sum_{i<j} l(v_i, u_j) - \sum_{j=0}^{k} l(v_j, U_j)$$

$\cup U_j = V$ and $u_i \cap u_j = \emptyset\ \forall i \neq j$. The minimum communication cost is achieved when there is only one group and the cost is maximized when there are n groups. Thus, the problem is to find the correct balance while satisfying the constraints.

7.4.1.2 Algorithm

Here we discuss the *MinMaxVMGrouping* algorithm for a single step (say level l) (SCAVP recursively uses the algorithm for all the levels in the hierarchy). Firstly, SCAVP checks the collocation constraints for level l and creates a new set of VMGs to satisfy the constraints. For example, if two VMs (VMGs) are needed to collocate at the server rack level, then a new VMG is created for the two VMs and any others that have transitive collocation requirements with them. This process continues until it processes the collocation constraints of all the VMs.

Secondly, SCAVP creates an $n * n$ communication matrix (n = # VMG at level l) to represent the communication requirement between different VMGs. Next, it picks the element with the highest value from the matrix and merges the two corresponding VMGs together (unless constraints are violated). Then, it updates the communication matrix to reflect the changes and picks new element with the highest value and continues the process until it merges all the possible VMGs. Our algorithm is illustrated in *MinMaxVMGrouping*.

7.4.1.3 Optimization

MinMaxVMGrouping is a greedy algorithm and produces locally optimized results. Thus, further optimization can be used to reduce the total communication cost. We have designed

Input: $V = (V_1, V_2, \cdots, V_n)$, #S
Output: $G = [G_i, G_2 \cdots G_k]$, where k ≤ n
begin
  $G = doCollocationAwareGrouping(V)$
  $G = [G_1, G_2, \cdots, G_m]$, where $\bigcup G_i = V, G_i \subseteq V$ & $G_i \cap G_j = \emptyset \; \forall i \neq j$
  $|C|_{nn} = buildCommunicationMatrix(G)$
  Let current = 0;
  while *true* **do**
    $e = pickTheNextMaxEdge(C, current)$
    $current \leftarrow e.getValue$
    if *current* = 0 **then**
      break
    end
    Let G_i, G_j are the two corresponding vertexes of e.
    if $G_i.size + G_j.size < \#S$ **then**
      $G_i \leftarrow G_i + G_j$
      $G \leftarrow G \setminus G_j$
      $C = updateCommunicationMatrix(G)$
    end
  end
end

Algorithm: MinMaxVMGrouping

another greedy algorithm *OptimizeTotalComcost* which further minimizes the communication cost of produced results. We use a simple heuristic: a VM is chosen from a VMG, and the total communication cost for that VM to and from other VMGs is calculated. If we find a VM that has a larger inter-communication cost (say VMG1) than intra-communication cost, then we move the VM to VMG1 only if the future total cost is less than the current total cost. For example, as shown in Fig 57, value `Z` represents the communication cost to V2 from all the VMs in G1, `X` represents the total cost from G3, and `Y` represent the total cost from G2. If `Z` is less than `X`, it means there is a higher communication cost for V2 from G3 than G1; hence, our algorithm tries to move V2 into G3 if the projected total communication cost is less than the current communication cost. The algorithm is outlined in *OptimizeTotalComcost*.

Input: $B = [VMG_i \cdots VMG_k]$, $V = (V_1, V_2, \cdots, V_n)$
Output: $B = [VMG_i \cdots VMG_k]$ where k < n
begin
  while *change* **do**
    change = FALSE
    $currentTolCost = calcualteCurrentComCost()$
    for *each* v_i *in* V **do**
      Let v_i resides in VMG_c
      C_i = CalCostForEachGroup(v_i)
      $C_i = \{c_1, c_2 \cdots, c_k\}$, where $c_j = cost(v_i, VMG_j) | 1 \leq j \leq k$
      Let c_intra as the communication cost from VMG_c
      $c_{max} = max\{c_i | 1 \leq i \leq k\}$
      Let VMG_{max} is the group with c_{max}
      if $c\_intra < c_{max}$ **then**
        $futureCost = costAfterMove()$
        if *futureCost* < *currentTolCost* **then**
          if *satisfy_group_constraints* **then**
            Move v_i to VMG_{max}
            change = TRUE break
          end
        end
      end
    end
  end
end

Algorithm: OptimizeTotalComcost

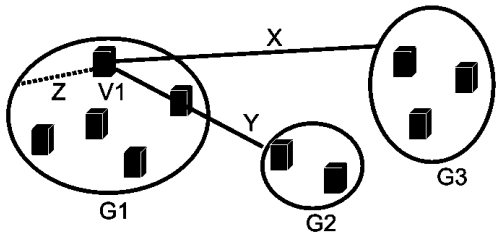


Figure 57: Intra and Inter Communication Cost for a VM *V1*

7.4.2 VMGs to Server Racks Assignment (top-down)

The process of assigning VMGs to racks proceeds in a top-down fashion: it starts with the top-most level and tries to find the most suitable node from the datacenter ('region', in the terminology of Fig 56(a)) to place the largest node at (0^{th}) level from the VM hierarchy. Next, it moves one level down, and tries to find the most appropriate zone (from the selected region) to place the largest zone from the VM group hierarchy. If the datacenter hierarchy has more levels, then this process continues until it reaches the server rack level.

At the server rack level, the problem is to assign a small set of VMGs to a small set of server racks. Here, SCAVP creates a VMG graph (VG) by using aggregated communication requirements. In VG, VMGs are represented as vertices and communication requirements are represented as weighted edges. Similarly by using server racks data, it produces a server rack graph (RG) to represent the communication cost among all the server racks in the selected set. In RG, each vertex represents a server rack and edges represent communication costs. This converts the assignment problem into a graph matching problem, which is said to be a *NP*-complete.

We have designed an approximation algorithm *AssignRacks* with a heuristic to solve the assignment problem. More concretely, we extended [253] to capture the complexity of our problem (e.g., structural constraints). In our heuristic, a VMG pair with largest communication requirement be assigned to a server rack pair with the lowest communication cost.

The *AssignRacks* algorithm finds VMG to server rack mappings. It works as follows: for each VMG, it finds a candidate server rack and then refines it using *DAP*. If *DAP* can place

the VMs on the candidate server rack to meet the requested demands, then algorithm moves to then next VMG, otherwise it tries to find another candidate and refines it with *DAP*. Once each VMG is assigned to a server rack the algorithm moves to the next zone. Failure to find an assignment to at least one VMG causes it to try another zone from the same region and repeat the above procedure. Unless it finds a suitable zone from the selected region, it tries another region from the datacenter and refines the assignment using *AssignRacks*.

Input: $VG = \{G_1, \cdots, G_k\}$, $RG = \{R_1, \cdots, R_r\}$ where k ≤ r
Output: $|P|_{kr}$
begin
  Let M is completed set of VMGs, and R is completed set of racks.
  $M \leftarrow \emptyset, R \leftarrow \emptyset$
  while $|M| < n$ **do**
    e_{max} = pickMaxEdge($VG\backslash$)
    $\{E\}$ = picMinEdges($RG \setminus R$)
    e_r = *pickAnEdge*(E)
    Let e_{max} *as the cost between* G_i *and* G_j
    Let e_r *as the cost between* R_x *and* R_y
    if G_i *and* G_j *can be placed on* R_x *and* R_y **then**
      Let $TV \leftarrow \{G_i, G_j\}$, and $TR \leftarrow \{R_x, R_y\}$
      while *hasmore* **do**
        $e_{max} = getMaxEdge(TV \rightarrow VG \setminus (M \cup TV))$
        if $e_{max} = NULL$ **then**
          hasmore = FALSE
        end
        else
          $e_r = pickAMaxEdge(TR \rightarrow RG \setminus (R \cup TR))$
          $N(e_{max}) = (v, v_k)$, where $v \in TV$
          $N(e_r) = (r, r_k)$, where $r \in TR$
          $addMapping(v_k \rightarrow r_k, P)$
          $M \leftarrow M \cup v_k, R \leftarrow R \cup r_k$
          $TV \leftarrow TV \cup v_k, TR \leftarrow TR \cup r_k$
        end
      end
    end
  end
end

Algorithm: AssignRacks

7.4.3 VM to PM Mapping (bin-packing)

In our system, we use Vespa [302] as the *Demands-Aware Planner*. For the purposes of this work, we can regard Vespa as a conventional placement engine which considers VM placement as a bin-packing problem and provides a placement plan that maximizes the demand satisfaction. Hence, the refining process is also an *NP*-complete problem. SCAVP uses Vespa to process the demand constraints and finds the VM to PM mapping. More concretely, *AssignRacks* finds candidate server racks for a VM group and refines it with Vespa. In that process, Vespa tries to place the set of VMs onto PMs in the given so that PMs can provide the projected demand. If the server rack can provide the projected demand then it produces the VM to PM mapping, else it returns `false` causing our algorithm to find another candidate server rack. When Vespa returns `true`, it also provides detailed a description with $VM \rightarrow PM$ mappings and individual demand satisfactions (e.g., CPU, Memory, #cores).

7.5 Evaluation

We have created a SCAVP prototype in Java, there we have implemented all the mentioned algorithms and integrated with Vespa. In this sections we analyze SCAVP in a simulated environment and provide experiment results to illustrate the efficiency of our approach. To the best of our knowledge no other existing tools handle the structural constraints supported in SCAVP. Thus, our comparative analysis is focused on finding the improvements of SCAVP with conventional placement approach Vespa.

We simulated four different application architectures each having different resource, communication, and availability requirements. We selected these four types to cover a broad range of commercial applications. For each application type we evaluated the placement efficiency by varying the application size from 20 VMs to 100 VMs (e.g., 20, 30, . . . ,100), and for each configuration, we performed 400 experiments and present here the averaged results.

- 3-tier: Modeled after the traditional three tier web serving architecture, wherein an initial web tier serves static content, an application tier serves dynamic content, and a database tier holds persistent data for the application tier. We assume all the servers in a given tier communicate with all the other servers in the same tier with similar communication costs. The communication cost between the web tier and application tier is less than that of the cost between the application and database tier. All the servers in any given tier are assumed to be homogeneous.

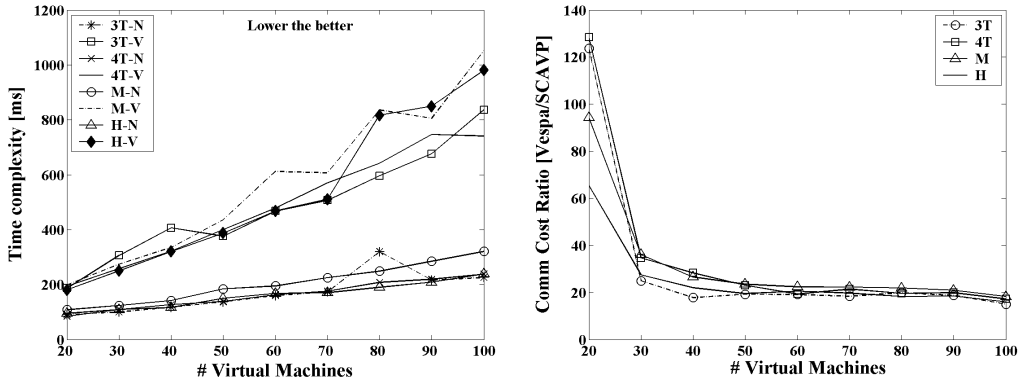
- 4-tier: The difference between 3-tier and 4-tier is the existence of a cache tier. We assume higher communication requirements among the serves in the cache tier compared to other tiers.

- Map-reduce architecture: All servers communicate with all the other servers with same communication requirements. All the servers are assumed to be homogeneous.

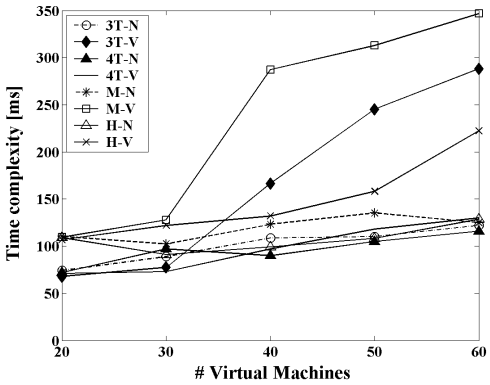
- Server mesh: An arbitrary application, where a set of heterogeneous application components are connected using heterogeneous communication requirements.

We simulated two types of datacenters: a small datacenter consisting of up to 60 PMs and a large datacenter consisting of over 200 PMs. Furthermore, we simulated each server rack to have at most 16 PMs, and each PM in a given server rack is homogeneous with respect to its types. The datacenter was simulated to have heterogeneous communication cost among different server pairs. To represent the running status of an actual datacenter each PM was simulated with different capabilities (resource utilization levels). Moreover, we simulated two and tree regions for the small and larger datacenter respectively.

Notation: We use a notation #A-#T to represent each concrete configuration. #A denotes the application architecture (*3T* for 3-tier, *4T* for 4-tiers, *M* for server mesh, and *H* for map-reduce). #T denotes the placement engine *N* for SCAVP and *V* for Vespa. For example *3T-N* represents 3-tier application evaluated using SCAVP.



(a) Time complexity on large datacenter **(b)** Com-cost improvement on large datacenter



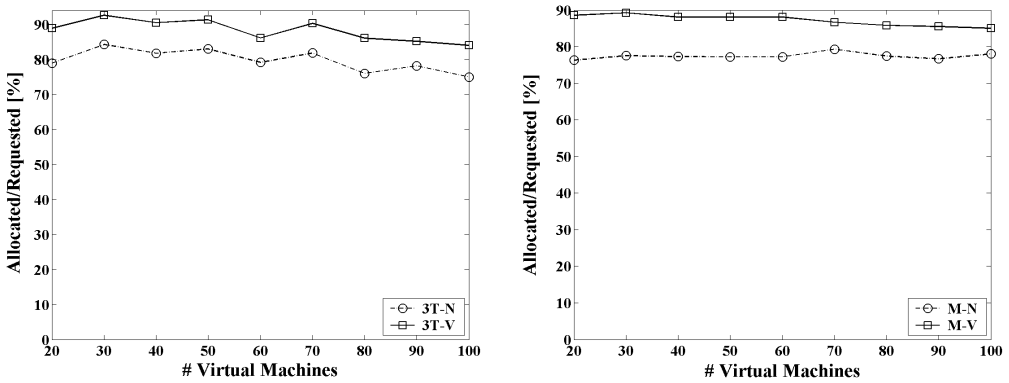
(c) Time complexity on small datacenter

Figure 58: Comparison of Time Complexity and Efficiency of Created Placement Plan

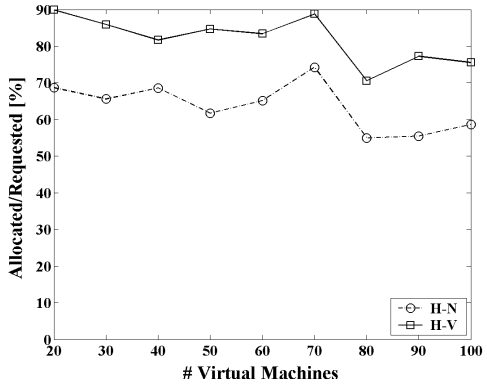
7.5.1 Evaluation on Large datacenter

First, we evaluated the time complexity of two approaches by analyzing the time taken to produce the placement plan. Our results showed that SCAVP performed much better when compared to Vespa alone. More importantly, as the size of the application increases, Vespa takes significantly longer than SCAVP. This is expected: as Vespa uses a bin-packing approach, when the problem size increases, the complexity increases exponentially. With SCAVP, the problem is subdivided, and overall complexity is reduced. Our experiment results are illustrated in Fig 58(a).

We then measured the improvement in communication cost when SCAVP is used instead of Vespa alone. Of course, Vespa does not consider communication cost, and so



(a) 3-tier system **(b)** Server mesh system

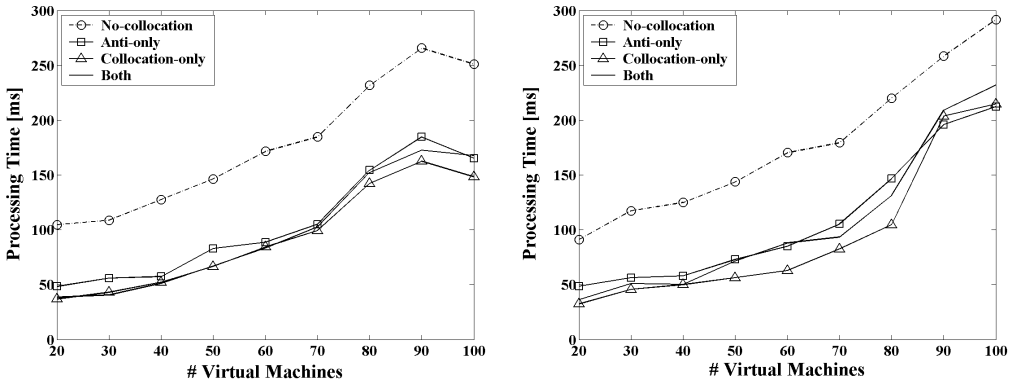


(c) Map-reduce system

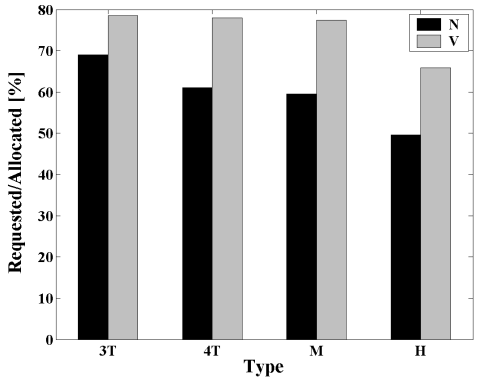
Figure 59: Percentage Demand Satisfaction on Large Datacenter (Requested resource demand/ Allocated resources)

an improvement is expected: Vespa tries to find the most suitable PMs that can provide the requested resources without considering their location. As a result, the PMs that are chosen can be spread across the datacenter. In contrast, SCAVP focuses on communication cost minimization; thus, it produces a placement with less communication cost. For each application size, we calculated the total communication cost produced by the two approaches. Then we calculated the ratio to find the improvements (Vespa's cost/SCAVP's cost), and our results are shown in Fig 58(b). As shown in the figure, by confirming our assumptions SCAVP produces a placement plan with significantly less total communication cost. More specifically, the plan produced by SCAVP has 20 times less communication cost to Vespa.

Next, we measured the demand satisfaction when SCAVP is used and compared it



(a) 3-tier systems on large datacenter **(b)** 4-tier systems on large datacenter



(c) Demand satisfaction (%)

Figure 60: a), b) Time Complexity with Allocation Constraints; c) Average Demand Satisfaction (%) for Four Application Types on Small Datacenter

with Vespa alone. Here we expect Vespa alone to achieve a higher demand satisfaction (at the cost of no network consideration and greater runtime). Vespa finds the placement plan to maximize demand satisfaction across all the VMs; in contrast, SCAVP tries to maximize demand within a small group (datacenter vs. a server rack). Our experiment results justified our assumptions, where Vespa performed better compared to SCAVP. Our results are illustrated in Fig 59(a), (b) and (c). As shown in the figure Vespa provides average of 90% demand satisfaction while SCAVP gives 80%.

We then evaluated the effect of availability constraints by measuring the time complexity. We analyzed it in four different ways: 1) collocation only, 2) anti-collocation only, 3) both collocation and anti-collocation, and 4) no collocation nor anti-collocation. Existence of structural constraints helps to reduce search speech significantly (e.g., collocation between

two VM enforces to consider two VMs as one VM), hence application with availability constraints produced the placement plan much faster compared to that of with no constraints. Our experiment results are shown in Fig 60(a) and (b), and as illustrated in the figure, when the application has either type of constraints, then time complexity reduces by around 30% (30%—50%).

7.5.2 Evaluation for Small datacenter

We extended our analysis to smaller datacenters with the intuition of finding the significance of SCAVP when the problem size is smaller. We first measured the time complexity for the two approaches and our results are illustrated in Fig 58(c). As shown in the figure, even for smaller problem sizes SCAVP performs better compared than Vespa alone. As shown in the figure, for smaller datacenters, the difference in time complexities not that significant: Vespa can find a placement plan very quickly. Nevertheless, the generated plan has similar communication gains compared to that of large datacenters. We also measured the demand satisfaction, due to the negligible variances we have given the average demand satisfaction in Fig 60(c).

7.6 Related Work

In modern virtualization-based datacenters, the problem of VM placement has become a crucial issue and attracted significant attention recently [9, 12, 30, 79, 94, 106, 182, 210, 228, 242, 302]. To achieve high levels of efficiency in cloud computing requires robust strategies for VM placement. We have previously presented a bin-packing based approach [302] which maximizes the resource satisfaction on a given datacenter. A static heuristic-based vector bin-packing algorithm to minimize the number of required servers is discussed in [285]. Grit et al. has presented a broker-based scheme architecture and algorithm for assigning VMs to physical machines [153]. Kimbrel et al. proposed an approach to minimize the number of VM migrations while reallocating resources [190]. The approach presented here supports both resource and structural constraints of the application.

Power optimization has also become a crucial factor in VM placement and greatly motivated the research community. For example, the authors of [201] focus mainly on power optimization. [131] proposes a linear power model for full-system power analysis and modeling, while [329] presents a power-aware placement algorithm based on a power model and a migration cost model. A similar approach is proposed in [165], which discussed how to maximize the revenues of cloud providers by trimming down their electricity costs.

Placement approaches focus on profit maximization have received significant attention too, for example in [106], the authors describe how to place VMs across a number of datacenters to increase profit, and [109] presents an example of an economic model of a datacenter. [278] proposes a graph based solution for semi-automated service creation, which expresses the mapping between a business support system and an operations support system.

Duong et al. present a placement approach to improve the performance on highly interactive distributed virtual environments, which has some similarities with our communication optimization approach [300]. The closest approach to ours is [228], which discusses network-aware placement; however, we go beyond and also present a hierarchical approach to support structural information of the application.

7.7 Summary

We proposed a structural constraint-aware virtual machine placement as an effective way to improve the performance and availability of services hosted on modern datacenters, particularly on IaaS clouds. Careful VM placement can reduce communication costs significantly and improve overall system availability. We formalized the datacenter and application, and used two models as inputs to the placement problem. We formulated VM placement as an optimization problem and show NP hardness, and we propose a hierarchical approach composed of four approximation algorithms to solve the placement problem for larger problem sizes. Finally, we illustrated the significance of proposed system on a

simulated environment.

CHAPTER VIII

CONCLUSION

In this dissertation, I argued that the challenges associated with large-scale experiment measurements can be successfully addressed by automating this process's key activities. First, I formulated the problem by showing the importance and challenges associated with large-scale performance studies in modern clouds. These were both management challenges (such as deployment, configuration and execution) and data challenges (i.e., collection, process, parse, store and analyze). Second, I developed an automated infrastructure to reduce the complexity of the performance measurement process that included a code generator, experiment driver, data extractor, data warehouse, and data analysis portal (Chapter 3). Next, I proposed an approach to provide a resilient deployment runtime to deploy and configure applications as well as recover from both deployment and runtime failures (Chapter 4). Finally, I demonstrated the success of our automated approach by using the infrastructure to conduct large-scale performance measurement studies (Chapter 5, Chpater 6, and Chapter 7).

The main technical contributions of my dissertation work can be arranged: first, building an infrastructure to automate experiment measurements; second, demonstrating the success of the approach through large-scale performance studies (see Figure 4). To present strong evidence in support of my thesis, the technical contributions for those to categories are listed below.

An Automated Infrastructure for Large-scale Performance Studies:

★ In Chapter 3, I discussed the challenges associated with large-scale experiments and in particularly challenges of building an automated infrastructure to aid the performance measurement process. We addressed the challenges by building a multi-stage code generator which uses XML, XSLT and AOP to provide the required level flexibility,

extensibility and heterogeneity.

★ We discussed the key components of the automated infrastructure we have developed to handle the unique challenges of the measurement process (create, manage and analyze): Code generator, Data Extractor, Data warehouse, and Data analyzer in Chapter 3.

★ Failures are common in modern clouds, yet recovery models provided by the cloud providers unaware of the application semantics. Hence, to provide a better recovery capability we built a model driven, peer-to-peer fault tolerant runtime to recover from cloud failures both during deployment time and after deployment (Chapter 4).

Demonstrating the Success of Automated Approach:

★ In Chapter 5, I demonstrate the success of automated experiment management infrastructure by using it to find the significance of performance difference when an n-tier application is migrated from a traditional data center (Emulab) to a public cloud (EC2). Through our study we observed non-trivial performance phenomena and discussed candidate solutions to mitigate those issues.

★ To demonstrate the success in different perspective, we evaluated performance differences in six clouds using two types of n-tier benchmark applications (RUBBoS and Cloudstone) and our results are presented in Chpater 6.

★ My study on the use of performance data for placing virtual machines within infrastructure functioning as service clouds (Chapter 7).

★ The same automated framework is used for many other performance studies by other group members [212–215, 219, 220, 333, 334, 348].

The main contribution made by this dissertation research can be considered as a first step toward building a comprehensive automated framework for enabling large-scale performance studies in modern cloud environments. Among the many potential future directions,

Super staying and *Modular Benchmarking* can be considered as immediate next-steps for the work.

Super Staging: Cloud brings new opportunities by enabling anyone to leverage massive computing capabilities. This helps users go beyond traditional prototype-based staging by enabling them to experiment with real, prospective workloads. By doing this, a user can tune their application to support future workloads. We call this approach "super-staging." Our software tools will automatically generate benchmark execution scripts that superficially resemble traditional staging (simulating a production run). The major difference is that our tools and scripts go much further than traditional staging, since we can generate and run benchmark executions with configurations that may far exceed the size and scope of the application's typical production runs. Thus, super-staging supports not only the validation of execution scripts but also the performance prediction and validation of future workloads and planned configurations. A super-staged benchmark execution script is a self-contained and complete execution environment, including all the components required for the execution such as the operating system, middleware, services and servers, the benchmark application itself plus its input databases and files. Using our tools, such super-staged scripts can authenticate themselves and auto-launch in clouds with sufficient resources.

Modular Benchmarks: The cloud is designed to run heterogeneous applications and workloads; hence, the traditional approach for evaluation with a single representative benchmark does not provide enough insight into the performance of clouds. Clouds need new benchmarks and benchmarking approaches [77]. In our approach, we adopt and develop modular benchmarks that can be divided into components, forming a benchmark kit. As examples, these benchmarks include n-tier applications (e.g., electronic commerce), embarrassingly parallel applications amenable to parallel execution through MapReduce, and micro benchmarks. The main challenge of the benchmark kit is a principled design of interactions

between application logic and back-end data store, since dependencies often arise, making the decomposition of mutually dependent interactions difficult.

The heterogeneity of clouds and diversity of applications we used for performance measurements have introduced many non-trivial challenges to the automated infrastructure discussed here. We have addressed most of those challenges and designed and developed our framework to be robust and reliable. Nevertheless, the framework itself has its own limitation as well. First, validation of generated code is not trivial but an important factor. In the current approach, we validate generated code by running them on the actual cloud, but as part of the future work we are planning to use a standard domain specific language (e.g., SmartFrog [16], OMG DEPL [6]). Second, visibility of meta-data, all the meta-data are stored/embedded into the templates themselves, and the framework can understand pre-defined sets of meta-data. Adding new tags or meta-data may not be visible to the end-user and may result in invalid codes. Third, dependency management, in particularly handling of deployment and start order is complex for applications like n-tier applications. We handle that using a sequence numbering scheme, our approach work well for small and medium size applications, however, applications with hundred of components introduce new challenges. To solve that, we plan to use Petri-net or similar dependency management approaches.

Most cloud users are interested in moving their application workload to a cloud to simplify their application management and reduce the cost of sophisticated IT expertise; however, the technical problem outlined in this document presents a steep learning curve for moving serious workloads into a cloud. In practice, most native cloud users are challenged to develop a robust hardware and software configuration to support their application's performance in the cloud, because they are challenged with acquiring the expertise to design and run experiments that test such configurations. As result, they confront the more substantive business problem stated earlier–choosing between constant and uniform costs with low

resource utilization or occasional but high costs (due to unpredictable response times) when workloads peak (i.e. high resource utilization).

Experimental measurements to predict performance thresholds for representative applications through standard benchmarks would provide significant value for cloud users and service providers. For cloud users, a performance predictability threshold would lower the barrier of entry, since they could focus on confirming the "safe" configurations, or those operating below the threshold, for their applications. For cloud service providers, a performance predictability threshold would lower operational costs and increase return on investment. By eliminating the uncertainty caused by the current (lack of) knowledge on performance unpredictability, service provides could run consolidated VMs at higher utilization levels than are otherwise currently feasible.

APPENDIX A

BENCHMARKS AND DATABASE MIDDLEWARE

A.1 RUBBoS Benchmark Application

RUBBoS [25] is an n-tier electronic commerce system modeled on bulletin board news sites such as Slashdot, and it is widely used in large-scale experiment studies. We modified the benchmark to suite for our experiment workloads and to collect various statistics. The workload consists of 24 different interactions such as register user, view story and post comments i.e., combination of read-only and read-write workloads. The benchmark can be deployed as a three-tier or four-tier system and places a high load on the database tier. Also, benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes. A typical RUBBoS deployment with MySQL Cluster is shown in Figure 61.

Workload Generator: RUBBoS comes with a workload generator which can be used to generate requests from multiple computing nodes. Workload generator creates HTTP connections and access different URLs in the applications, as mentioned before RUBBoS has 24 different truncation types and each has a unique URL. Workload generator consists of multiple clients and each simply emulates a typical client behavior (like accessing a web page), the distribution of different transaction types can be easily configurable. Each client runs as a separate thread and generates HTTP requests using an exponential distribution with average think time of $7s$ between two subsequent requests. A workload generator can have many clients, for example workload of 1000 means 1000 concurrent clients (i.e., 1000 threads). In our approach, we use multiple workload generators and each has equal number of clients. As an example, when generating workloads of 1000 with five workload generators, each generates 200 users.

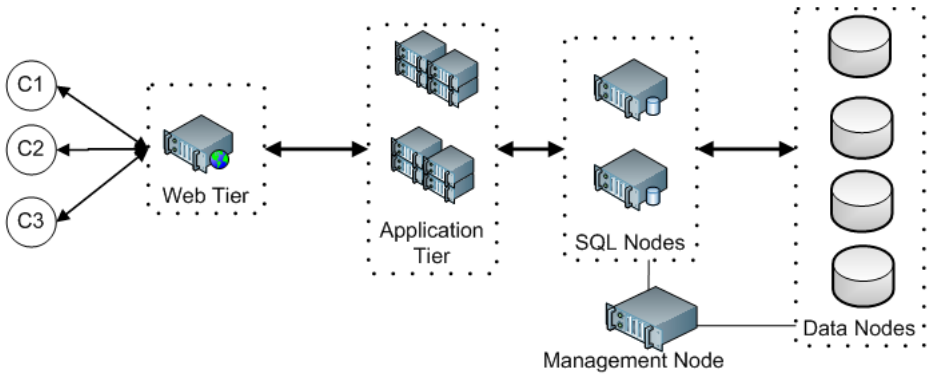


Figure 61: RUBBoS Deployment Topology with MySQL Cluster

A.2 Cloudstone Benchmark Application

CloudStone [294] is a Web 2.0 benchmark, which includes a Web 2.0 social-events application (Olio) and can be considered as a multi-platform, multi-language performance measurement tool for Web 2.0 and Cloud Computing. Cloudstone spun out of a research project at UC Berkeley. More than a benchmarking service itself, it has created an open-source framework for testing cloud performance. The research team built a selection of tools for generating various load levels and testing the performance under those loads across various cloud providers. Its workload generator is implemented using *Faban*, however, in our study we used *Rain* workload generator toolkit. Rain is a statistics-based workload generation toolkit that uses parameterized or empirical distributions to characterize different classes of workload variations. A typical Cloudstone deployment is shown in Figure 62.

A.3 MySQL Cluster

MySQL Cluster [11] is an open source transactional database designed for scalable high performance access to data through both data partitioning and data replication. MySQL Cluster is implemented by a combination of three types of nodes (Management nodes, SQL nodes and Data nodes). A Management node maintains the configuration of SQL nodes and Data nodes, plus starting and stopping of those nodes. A Data node stores cluster data,

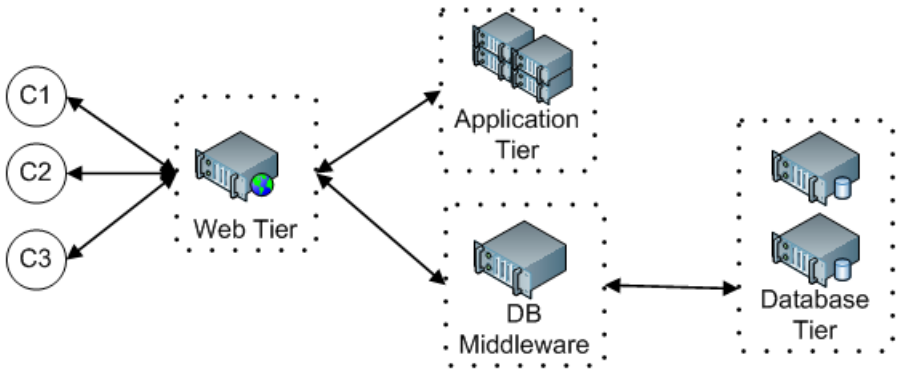


Figure 62: Cloudstone Deployment Topology

and an SQL node is the clustering middleware that handles the internal data routing for partitioning, and consistency among replicas for replication. Figure 61 illustrates three types of nodes and how they are connected. The total number of Data nodes is the product of the replication factor and the number of partitions. For example, with replication factor of 2 and database divided into 2 partitions, the resulting MySQL Cluster will have four Data nodes. Notice, all the experiments described in this document were run with replication factor as 2.

APPENDIX B

TUTORIAL - AUTOMATED DATA EXTRACTION

This tutorial explains how to export performance measurement (raw) data collected by running large-scale experiments to the the data warehouse (i.e., Exprestore). Automated data extractor comes with a number of data processors and parsers to aid the data extraction process, here we discuss how to use the data extractors and relevant configuration files. The main input to the data extractor is a mapping file which tells how to map log (i.e., response time, throughput and resource monitoring data) files to dynamically created tables. One can create the mapping file manually, generated automatically or customize an auto generated file. Regardless of the way the user creates the file, the user can use the automated script to upload/export experimental data to Experstore.

Notice, that a user needs to have a separate mapping file for each individual experiment, nevertheless the user can reuse most of the contents and might need to change only the directory to workload mapping.

B.1 Information about Mapping File

This section provides an overview about the different elements in the mapping file and how to use them to customize the data extraction process.

B.1.1 Experiment Metadata

In general, when analyzing performance measurement data, it is good to know additional information (i.e., meta-data) about the experiments such as date, time, user, cloud, and application. As a result, during the data extraction process, data extraction tools collect a set of parameters and a code snippet of the mapping file with such parameters are shown below:

Listing B.1: Code Listening for Experiment Metadata

```
<name>rubbos-1-2-2-2-2vcps-vmware</name>
  <description>RUBBoS Experiment with VMware (2 VCPUs)</description>
  <user>deepal</user>
  <date>2012-03-27</date>
  <cloud>Sigiri Cluster</cloud>
  <nodecount>8</nodecount>
  <rampuptime>480000</rampuptime>
  <runningtime>720000</runningtime>
  <downramptime>30000</downramptime>
  <application>RUBBoS Benchmark</application>
```

B.1.2 Profile

One of the key elements in the mapping file is the *profile*; profile tells the structure of the database table (i.e., database schema), the corresponding parser to process the file, data types and their formats, and what data to read and not to read. Two samples profiles are shown below, one is configured for `dstat` monitor based data and other is for `sar` monitor based data.

Listing B.2: A Sample Profile for Dstat Monitoring Data

```
<profile>
    <separator>,</separator>
    <resource-name>CPU0</resource-name>
    <processor-class>dataimport.filter.CSVFileProcessor</processor-class>
    <!--Column index start with 0, i.e., first column is zero -->
    <column index="'0" colname="user" datatype="double"/>
    <column index="'1" colname="system" datatype="double"/>
    <column index="'2" colname="idle" datatype="double"/>
    <column index="3" colname="wait" datatype="double"/>
    <column index="4" colname="hi" datatype="double"/>
    <column index="5" colname="si" datatype="double"/>
    <start-index>10</start-index>
    <end-index>0</end-index>
 </profile>
```

Listing B.3: A Sample Profile for SAR Monitoring Data

```
<profile>
```

```
    <separator> </separator>
    <resource-name>CPU0-SAR</resource-name>
    <processor-class>dataimport.filter.SarLogParser</processor-class>
    <!--Column index start with 0, i.e., first column is zero -->
    <column index="0" colname="timestamp" datatype="timestamp" format="hh:mm:ss"/>
    <column index="3" colname="user" datatype="double"/>
    <column index="4" colname="nice" datatype="double"/>
    <column index="5" colname="system" datatype="double"/>
    <column index="6" colname="iowait" datatype="double"/>
    <column index="7" colname="steal" datatype="double"/>
    <column index="8" colname="idle" datatype="double"/>
    <!--Which row need to read, start with 1 -->
    <row index="0" rowname="name-of-row"></row>
    <start-index>2</start-index>
    <end-index>0</end-index>
</profile>
```

Following list provides a detailed description about all the elements:

- *separator*: The data will be in a some sort of a file either a CSV file or some of other format, and each file might contain multiple types of data (e.g., user, sys, wait etc, network send, receive) and each of which might be separated by a delimiter (e.g., ',', ':', ' '). So *separator* is used to specify the delimiter.

- *resource-name*: Name of the resources, this becomes the name of the profile as well as a part of the name of dynamically created table.

- *processor-class*: is the Java class which knows how to process the resource file e.g., a CSV file for a given resource.

- *column*: Each file might contain monitoring data for different resources (e.g., CPU, Memory), and each of which possibly becomes a column in the created table. This provides a way to specify that mapping i.e., which data column corresponds to which column in the table.

  - *index*: Index of the column (always start with '0').

- *colname*: Name of the column used in the database table.
- *datatype*: Data type for the database.
- *format*: How to format the data, for example if the data is `Datetime`, then what should be the format.
- *size*: What is the data size, default is set to 15 (for varchar).

- *row*: Some monitoring tools record monitoring data as a series of rows, for example `sar` collects data in this manner, similarly RUBBoS collect its client response time data in a similar manner. So, this variable is used to specify which row to read.

  index: There are instances where we need to ignore some rows, so this is to specify what to include and what to exclude.

  rowname: Name of the row you need to read.

- *start-index*: There are instances where you do not need to read from a beginning of a log file (i.e., ignore first few rows), this will help to achieve that.

- *end-index*: Similar to start index (0 means read all).

A profile can refer to another profile, and the intuition of doing that is to share a same Database table across multiple profiles. A user can refer a profile as shown below:

Listing B.4: Code Listening for Refereed Profile

```
<profile ref="CPU1">
```

B.1.3 Mapping

For each node in the experiment, we monitor several resources (e.g., CPU, memory, I/O) and depending on the node we might have different sets of resources. In the profile section we create all the potential resources that we monitor during the experiment, rather all the types of data we would like to upload to the database. Here we specify which node has what data and what log file corresponding to what node. A sample mapping file is shown below:

Listing B.5: Code Listening for Mapping

```
<mapping nodename="Apache" filname="169.254.100.3.csv" startwith="false" endwith="false"
                profiles="CPU0,DISK,CPU1,NETWORK,SYSTEM"/>
```

- nodename: Name of the node, this is simply referring to the name that a user specified under Nodes.
- filename: The name of the file, for this we can have three scenarios:
    1. complete file name: If the user knows the full file name across all the workloads for example 169.254.100.3.csv (in this case the user can set both startwith and endwith to false).
    2. startwith: The user does not know the complete file path, but he/she knows the file prefix for example 169.254.100.3-*. In this case the user can set the startwith flag to be *true* and set the file name as the file prefix.
    3. endwith: similar to startwith.
- profiles: What the type of profile that a user can retrieve from this file. For most files a user will have one profile, but for some a user can have many.

B.1.4 Experiment Node

Each experiment consists of multiple nodes, and we need to specify all the nodes we would like consider for the data analysis. The way of specifying the node information is as follows:

Listing B.6: Code Listening for Experiment Node

```
<node>
    <name>MySQL2</name>
    <ip>169.254.100.7</ip>
    <profile>pc3000</profile>
    <hostnode>PH1</hostnode>
    <software>mysql-cluster-gpl-6.2.15</software>
  </node>
```

- name: Name of the node (This refers to same value you used under mapping).
- ip: IP address.
- profile: hardware profile name (assuming you have a way to map this to something you know).
- hostnode: if this is a VM, then where you have hosted this (an IP address of the physical machine).
- software: What kind of software do you run on this server.

B.1.5 Workloads of the Experiment

Each experiment consists of multiple workloads, and in RUBBoS each workload results in a separate directory. So when parsing the data, there should be a way to map a directory to the workload. This section helps you to do that (you need to add the mapping for each workload).

Listing B.7: Code Listening for Workload

```
<workload name="1000" readonly="true" directory="2012-03-02@16-48-03"/>
```

- name: Name of the workload 1000, 2000 etc...
- readonly - Indicating whether the workload is read only or read-write
- directory - File name of the directory

B.1.6 Database Information

Most of the time you will be using the same database, but in case you need to use a separate database, then user can use following procedure to change the database settings.

Listing B.8: Code Listening for Workload

```
<dbinfo>
        <username>dbuername</username>
```

```
        <passowrd>dbpassword</passowrd>
        <url>dburl/elba</url>
        <dbclass>com.mysql.jdbc.Driver</dbclass>
</dbinfo>
```

B.2 Steps of Using the Tool

The experiments which have completed with `Dstat` as the monitor and modified RUBBoS client workload generator, then you can use the following command to auto generate the mapping file:

1. Login to the script host (bonn or elbafs.cc.gatech.edu)
2. Go to your result directory
3. Execute */mnt/sdb/elbadata-store/scripts/generateMapping.sh . full-path-to-XTBL-file*

The above script generates a valid mapping file for the results, but you might need to modify it to suite for your requirements (i.e., structure of the results directory and format of measurement data). In most cases it might not have the workload mappings (i.e., a result directory to a workload), so you need to create that manually (above script also support for other scenarios e.g., `sar`, but you need to generate and modify the file to make sure it correctly map your experiment).

You can manually create the complete mapping file, once you create the mapping file use the following command to upload the data to the database. First, go to the results directory and then run:

/mnt/sdb/elbadata-store/dbscripts/dataimport.sh . path-to-mapping-file

The process might take few hours depending on the size of the data, so it is recommend running in the data extraction as a background process.

APPENDIX C

TUTORIAL - DATA VISUALIZATION

Experstore is an attempt to aid the analysis of large amount of data collected from experiment measurements. It equips the user with the capability to identify patterns, trends and relations which generally gets obscured by the massive quantities of data. Also, it equips the user with capabilities to control the way in which data is represented which further enhances his productivity and analytical capabilities. The application provides a simple interface using which data from seemingly unrelated sources could be easily compared against each other and in this way various latent, potentially interesting, relations can be identified. We have implemented the data visualization tool as a Web portal and it provides following features:

C.1 Graphs as a Function of Original Values

Besides the standard graph, Experstore supports customized graph, that is representing graphs as a function of original values. It is implemented by offset based modifications and complex mathematical function that can be used to compute the customized value based on the input values. Figure 63 below illustrates the customized graphs feature. The blue line is the original graph, and the red line is the customized graph computed after function `f(x) = x + 10` to specify the offset.

The user function is taken as a string and the character 'x' is replaced with the value that we obtain from the database. The graph is plotted by evaluating the arithmetic expression in the resultant string.

C.2 Bucketed Graph for Correlation Analysis

Experstore supports the grouping of ranges of data values based on input parameter. The user can specify a step size and we use it as a range, and the application will group the values

Figure 63: Utilization Line Graph for CPU vs. Customized Graph with `f(x) = x + 10`

in the requested series to make trend analysis easier. Figure 64 and Figure 65 illustrate the bucketed graph feature. Figure 64 shows the original graph for series CPU1, ranging from 0 to 75. Figure 65 shows the grouped bucketed graph after the user specifying the step as 10 and select "Stepwise Plot" in the operations field. The whole range is divided into 8 sub ranges and the data in each range is grouped.

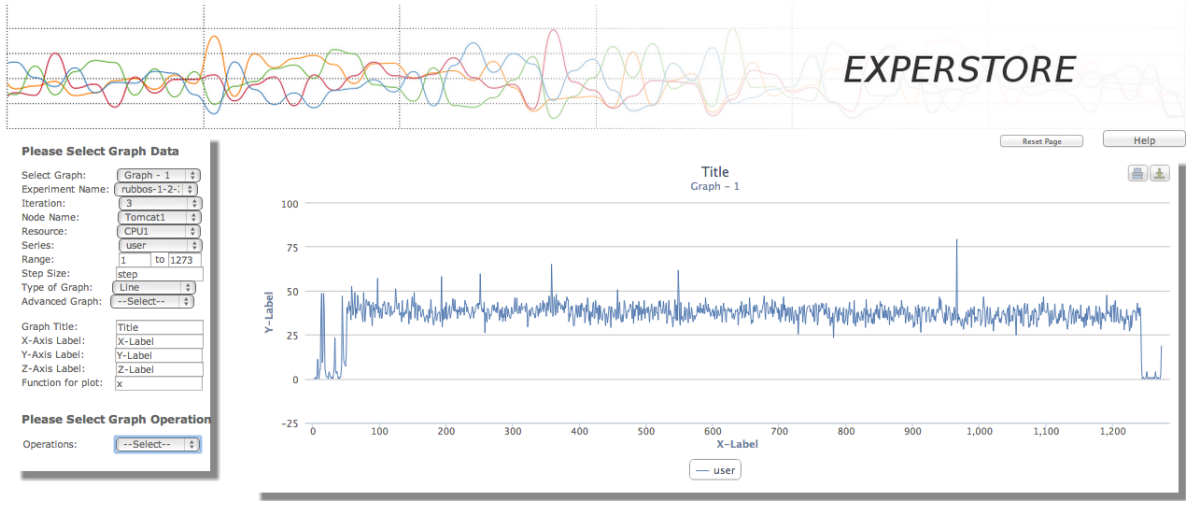


Figure 64: Utilization Line Graph for CPU1

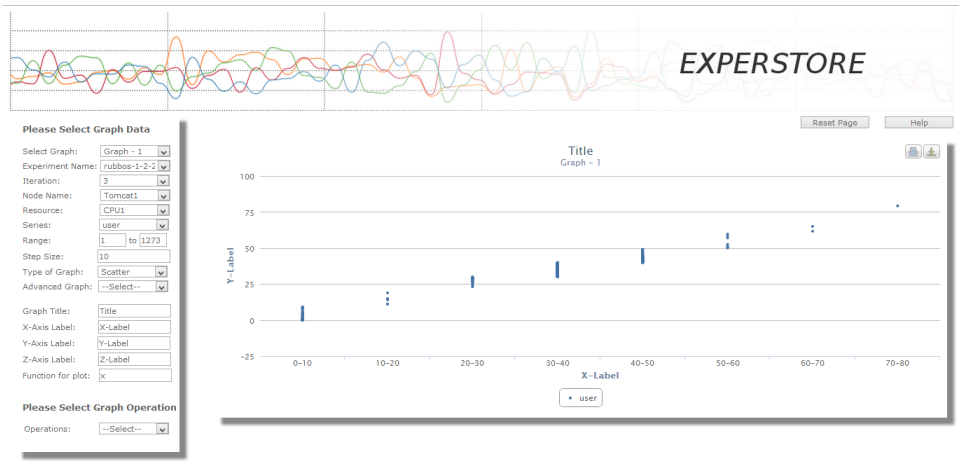


Figure 65: Utilization Bucketed Graph with Step as 10

C.3 Range based Frequency Analysis

Besides bucketed graphs, Experstore provides frequency data about number of points based on data ranges. The user can specify an arbitrary step for the range. Figure 66 and Figure 67 illustrate the frequency analysis feature. Figure 66 shows the original graph for series CPU1, ranging from 0 to 75. Figure 67 shows the frequency analysis graph after the user specifying the step as 10 and select ”Stepwise Frequency” in the operations field. The whole range is divided into 8 sub ranges and the data of how many points in each range is displayed.

Figure 66: Utilization Line Graph for CPU1

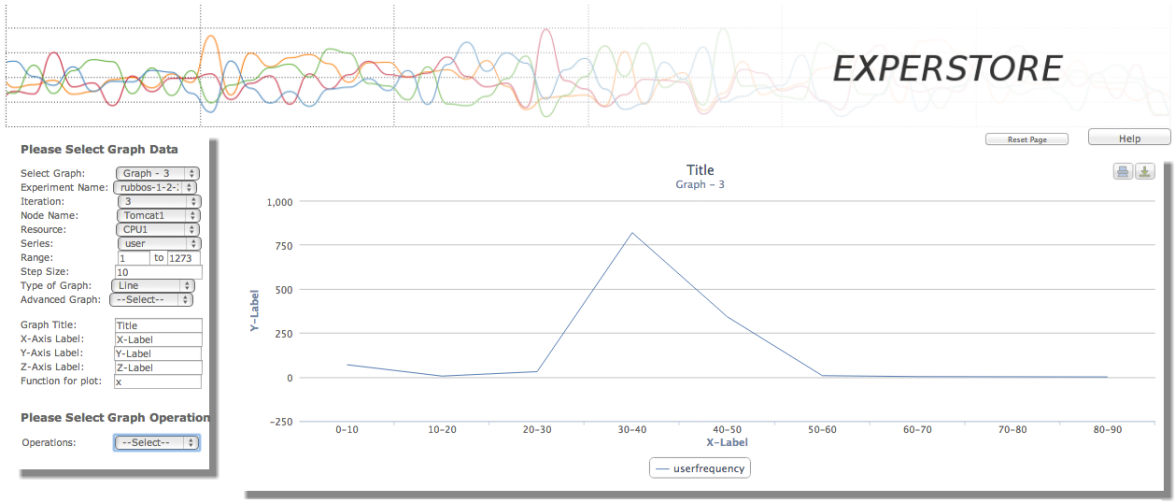


Figure 67: Frequency Analysis Graph with Step as 10

C.4 Correlation Analysis

Besides the analysis for individual series, Experstore also provides mutual correlation and disparity between multiple graphs so that trends can be identified easily. The user can select any two series and draw a correlative graph to view the correlations. Figure 68, Figure 69 and Figure 70 illustrate the correlation feature. Figure 68 shows the original graph for series CPU0, and Figure 69 shows the original graph for series CPU1. And Figure 70 shows the correlative graph between CPU0 and CPU1 with function `f(x) = x/y`, where x stands for data of CPU1, whereas y stands for data of CPU0.

Instead of using just one set of series, we ask user to input two series and then for all value of time we get data for both the series and then use the equation substitution mechanism, this time for x for first series, y for second series. Then we calculate the result and use it instead to plot the graph.

C.5 3-Dimensional graphs

Besides 2-Dimensional graphs, Experstore support 3-Dimensional graphs that can improve visualization power of the tool. Figure 71 and Figure 72 show the 3D graph feature.

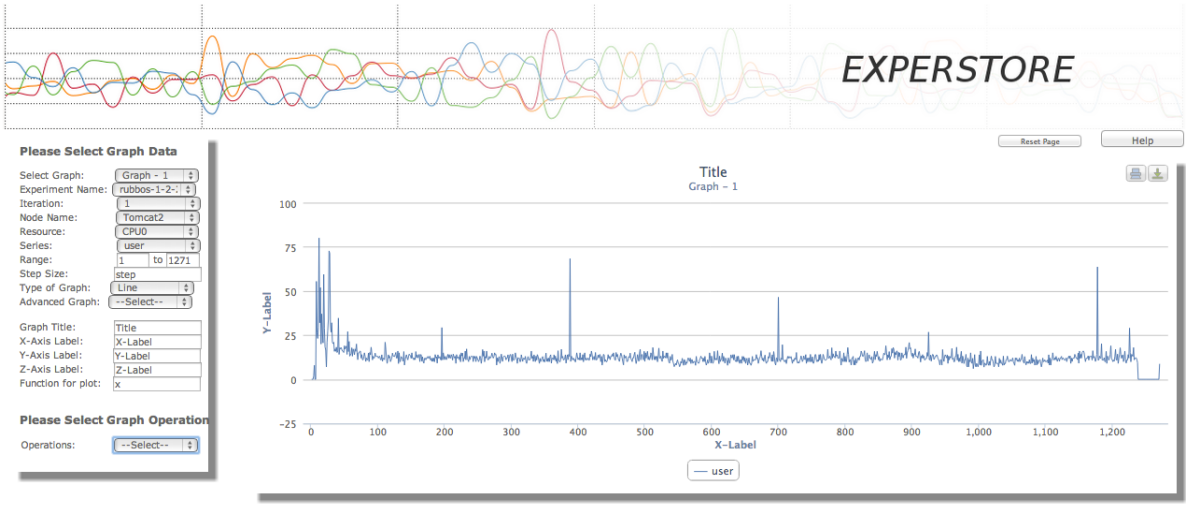


Figure 68: Utilization Line Graph for CPU0

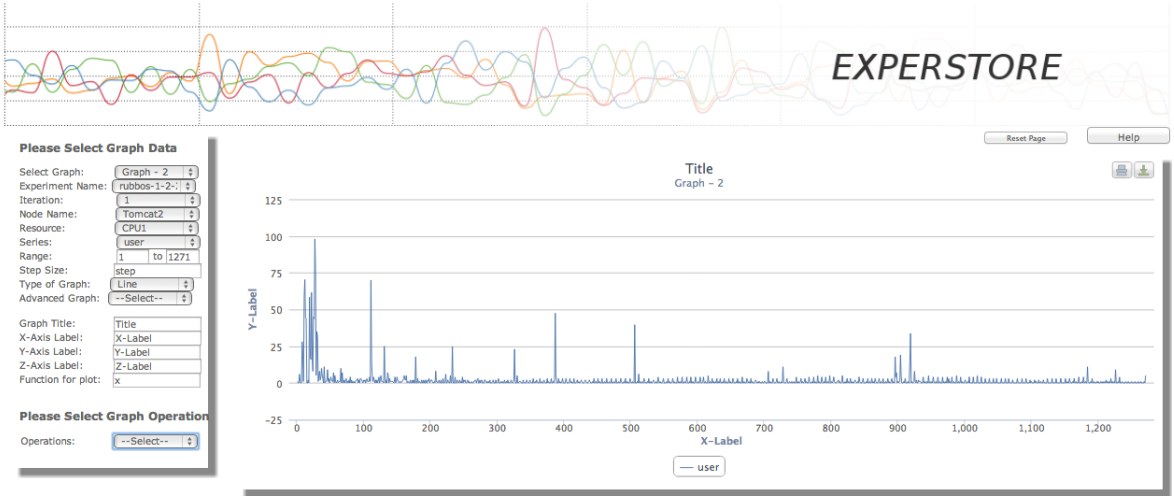


Figure 69: Utilization Line Graph for CPU1

C.6 Cumulative Graphs

As to the data analysis capabilities, Experstore supports the customized cumulative graphs for trend analysis with support for threshold based analysis. Comparing to standard graph, for each point in the graph, this feature will display the cumulative values based on all previous data instead of original value. Figure 73 and Figure 74 illustrate the cumulative graphs feature.

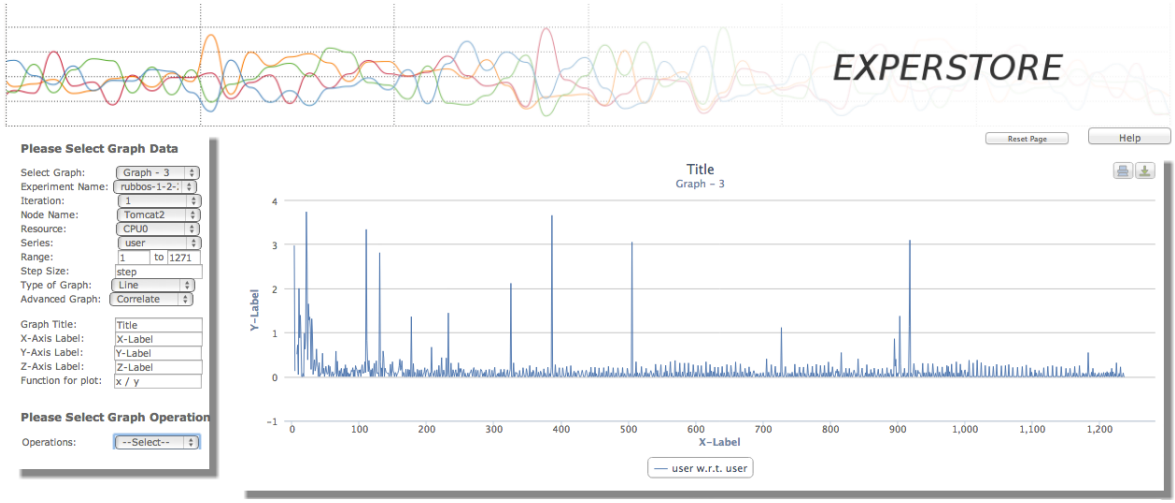


Figure 70: Correlative Graph with Series CPU0 and CPU1

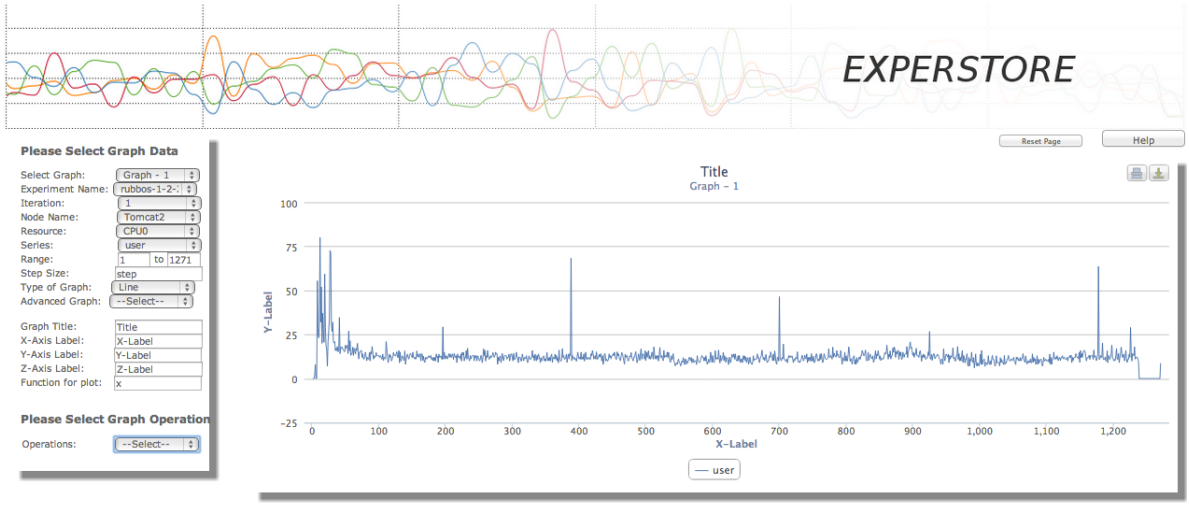


Figure 71: 2-Dimensional Graph for Series CPU0

C.7 Additional Features

The application has support for multiple graph types like line, spline, area, column, bar, scatter, pie and there many different variants. Most of these are compatible with each other in the fact that they require the same format of data input that is utilized for plotting the values. The application gives the capability to the user so that user can switch between different graph types without the need for providing the input data again. This greatly enhances the

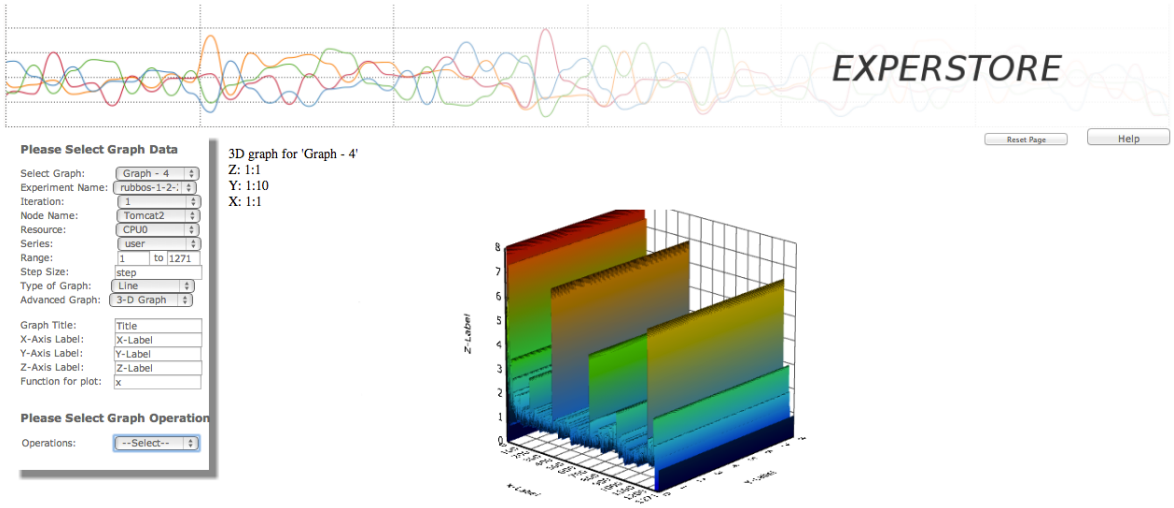


Figure 72: 3-Dimensional Graph for Series CPU0

Figure 73: Utilization Line Graph for Series CPU0

usability of the application. It also provides the user with the option of having multiple different types of graphs at the same time so that the comparison can be made as desired. In addition to the specific features discussed above, Experstore web portal supports many other features and some of them are listed below:

- Support for multiple graphs: A user has the option to have many different graphs of potentially different type on the same page so that he can compare the results between

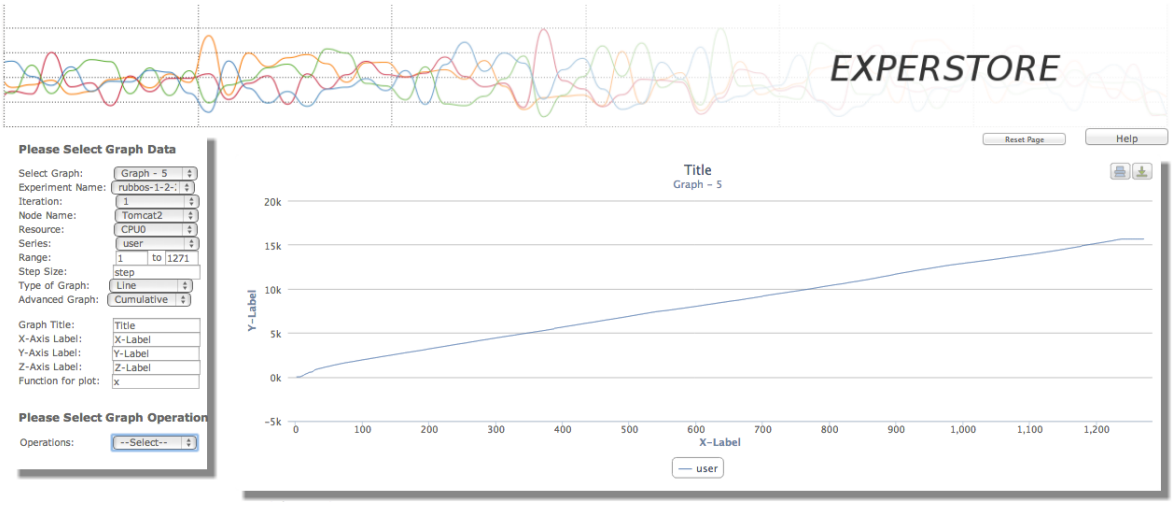


Figure 74: Cumulative Graph for Series CPU0

the experiments in the desired manner. This feature is especially useful when complex analysis tasks are performed.

- Simultaneous graphs: Even on the same graphs, the application permits the plot for different values simultaneously which aids in detailed relative comparisons to be done on the values from different sources.

- Zooming capability in graphs: Whenever a large amount of data is analyzed, there is always the scenario where one wishes to pick a part of the data and analyze if further. The application facilitates this task by giving the option to the user to zoom into a particular region of the graph and then view it in detailed. This could turn out to be really useful in case of studying an anomaly in the behavior of the system.

- Graph export support: The application provides user the option to export the current graph in form of an image file. Currently the graph support the option of export in JPEG, PNG and PDF formats.

- Graph printing support: For effective sharing, printing and then analyzing the graph could turn out to be very useful.

- Dynamic graphs: The use of dynamic graphs for understanding and monitoring the data in real time could be very useful. This is particularly useful when an ongoing experiment is to be monitored.

- Customizable features: Various features like axes labels, titles and range of the displayed data is fully customizable. Which helps the user in differentiating between graphs easily can could be very helpful for sharing of the information. It also incorporates the feature of modifying an existing graph and also removing any particular graph if the user wishes to do so.

- Auto-adjustable axes: The use of auto adjustable axes gives user the ability to fully utilize the visible space. The axes expand and shrink based on the maximum value that to be plotted by the graph.

REFERENCES

[1] "Amazon ec2 outage downs reddit, quora.." http://money.cnn.com/2011/04/21/technology/amazon_server_outage/index.htm.

[2] "Amazon's dec. 24th outage: A closer look.." http://www.informationweek.com/cloud-computing/infrastructure/amazons-dec-24th-outage-a-closer-look/240145533.

[3] "Apache http server project." http://httpd.apache.org/.

[4] "Apache tomcat." http://tomcat.apache.org/.

[5] "Automating .net application deployment and configuration.." http://www.rpath.com/corp/images/stories/white_papers/rPath_WP_Windows.pdf.

[6] "Deployment and configuration of component-based distributed applications .." http://www.omg.org/spec/DEPL/.

[7] "Distributed it configuration management.." http://cfengine.com.

[8] "Gartner." http://www.gartner.com/.

[9] "Ibm websphere cloudburst appliance.." http://www-01.ibm.com/software/webservers/cloudburst.

[10] "MySQL manuals." http://dev.mysql.com/doc/manual.

[11] "MySQL Cluster." http://www.mysql.com/products/database/cluster.

[12] "Novell platespin recon.." http://www.novell.com/prodcuts/recon/.

[13] "PostgreSQL." http://www.postgresql.org/.

[14] "Puppet data center automation solution.." http://www.puppetlabs.com/.

[15] "Slony-I." http://www.slony.info.

[16] "Smart framework for object groups.." http://www.smartfrog.org/.

[17] "Software testing automation framework.." http://staf.sourceforge.net.

[18] "A network protocol independent performance evaluator.." http://www.scl.ameslab.gov/netpipe/.

[19] "Animoto's Facebook Scale-up." http://blog.rightscale.com/2008/04/23/animoto-facebook-scale-up/.

[20] "EC2: Amazon Elastic Compute Cloud." http://aws.amazon.com/ec2.

[21] "Emulab: Network Emulation Testbed." http://www.emulab.net.

[22] "textscIBM autonomic computing." http://www.ibm.com/autonomic.

[23] "MySQL Cluster." http://www.mysql.com/products/database/cluster/.

[24] "Open Cirrus.." https://opencirrus.org/.

[25] "RUBBoS: Rice University Bulletin Board System." http://jmob.objectweb.org/rubbos.html.

[26] "RUBiS: Rice University Bidding System." http://rubis.objectweb.org/.

[27] "The Elba Project." http://www.cc.gatech.edu/systems/projects/Elba/.

[28] "Tools for performance analysis." http://www.bitmover.com/lmbench/.

[29] "Wipro technologies.." www.wipro.com/.

[30] "Vmware capacity planner." http://www.vmware.com/products/capacityplanner/.

[31] "The business rules group, defining business rules - what are they really?." http://www.businessrulesgroup.org, July 2000.

[32] *29th EUROMICRO Conference 2003, New Waves in System Architecture, 3-5 September 2003, Belek-Antalya, Turkey*, IEEE Computer Society, 2003.

[33] *Proceedings of the IEEE International Conference on Web Services (ICWS'04), June 6-9, 2004, San Diego, California, USA*, IEEE Computer Society, 2004.

[34] *2005 IEEE International Conference on Web Services (ICWS 2005), 11-15 July 2005, Orlando, FL, USA*, IEEE Computer Society, 2005.

[35] *Proceedings of the 12th International Workshop on Abstract State Machines, ASM 2005, March 8-11, 2005, Paris, France*, 2005.

[36] *Proceedings of the 1st International Conference on Collaborative Computing: Networking, Applications and Worksharing, San Jose, CA, USA, December 19-21, 2005*, IEEE, 2005.

[37] *15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2006), 26-28 June 2006, Manchester, United Kingdom*, IEEE Computer Society, 2006.

[38] *2006 IEEE International Conference on Services Computing (SCC 2006), 18-22 September 2006, Chicago, Illinois, USA*, IEEE Computer Society, 2006.

[39] *2007 IEEE International Conference on Granular Computing, GrC 2007, San Jose, California, USA, 2-4 November 2007*, IEEE, 2007.

[40] *31st Annual IEEE/NASA Software Engineering Workshop (SEW-31 2007), 6-8 March 2007, Loyola College, Columbia, MD, USA*, IEEE Computer Society, 2007.

[41] *International Conference on Internet and Web Applications and Services (ICIW 2007), May 13-19, 2007, Le Morne, Mauritius*, IEEE Computer Society, 2007.

[42] "jBPM." `http://www.jboss.org/jbossjbpm`, July 2008.

[43] "Workflow management coalition (WfMC) resource page." `http://www.wfmc.org`, September 2008.

[44] *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*, www.crdrdb.org, 2009.

[45] "Fujitsu SysViz: System Visualization." `"http://www.google.com/patents/US7873594"`, 2010.

[46] ABLE Project Team IBM T. J. Watson Research Center, ABLE *Rule Language, User's Guide and Reference, Version 2.0.1*, October 2003.

[47] Abramowicz, W. and Mayr, H. C., eds., *Business Information Systems, 9th International Conference on Business Information Systems, BIS 2006, May 31 - June 2, 2006, Klagenfurt, Austria*, vol. 85 of *LNI*, GI, 2006.

[48] Adams, K. and Agesen, O., "A comparison of software and hardware techniques for x86 virtualization," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS XII, (New York, NY, USA), pp. 2–13, ACM, 2006.

[49] Agarwala, S., Alegre, F., Schwan, K., and Mehalingham, J., "E2eprof: Automated end-to-end performance management for enterprise systems," in *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, (Washington, DC, USA), pp. 749–758, IEEE Computer Society, 2007.

[50] Aggar, M., "The IT energy efficiency imperative," white paper, Microsoft Corporation, June 2011.

[51] Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P., and Muthitacharoen, A., "Performance debugging for distributed systems of black boxes," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 74–89, ACM, 2003.

[52] Aiello, M., Aoyama, M., Curbera, F., and Papazoglou, M. P., eds., *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings*, ACM, 2004.

[53] Alexopoulos, C., "Statistical analysis of simulation output: state of the art," in *WSC '07: Proceedings of the 39th conference on Winter simulation*, (Piscataway, NJ, USA), pp. 150–161, IEEE Press, 2007.

[54] Altmann, J., Ion, M., Adel, A., and Mohammed, B., "A taxonomy of grid business models," in *Gecon2007, Intl. Workshop on Grid Economics and Business Models*, 2007.

[55] Amza, C., Cox, A. L., and Zwaenepoel, W., "A comparative evaluation of transparent scaling techniques for dynamic content servers," in *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, (Washington, DC, USA), pp. 230–241, IEEE Computer Society, 2005.

[56] Amza, C., Cecchet, E., Ch, A., Cox, A. L., Elnikety, S., Gil, R., Marguerite, J., Rajamani, K., and Zwaenepoel, W., "Bottleneck characterization of dynamic web site benchmarks." `http://www.cs.rice.edu/CS/Systems/DynaServer/dyna_bottleneck.pdf`, 2002.

[57] Amza, C., Ch, A., Cox, A. L., Elnikety, S., Gil, R., Rajamani, K., and Zwaenepoel, W., "Specification and implementation of dynamic web site benchmarks," in *In 5th IEEE Workshop on Workload Characterization*, pp. 3–13, 2002.

[58] Apparao, P., Makineni, S., and Newell, D., "Characterization of network processing overheads in xen," in *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*, pp. 2–2, 2006.

[59] Arlitt, M. and Jin, T., "A workload characterization study of the 1998 world cup web site," *Network, IEEE*, vol. 14, no. 3, pp. 30–37, 2000.

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

[61] Armbrust, M., Fox, A., Patterson, D. A., Lanham, N., Trushkowsky, B., Trutna, J., and Oh, H., "SCADS: Scale-independent storage for social computing applications," in *CIDR* [44].

[62] Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H., "Information and control in gray-box systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 43–56, 2001.

[63] Arshad, N., Heimbigner, D., and Wolf, A. L., "Dealing with failures during failure recovery of distributed systems," in *Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, DEAS '05, (New York, NY, USA), pp. 1–6, ACM, 2005.

[64] Avetisyan, A. I., Campbell, R., Gupta, I., Heath, M. T., Ko, S. Y., Ganger, G. R., Kozuch, M. A., O'Hallaron, D., Kunze, M., Kwan, T. T., Lai, K., Lyons, M.,

Milojicic, D. S., Lee, H. Y., Soh, Y. C., Ming, N. K., Luke, J.-Y., and Namgoong, H., "Open cirrus: A global cloud computing testbed," *Computer*, vol. 43, pp. 35–43, 2010.

[65] Babu, S., Borisov, N., Duan, S., Herodotou, H., and Thummala, V., "Automated experiment-driven management of (database) systems," in *Proceedings of the 12th conference on Hot topics in operating systems*, HotOS'09, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2009.

[66] Baeten, J. C. M. and Weijland, W. P., *Process algebra*. New York, NY, USA: Cambridge University Press, 1990.

[67] Balbo, G. and Serazzi, G., "Asymptotic analysis of multiclass closed queueing networks: multiple bottlenecks," *Perform. Eval.*, vol. 30, no. 3, pp. 115–152, 1997.

[68] Balsamo, S., Di Marco, A., Inverardi, P., and Simeoni, M., "Model-based performance prediction in software development: A survey," *IEEE Trans. Softw. Eng.*, vol. 30, pp. 295–310, May 2004.

[69] Barham, P., Donnelly, A., Isaacs, R., and Mortier, R., "Using magpie for request extraction and workload modelling," in *OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*, (Berkeley, CA, USA), pp. 18–18, USENIX Association, 2004.

[70] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A., "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, (New York, NY, USA), pp. 164–177, ACM, 2003.

[71] Basseville, M. and Nikiforov, I. V., *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc, 1993.

[72] Batini, C., Ceri, S., and Navathe, S. B., *Conceptual database design: an Entity-relationship approach*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1992.

[73] Baumgartner, R., Gatterbauer, W., and Gottlob, G., "Web data extraction system," 2009.

[74] Becker, S., Koziolek, H., and Reussner, R., "The palladio component model for model-driven performance prediction," *J. Syst. Softw.*, vol. 82, pp. 3–22, Jan. 2009.

[75] Benatallah, B., Casati, F., and Traverso, P., eds., *Service-Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005, Proceedings*, vol. 3826 of *Lecture Notes in Computer Science*, Springer, 2005.

[76] Bertino, E. and Joshi, J. B. D., eds., *Collaborative Computing: Networking, Applications and Worksharing, 4th International Conference, CollaborateCom 2008, Orlando, FL, USA, November 13-16, 2008, Revised Selected Papers*, vol. 10 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, 2008.

[77] Binnig, C., Kossmann, D., Kraska, T., and Loesing, S., "How is the weather tomorrow?: towards a benchmark for the cloud," in *Proceedings of the Second International Workshop on Testing Database Systems*, DBTest '09, (New York, NY, USA), pp. 9:1–9:6, ACM, 2009.

[78] Blake, R. and Breese, J. S., "Automatic bottleneck detection." `ftp://ftp.research.microsoft.com/pub/tr/tr-95-10.ps`, 1995.

[79] Bobroff, N., Kochut, A., and Beaty, K., "Dynamic placement of virtual machines for managing sla violations," in *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pp. 119–128, 2007.

[80] Bodic, P., Friedman, G., Biewald, L., Levine, H., Candea, G., Patel, K., Tolle, G., Hui, J., Fox, A., Jordan, M. I., and Patterson, D., "Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, (Washington, DC, USA), pp. 89–100, IEEE Computer Society, 2005.

[81] Bodík, P., Fox, O., Jordan, M. I., Patterson, D., Banerjee, A., Jagannathan, R., Su, T., Tenginakai, S., Turner, B., and Ingalls, J., "Advanced tools for operators at `Amazon.com`," in *In HotAC Workshop*, 2006.

[82] Bodík, P., Goldszmidt, M., and Fox, A., "Hilighter: Automatically building robust signatures of performance behavior," in *In 3rd Workshop on Tackling System Problems with Machine Learning Techniques (SysML'08)*, 2008.

[83] Bolognesi, T. and Brinksma, E., "Introduction to the ISO specification language LOTOS," *Comput. Netw. ISDN Syst.*, vol. 14, no. 1, pp. 25–59, 1987.

[84] Bordeaux, L. and Salaün, G., "Using process algebra for web services: Early results and perspectives," in Shan *et al.* [286], pp. 54–68.

[85] Bortnikov, V., Chockler, G., Roytman, A., and Spreitzer, M., "Bulletin board: a scalable and robust eventually consistent shared memory over a peer-to-peer overlay," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 64–70, Apr. 2010.

[86] Braghetto, K. R., Ferreira, J. E., and Pu, C., "Using control-flow patterns for specifying business processes in cooperative environments," in Cho *et al.* [113], pp. 1234–1241.

[87] Braghetto, K. R., Ferreira, J. E., and Pu, C., "Business processes management using process algebra and relational database model," in *ICE-B*, 2008.

[88] Braghetto, K. R., Ferreira, J. E., and Pu, C., "Using process algebra to control the execution of business processes," in Wainwright and Haddad [330], pp. 128–129.

[89] Bravetti, M., Kloul, L., and Zavattaro, G., eds., *Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings*, vol. 3670 of *Lecture Notes in Computer Science*, Springer, 2005.

[90] Brockwell, P. and Davis, R., *Introduction to time series and forecasting*. Springer texts in statistics, Springer, 2002.

[91] Brown, A., Hettena, D., Kuroda, J., Treuhaft, N., Patterson, D. A., and Yelick, K., "Roc-1: Hardware support for recovery-oriented computing," *IEEE Trans. Comput*, vol. 51, p. 2002, 2002.

[92] Bucur, S., Ureche, V., Zamfir, C., and Candea, G., "Parallel symbolic execution for automated real-world software testing," in *Proceedings of the sixth conference on Computer systems*, EuroSys '11, (New York, NY, USA), pp. 183–198, ACM, 2011.

[93] Buyya, R., Yeo, C. S., and Venugopal, S., "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, (Washington, DC, USA), pp. 5–13, IEEE Computer Society, 2008.

[94] C. Hyser, B. McKee, R. G. and Watson., B. J., "Autonomic virtual machine placement in the data center," tech report hpl-2007-189, October 2007.

[95] Cai, Y., Grundy, J., and Hosking, J., "Experiences integrating and scaling a performance test bed generator with an open source case tool," in *Proceedings of the 19th IEEE international conference on Automated software engineering*, ASE '04, (Washington, DC, USA), pp. 36–45, IEEE Computer Society, 2004.

[96] Camargos, F. L., Girard, G., and Ligneris, B. D., "Virtualization of Linux servers: a comparative study," in *2008 Ottawa Linux Symsposium*, pp. 63–76, July 2008.

[97] Candea, G., Cutler, J., Fox, A., Doshi, R., Garg, P., and Gowda, R., "Reducing recovery time in a small recursively restartable system," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 605–614, 2002.

[98] Candea, G., Bucur, S., and Zamfir, C., "Automated software testing as a service," in *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, (New York, NY, USA), pp. 155–160, ACM, 2010.

[99] Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G., and Fox, A., "Microreboot — a technique for cheap recovery," in *Proceedings of the 6th conference on*

Symposium on Opearting Systems Design & Implementation - Volume 6, OSDI'04, (Berkeley, CA, USA), pp. 3–3, USENIX Association, 2004.

[100] Casale, G., Mi, N., Cherkasova, L., and Smirni, E., "How to parameterize models with bursty workloads," in *Proc. of First Workshop on Hot Topics in Measurement & Modeling of Computer Systems (HotMetrics '08)*, (Annapolis, MD), June 2008.

[101] Casale, G. and Serazzi, G., "Bottlenecks identification in multiclass queueing networks using convex polytopes," in *MASCOTS '04: Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, (Washington, DC, USA), pp. 223–230, IEEE Computer Society, 2004.

[102] Cecchet, E., Candea, G., and Ailamaki, A., "Middleware-based Database Replication: The Gaps Between Theory and Practice," in *ACM SIGMOD'08*, 2008.

[103] Cecchet, E., Ch, A., Elnikety, S., Marguerite, J., and Zwaenepoel, W. in *In Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*.

[104] Cecchet, E., Marguerite, J., and Zwaenepoel, W., "Partial replication: Achieving scalability in redundant arrays of inexpensive databases," *Lecture Notes in Computer Science*, vol. 3144, pp. 58–70, July 2004.

[105] Cecchet, E., Marguerite, J., and Zwaenepoel, W., "C-JDBC: Flexible database clustering middleware," in *In Proceedings of the USENIX 2004 Annual Technical Conference*, pp. 9–18, 2004.

[106] Chaisiri, S., Lee, B.-S., and Niyato, D., "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pp. 103–110, 2009.

[107] Chakrabarti, S., Sarawagi, S., and Dom, B., "Mining surprising patterns using temporal description length," in *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, (San Francisco, CA, USA), pp. 606–617, Morgan Kaufmann Publishers Inc., 1998.

[108] Chang, F., Ren, J., and Viswanathan, R., "Optimal resource allocation in clouds," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 418–425, 2010.

[109] Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., and Doyle, R. P., "Managing energy and server resources in hosting centers," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, (New York, NY, USA), pp. 103–116, ACM, 2001.

[110] Chen, Y., Wo, T., and Li, J., "An efficient resource management system for on-line virtual cluster provision," in *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, pp. 72–79, 2009.

[111] Chirichiello, A. and Salaün, G., "Encoding abstract descriptions into executable web services: Towards a formal development," in Skowron *et al.* [291], pp. 457–463.

[112] Chirichiello, A. and Salaun, G., "Encoding process algebraic descriptions of web services into BPEL," *Web Intelligence and Agent Systems*, vol. 5, no. 4, pp. 419–434, 2007.

[113] Cho, Y., Wainwright, R. L., Haddad, H., Shin, S. Y., and Koo, Y. W., eds., *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007*, ACM, 2007.

[114] Chun, B. N., "Dart: distributed automated regression testing for large-scale network applications," in *Proceedings of the 8th international conference on Principles of Distributed Systems*, OPODIS'04, (Berlin, Heidelberg), pp. 20–36, Springer-Verlag, 2005.

[115] Clark, B., Deshane, T., Dow, E., Evanchik, S., Finlayson, M., Herne, J., and Matthews, J. N., "Xen and the art of repeated research," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, (Berkeley, CA, USA), pp. 47–47, USENIX Association, 2004.

[116] Clemm, A., Granville, L. Z., and Stadler, R., eds., *Managing Virtualization of Networks and Services, 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2007, San José, CA, USA, October 29-31, 2007, Proceedings*, vol. 4785 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), Springer-Verlag, 2007.

[117] Cohen, I., Goldszmidt, M., Kelly, T., Symons, J., and Chase, J. S., "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*, (Berkeley, CA, USA), pp. 16–16, USENIX Association, 2004.

[118] Cohen, I., Zhang, S., Goldszmidt, M., Symons, J., Kelly, T., and Fox, A., "Capturing, indexing, clustering, and retrieving system history," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 105–118, 2005.

[119] Corporation, I. D., "Cloud computing 2010: An IDC update." `http://www.slideshare.net/JorFigOr/cloud-computing-2010-an-idc-update`, 2010.

[120] Curino, C., Jones, E. P., Madden, S., and Balakrishnan, H., "Workload-aware database monitoring and consolidation," in *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, (New York, NY, USA), pp. 313–324, ACM, 2011.

[121] Dai, Y., Xiang, Y., and Zhang, G., "Self-healing and hybrid diagnosis in cloud computing," in *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, (Berlin, Heidelberg), pp. 45–56, Springer-Verlag, 2009.

[122] Davies, J. and Gibbons, J., eds., *Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings*, vol. 4591 of *Lecture Notes in Computer Science*, Springer, 2007.

[123] de Assuncao, M. D., di Costanzo, A., and Buyya, R., "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, (New York, NY, USA), pp. 141–150, ACM, 2009.

[124] de Meer, H., Trivedi, K. S., and Dal Cin, M., "Guarded repair of dependable systems," *Theor. Comput. Sci.*, vol. 128, pp. 179–210, June 1994.

[125] Deepal Jayasinghe, Fábio Oliveira, F. R. C. P. and Eilam., T., "Aeson: A model-driven and fault tolerant composite deployment runtime for iaas clouds," in *Services Computing (SCC), 2013 IEEE International Conference on*, 2011.

[126] Deepal Jayasinghe, Simon Malkowski, J. L. Q. W. Z. W. and Pu, C., "Variations in performance and scalability: An experimental study in iaas clouds using multi-tier workloads," *Submitted to Service Computing, IEEE Transactions on*, 2013.

[127] Deepal Jayasinghe, Josh Kimball, S. C. T. Z. and Pu., C., "An automated approach to create, store, and analyze large-scale experimental data in clouds," in *Submitted to IEEE IRI 2013*, 2013.

[128] Denning, P. J. and Buzen, J. P., "The operational analysis of queueing network models," *ACM Comput. Surv.*, vol. 10, no. 3, pp. 225–261, 1978.

[129] Dumas, M., van der Aalst, W. M., and ter Hofstede, A. H., *Process-aware information systems: bridging people and software through process technology*. New York, NY, USA: John Wiley & Sons, Inc., 2005.

[130] Economist, T.

[131] Economou, D., Rivoire, S., and Kozyrakis, C., "Full-system power analysis and modeling for server environments," in *In Workshop on Modeling Benchmarking and Simulation (MOBS*, 2006.

[132] Ehrig, H. and Mahr, B., *Fundamentals of Algebraic Specification I*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1985.

[133] Eilam, T., Elder, M., Konstantinou, A., and Snible, E., "Pattern-based composite application deployment," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pp. 217–224, 2011.

[134] El Maghraoui, K., Meghranjani, A., Eilam, T., Kalantar, M., and Konstantinou, A. V., "Model driven provisioning: bridging the gap between declarative object models and procedural provisioning tools," in *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, (New York, NY, USA), pp. 404–423, Springer-Verlag New York, Inc., 2006.

[135] Elmasri, R. and Navathe, S. B., *Fundamentals of Database Systems (5th Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[136] Elnikety, S., Dropsho, S., and Pedone, F., "Tashkent: uniting durability with transaction ordering for high-performance scalable database replication," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 4, pp. 117–130, 2006.

[137] Fahland, D. and Reisig, W., "ASM-based semantics for BPEL: The negative control flow," in *Abstract State Machines* [35], pp. 131–152.

[138] Feitelson, D., "Workload modeling for computer systems performance evaluation." `http://www.cs.huji.ac.il/~feit/wlmod/`, 2011.

[139] Feitelson, D. G., "Experimental computer science: Experimental computer science: The need for a cultural change." White paper, `http://www.cs.huji.ac.il/~feit/papers/exp05.pdf`, December 2006.

[140] Ferrara, A., "Web services: a process algebra approach," in Aiello *et al.* [52], pp. 242–251.

[141] Ferreira, J. E., Malkowski, S., Takai, O., and Pu, C., "Transactional recovery support for robust workflows through automated exception handling," in *under review at The 37th International Conference on Very Large Data Bases (PVLDB)*, 2011.

[142] Ferreira, J. E., Takai, O. K., Braghetto, K. R., and Pu, C., "Large scale order processing through navigation plan concept," in *IEEE SCC* [38], pp. 297–300.

[143] Ferreira, J. E., Takai, O. K., Malkowski, S., and Pu, C., "Reducing exception handling complexity in business process modeling and implementation: The wedflow approach," in Meersman *et al.* [223], pp. 150–167.

[144] Ferreira, J. E., Takai, O. K., and Pu, C., "Integration of collaborative information system in internet applications using riverfish architecture," in *CollaborateCom* [36].

[145] Ferreira, J. E., Wu, Q., Malkowski, S., and Pu, C., "Towards flexible event-handling in workflows through data states," *Services, IEEE Congress on*, vol. 0, pp. 344–351, 2010.

[146] Ferreto, T. C., Netto, M. A. S., Calheiros, R. N., and De Rose, C. A. F., "Server consolidation with migration control for virtualized data centers," *Future Gener. Comput. Syst.*, vol. 27, pp. 1027–1034, October 2011.

[147] Fisteus, J. A., Fernández, L. S., and Kloos, C. D., "Formal verification of BPEL4WS business collaborations," *E-Commerce and Web Technologies*, pp. 76–85, 2004.

[148] Fokkink, W., *Introduction to Process Algebra*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000.

[149] Foster, H., Uchitel, S., Magee, J., and Kramer, J., "Tool support for model-based engineering of web service compositions," in *ICWS* [34], pp. 95–102.

[150] Gmach, D., Krompass, S., Scholz, A., Wimmer, M., and Kemper, A., "Adaptive quality of service management for enterprise services," *ACM Trans. Web*, vol. 2, no. 1, pp. 1–46, 2008.

[151] Gong, Z. and Gu, X., "PAC: Pattern-driven application consolidation for efficient cloud computing," *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on*, vol. 0, pp. 24–33, 2010.

[152] Gray, J., Helland, P., O'Neil, P., and Shasha, D., "The dangers of replication and a solution," in *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 173–182, ACM, 1996.

[153] Grit, L., Irwin, D., Yumerefendi, A., and Chase, J., "Virtual machine hosting for networked clusters: Building the foundations for "autonomic" orchestration," in *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, VTDC '06, (Washington, DC, USA), pp. 7–, IEEE Computer Society, 2006.

[154] Grundy, J., Cai, Y., and Liu, A., "Softarch/mte: Generating distributed system testbeds from high-level software architecture descriptions," *Automated Software Engg.*, vol. 12, pp. 5–39, Jan. 2005.

[155] Gustafsson, F., *Adaptive Filtering and Change Detection*. John Wiley & Sons, Inc., 2001.

[156] Haddad, S., Melliti, T., Moreaux, P., and Rampacek, S., "Modelling web services interoperability," in *ICEIS (4)*, pp. 287–295, 2004.

[157] Halle, B. V., *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. New York, NY, USA: John Wiley & Sons, Inc., 2001. Foreword By-Ronald G. Ross.

[158] Hanemann, A., Schmitz, D., and Sailer, M., "A framework for failure impact analysis and recovery with respect to service level agreements," in *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, (Washington, DC, USA), pp. 49–58, IEEE Computer Society, 2005.

[159] Hanson, B. E., "Bandwidth selection for nonparametric distribution estimation." `http://www.ssc.wisc.edu/~bhansen/papers/wp.htm`, May 2004.

[160] Harizopoulos, S., Abadi, D. J., Madden, S., and Stonebraker, M., "Oltp through the looking glass, and what we found there," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 981–992, ACM, 2008.

[161] Heckel, R. and Lohmann, M., "Towards contract-based testing of web services," *Electronic Notes in Theoretical Computer Science*, vol. 116, pp. 145–156, 2005.

[162] Hedwig, M., Malkowski, S., Bodenstein, C., and Neumann, D., "Datacenter investment support system (DAISY)," in *HICSS '10: Proceedings of the 43nd Hawaii International Conference on System Sciences*, (Washington, DC, USA), IEEE Computer Society, 2009.

[163] Hedwig, M., Malkowski, S., and Neumann, D., "Taming energy costs of large enterprise systems through adaptive provisioning," in *ICIS '09: Proceedings of the Eight IEEE/ACIS International Conference on Computer and Information Science*, (Phoenix, AZ, USA), IEEE Computer Society, 2009.

[164] Hedwig, M., Malkowski, S., and Neumann, D., "Towards autonomic cost-aware allocation of cloud resources," in *ICIS '10: International Conference on Information Systems*, 2010.

[165] Henzinger, T. A., Singh, A. V., Singh, V., Wies, T., and Zufferey, D., "Flexprice: Flexible provisioning of resources in a cloud environment," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, (Washington, DC, USA), pp. 83–90, IEEE Computer Society, 2010.

[166] Hofstede, A. H. M. T. and Weske, M., "Business process management: A survey," in *Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS*, pp. 1–12, Springer-Verlag, 2003.

[167] Huang, C., Cohen, I., Symons, J., and Abdelzaher, T., "Achieving scable autoamted diagnosis of distributed systems performance problems," tech. rep., HP Labs, 2007.

[168] Hull, R. and Su, J., "Tools for design of composite web services," in Weikum *et al.* [339], pp. 958–961.

[169] Inc., A., "Amazon web services. cloudformation." `http://aws.amazon.com/cloudformation/`, 2010.

[170] Ioannidis, Y., Livny, M., Gupta, S., and Ponnekanti, N., "Zoo: A desktop experiment management environment," in *In Proc. 22nd International VLDB Conference*, pp. 274–285, Morgan Kaufmann, 1996.

[171] ISO, "Information processing systems–open systems interconnection–lotos–a formal description technique based on the temporal ordering of observational behavior." DIS 8807, 1986.

[172] Jain, R., *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* New York, NY, USA: John Wiley & Sons, Inc., 1991.

[173] Jayasinghe, D., Pu, C., Eilam, T., Steinder, M., Whally, I., and Snible, E., "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Services Computing (SCC), 2011 IEEE International Conference on*, pp. 72–79, 2011.

[174] Jayasinghe, D., Malkowski, S., Wang, Q., Li, J., Xiong, P., and Pu, C., "Variations in performance and scalability when migrating n-tier applications to different clouds," in *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, (Washington, DC, USA), pp. 73–80, IEEE Computer Society, 2011.

[175] Jayasinghe, D., Swint, G., Malkowski, S., Li, J., Wang, Q., Park, J., and Pu, C., "Expertus: A generator approach to automate performance testing in iaas clouds," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, (Washington, DC, USA), pp. 115–122, IEEE Computer Society, 2012.

[176] Jordan, D. and Evdemon, J., "Web services business process execution language version 2.0.." Public Review Draft OASIS WS-BPEL Technical Committee, April 2007.

[177] Joshi, K. R., Sanders, W. H., Hiltunen, M. A., and Schlichting, R. D., "Automatic model-driven recovery in distributed systems," in *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, SRDS '05, (Washington, DC, USA), pp. 25–38, IEEE Computer Society, 2005.

[178] Jung, G., *Multi-dimensional optimization for cloud based multi-tier applications*. PhD thesis, Georgia Institute of Technology, Atlanta, Georgia, USA, 2010.

[179] Jung, G., Hiltunen, M. A., Joshi, K. R., Schlichting, R. D., and Pu, C., "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *ICDCS '10: Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, (Washington, DC, USA), pp. 62–73, IEEE Computer Society, 2010.

[180] Jung, G., Joshi, K., Hiltunen, M., Schlichting, R., and Pu, C., "Generating adaptation policies for multi-tier applications in consolidated server environments," in *ICAC '08: Proceedings of the 2008 International Conference on Autonomic Computing*, (Washington, DC, USA), pp. 23–32, IEEE Computer Society, 2008.

[181] Jung, G., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., and Pu, C., "A cost-sensitive adaptation engine for server consolidation of multitier applications," in *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, (New York, NY, USA), pp. 1–20, Springer-Verlag New York, Inc., 2009.

[182] Jung, G., Joshi, K., Hiltunen, M., Schlichting, R., and Pu, C., "Performance and availability aware regeneration for cloud based multitier applications," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pp. 497–506, 2010.

[183] Jung, G., Pu, C., and Swint, G., "Mulini: an automated staging framework for QoS of distributed multi-tier applications," in *WRASQ '07: Proceedings of the 2007 workshop on Automating service quality*, (New York, NY, USA), pp. 10–15, ACM, 2007.

[184] Jung, G., Swint, G., Parekh, J., Pu, C., and Sahai, A., "Detecting bottleneck in n-tier it applications through analysis," *Large Scale Management of Distributed Systems*, pp. 149–160, 2006.

[185] Karavanic, K. L. and Miller, B. P., "Experiment management support for performance tuning," in *Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '97, (New York, NY, USA), pp. 1–10, ACM, 1997.

[186] Katti, S., Katabi, D., Blake, C., Kohler, E., and Strauss, J., "Multiq: automated detection of multiple bottleneck capacities along a path," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, (New York, NY, USA), pp. 245–250, ACM, 2004.

[187] Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., and Barreto, C., "Web services choreography description language version 1.0." W3C Candidate Recommendation, November 2005.

[188] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., marc Loingtier, J., and Irwin, J., "Aspect-oriented programming," in *ECOOP*, SpringerVerlag, 1997.

[189] Kim, K., Jeon, K., Han, H., Kim, S.-g., Jung, H., and Yeom, H. Y., "Mrbench: A benchmark for mapreduce framework," in *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, (Washington, DC, USA), pp. 11–18, IEEE Computer Society, 2008.

[190] Kimbrel, T., Steinder, M., Sviridenko, M., and Tantawi, A., "Dynamic application placement under service and memory constraints," in *Proceedings of the 4th international conference on Experimental and Efficient Algorithms*, WEA'05, (Berlin, Heidelberg), pp. 391–402, Springer-Verlag, 2005.

[191] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C., "An Analysis of Performance Interference Effects in Virtual Environments," in *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pp. 200–209, IEEE, Apr. 2007.

[192] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C., "An analysis of performance interference effects in virtual environments," in *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2007.

[193] Koh, Y., Pu, C., Shinjo, Y., Eiraku, H., Saito, G., and Nobori, D., "Improving virtualized windows network performance by delegating network processing," in *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, pp. 203–210, 2009.

[194] Kohavi, R., Henne, R. M., and Sommerfield, D., "Practical guide to controlled experiments on the web: listen to your customers not to the hippo," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, (New York, NY, USA), pp. 959–967, ACM, 2007.

[195] Kossmann, D. and Kraska, T., "Data management in the cloud: Promises, state-of-the-art, and open questions," *Datenbank-Spektrum*, vol. 10, pp. 121–129, 2010. 10.1007/s13222-010-0033-3.

[196] Kossmann, D., Kraska, T., and Loesing, S., "An evaluation of alternative architectures for transaction processing in the cloud," in *SIGMOD Conference* (Elmagarmid, A. K. and Agrawal, D., eds.), pp. 579–590, ACM, 2010.

[197] Kraft, S., Casale, G., Krishnamurthy, D., Greer, D., and Kilpatrick, P., "Performance models of storage contention in cloud environments," *Software & Systems Modeling*, pp. 1–24, 2012.

[198] Krauter, K., Buyya, R., and Maheswaran, M., "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Pract. Exper.*, vol. 32, no. 2, pp. 135–164, 2002.

[199] Lacour, S., Perez, C., and Priol, T., "Generic application description model: Toward automatic deployment of applications on computational grids," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, (Washington, DC, USA), pp. 284–287, IEEE Computer Society, 2005.

[200] Lampson, B. W., "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[201] Li, B., Li, J., Huai, J., Wo, T., Li, Q., and Zhong, L., "Enacloud: An energy-saving application live placement approach for cloud computing environments," *2012 IEEE Fifth International Conference on Cloud Computing*, vol. 0, pp. 17–24, 2009.

[202] Li, J., Chinneck, J., Woodside, M., Litoiu, M., and Iszlai, G., "Performance model driven qos guarantees and optimization in clouds," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, (Washington, DC, USA), pp. 15–22, IEEE Computer Society, 2009.

[203] Li, J., He, J., Zhu, H., and Pu, G., "Modeling and verifying web services choreography using process algebra," in *SEW* [40], pp. 256–268.

[204] Lilja, D. J., *Measuring Computer Performance - A Practitioner's Guide*. New York, NY, USA: Cambridge University Press, 2000.

[205] Lim, C., Singh, N., and Yajnik, S., "A log mining approach to failure analysis of enterprise telephony systems," in *DSN '08: Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, (Washington, DC, USA), pp. 398–403, IEEE Computer Society, 2008.

[206] Lim, H. C., Babu, S., and Chase, J. S., "Automated control for elastic storage," in *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, (New York, NY, USA), pp. 1–10, ACM, 2010.

[207] Linden, G., "Make your data useful." http://home.blarg.net/~glinden/StanfordDataMining.2006-11-29.ppt, November 2006.

[208] Litoiu, M., "A performance analysis method for autonomic computing systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 2, March 2007.

[209] Luthi, J., "Interval matrices for the bottleneck analysis of queuing network models with histogram based parameters," in *In IEEE International Computer Performance & Dependability Symposium*, pp. 142–151, IEEE Computer Society Press, 1998.

[210] Machida, F., Kawato, M., and Maeno, Y., "Redundant virtual machine placement for fault-tolerant consolidated server clusters," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pp. 32–39, 2010.

[211] Malkowski, S., *An Emprical Approach to Automate Performance Management for Elastic n-Tier Applications in Computing Cloud*. PhD thesis, Georgia Institute of Technology, Atlanta, Georgia, USA, 2012.

[212] Malkowski, S., Hedwig, M., Jayasinghe, D., Park, J., Kanemasa, Y., and Pu, C., "A new perspective on experimental analysis of n-tier systems: Evaluating database scalability, multi-bottlenecks, and economical operation," in *CollaborateCom '09*, (Washington, DC, USA), IEEE Computer Society, 2009.

[213] Malkowski, S., Hedwig, M., Jayasinghe, D., Pu, C., and Neumann, D., "CloudXplor: A tool for configuration planning in clouds based on empirical data," in *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*, (New York, NY, USA), ACM, 2010.

[214] Malkowski, S., Hedwig, M., Li, J., Pu, C., and Neumann, D., "Automated control for elastic n-tier workloads based on empirical modeling," in *The 8th International Conference on Autonomic Computing (ICAC)*, 2011.

[215] Malkowski, S., Hedwig, M., Parekh, J., Pu, C., and Sahai, A., "Bottleneck detection using statistical intervention analysis," in Clemm *et al.* [116], pp. 122–134.

[216] Malkowski, S., Hedwig, M., Parekh, J., Pu, C., and Sahai, A., "Bottleneck detection using statistical intervention analysis," in *Proceedings of the Distributed systems: operations and management 18th IFIP/IEEE international conference on Managing virtualization of networks and services*, DSOM'07, (Berlin, Heidelberg), pp. 122–134, Springer-Verlag, 2007.

[217] Malkowski, S., Hedwig, M., and Pu, C., "Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks," in *IISWC '09: Proceedings of the 2009 IEEE 12th International Symposium on Workload Characterization*, (Austin, TX, USA), IEEE Computer Society, 2009.

[218] Malkowski, S., Jayasinghe, D., Hedwig, M., Park, J., Kanemasa, Y., and Pu, C., "Empirical analysis of database server scalability using an n-tier benchmark with

read-intensive workload," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (New York, NY, USA), pp. 1680–1687, ACM, 2010.

[219] Malkowski, S., Jayasinghe, D., Hedwig, M., Park, J., Kanemasa, Y., and Pu, C., "Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload," in *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*, (New York, NY, USA), ACM, 2010.

[220] Malkowski, S., Kanemasa, Y., Chen, H., Yamamoto, M., Jayasinghe, Q. W. D., Pu, C., and Kawaba, M., "Challenges and opportunities in consolidation at high resource utilization: Non-monotonic response time variations in n-tier applications," in *CLOUD, 2012 IEEE 5th International Conference on*, 2012.

[221] Manku, G. S. and Motwani, R., "Approximate frequency counts over data streams," in *In VLDB*, pp. 346–357, 2002.

[222] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., and McIlraith, S., "OWL-S: Semantic markup for web services." W3C Member Submission, November 2004.

[223] Meersman, R., Dillon, T. S., and Herrero, P., eds., *On the Move to Meaningful Internet Systems: OTM 2010 - Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, October 25-29, 2010, Proceedings, Part I*, vol. 6426 of *Lecture Notes in Computer Science*, Springer, 2010.

[224] Menascé, D. A., "Virtualization: Concepts, Applications, and Performance Modeling," in *Computer Measurement Group Conference*, pp. 407–414, 2005.

[225] Menascé, D. A. and Bennani, M. N., "Autonomic Virtualized Environments," in *International Conference on Autonomic and Autonomous Systems*, 2006.

[226] Menascé, D. A., Almeida, V. A. F., and Dowdy, L. W., *Capacity planning and performance modeling: from mainframes to client-server systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[227] Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E., and Pendarakis, D., "Efficient resource provisioning in compute clouds via vm multiplexing," in *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, (New York, NY, USA), pp. 11–20, ACM, 2010.

[228] Meng, X., Pappas, V., and Zhang, L., "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of the 29th conference on Information communications*, INFOCOM'10, (Piscataway, NJ, USA), pp. 1154–1162, IEEE Press, 2010.

[229] Mi, N., "Performance impacts of autocorrelated flows in multi-tiered systems," *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 3, pp. 44–45, 2007.

[230] Mi, N., Casale, G., Cherkasova, L., and Smirni, E., "Burstiness in multi-tier applications: symptoms, causes, and new models," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, (New York, NY, USA), pp. 265–286, Springer-Verlag New York, Inc., 2008.

[231] Mi, N., Casale, G., Cherkasova, L., and Smirni, E., "Injecting realistic burstiness to a traditional client-server benchmark," in *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, (New York, NY, USA), pp. 149–158, ACM, 2009.

[232] Microsoft, "Exchange server documentation." `http://technet.microsoft.com/en-us/library/dd298121.aspx`, April 2010.

[233] Milan-Franco, J. M., Jimenez-Peris, R., Patino-Martinez, M., and Kemme, B., "Adaptive middleware for data replication," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, (New York, NY, USA), pp. 175–194, Springer-Verlag New York, Inc., 2004.

[234] Milner, R., *Communication and concurrency*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[235] Milner, R., *Communicating and mobile systems: the π-calculus*. New York, NY, USA: Cambridge University Press, 1999.

[236] Moss, M., "Comparing centralized and distributed approaches for operational impact analysis in enterprise systems," in *GrC* [39], pp. 765–769.

[237] Moss, M. and Pu, C., "Assessing operational impact in enterprise systems by mining usage patterns," in Clemm *et al.* [116], pp. 159–170.

[238] Motta, E., *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1999.

[239] Muditha Perera, P. and Keppitiyagama, C., "A performance comparison of hypervisors," in *Advances in ICT for Emerging Regions (ICTer), 2011 International Conference on*, pp. 120–120, 2011.

[240] Nagarajan, A. B., Mueller, F., Engelmann, C., and Scott, S. L., "Proactive fault tolerance for hpc with xen virtualization," in *Proceedings of the 21st annual international conference on Supercomputing*, ICS '07, (New York, NY, USA), pp. 23–32, ACM, 2007.

[241] Nathuji, R., Kansal, A., and Ghaffarkhah, A., "Q-clouds: managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, (New York, NY, USA), pp. 237–250, ACM, 2010.

[242] Nguyen Van, H., Dang Tran, F., and Menaud, J.-M., "Autonomic virtual resource management for service hosting platforms," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, (Washington, DC, USA), pp. 1–8, IEEE Computer Society, 2009.

[243] Oikawa, M. K., Ferreira, J. E., Malkowski, S., and Pu, C., "Towards algorithmic generation of business processes: From business step dependencies to process algebra expressions," in *BPM '09: Proceedings of the 7th International Conference on Business Process Management*, (Berlin, Heidelberg), pp. 80–96, Springer-Verlag, 2009.

[244] Oliveira, F., Eilam, T., Kalantar, M., and Rosenberg, F., "Semantically-rich composition of virtual images," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, (Washington, DC, USA), pp. 277–284, IEEE Computer Society, 2012.

[245] Ouyang, C., Verbeek, E., van der Aalst, W. M. P., Breutel, S., Dumas, M., and ter Hofstede, A. H. M., "WofBPEL: A tool for automated analysis of BPEL processes," in Benatallah *et al.* [75], pp. 484–489.

[246] Özsu, M. T. and Valduriez, P., *Principles of distributed database systems (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

[247] Padala, P., Hou, K.-Y., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., and Merchant, A., "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, (New York, NY, USA), pp. 13–26, ACM, 2009.

[248] Padala, P., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., and Salem, K., "Adaptive control of virtualized resources in utility computing environments," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, (New York, NY, USA), pp. 289–302, ACM, 2007.

[249] Palankar, M. R., Iamnitchi, A., Ripeanu, M., and Garfinkel, S., "Amazon s3 for science grids: a viable solution?," in *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, (New York, NY, USA), pp. 55–64, ACM, 2008.

[250] Parekh, J., Jung, G., Swint, G., Pu, C., and Sahai, A., "Issues in bottleneck detection in multi-tier enterprise applications," *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pp. 302–303, June 2006.

[251] Patino-Martinez, M., Jimenez-Peris, R., Kemme, B., and Alonso, G., "Middle-r: Consistent database replication at the middleware level," *ACM Trans. Comput. Syst.*, vol. 23, no. 4, pp. 375–423, 2005.

[252] Petri, C. A., *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, Darmstadt, Germany, 1962.

[253] Pettie, S. and Sanders, P., "A simpler linear time $2/3 - \epsilon$ approximation for maximum weight matching," Tech. Rep. MPI-I-2004-1-002, MPII, 2004.

[254] Plattner, C. and Alonso, G., "Ganymed: scalable replication for transactional web applications," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, (New York, NY, USA), pp. 155–174, Springer-Verlag New York, Inc., 2004.

[255] Porter, G. and Katz, R. H., "Effective web service load balancing through statistical monitoring," *Commun. ACM*, vol. 49, no. 3, pp. 48–54, 2006.

[256] Powers, R., Goldszmidt, M., and Cohen, I., "Short term performance forecasting in enterprise systems," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, (New York, NY, USA), pp. 801–807, ACM, 2005.

[257] Preparata, F. P., Metze, G., and Chien, R. T., "On the connection assignment problem of diagnosable systems," *Electronic Computers, IEEE Transactions on*, vol. EC-16, no. 6, pp. 848–854, 1967.

[258] Preparata, F. P., Metze, G., and Chien, R. T., "On the connection assignment problem of diagnosable systems," *Electronic Computers, IEEE Transactions on*, vol. EC-16, no. 6, pp. 848–854, 1967.

[259] Prodan, R. and Fahringer, T., "Zen: A directive-based language for automatic experiment management of distributed and parallel programs," 2002.

[260] Prodan, R. and Fahringer, T., "Zenturio: An experiment management system for cluster and grid computing," in *In Proceedings of the 4th International Conference on Cluster Computing (CLUSTER 2002*, pp. 9–18, IEEE Computer Society Press. http://www.par.univie.ac.at/project/zenturio, 2002.

[261] Pu, C., Sahai, A., Parekh, J., Jung, G., Bae, J., Cha, Y.-K., Garcia, T., Irani, D., Lee, J., and Lin, Q., "An observation-based approach to performance characterization of distributed n-tier applications," in *IISWC '07*, September 2007.

[262] Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., and Pu, C., "Understanding performance interference of I/O workload in virtualized cloud environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 51–58, 2010.

[263] Puhlmann, F., "Why do we actually need the π-calculus for business process management?," in Abramowicz and Mayr [47], pp. 77–89.

[264] Puhlmann, F. and Weske, M., "Using the π-calculus for formalizing workflow patterns," *Business Process Management*, pp. 153–168, 2005.

[265] Reijers, H. A., "Workflow flexibility: The forlorn promise," in *WETICE* [37], pp. 271–272.

[266] Ricciato, F., Vacirca, F., and Svoboda, P., "Diagnosis of capacity bottlenecks via passive monitoring in 3g networks: An empirical analysis," *Comput. Netw.*, vol. 51, no. 4, pp. 1205–1231, 2007.

[267] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S., "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, (New York, NY, USA), pp. 199–212, ACM, 2009.

[268] Rodrigues, M. C., Malkowski, S., and Ferreira, J. E., "Implementing rigorous web services with process algebra: navigation plan for web services," in *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, (New York, NY, USA), pp. 625–631, ACM, 2009.

[269] Roman, D., Lausen, H., and Keller, U., "Web service modeling ontology." WSMO Final Draft, October 2006.

[270] Romijn, J., Smith, G., and van de Pol, J., eds., *Integrated Formal Methods, 5th International Conference, IFM 2005, Eindhoven, The Netherlands, November 29 - December 2, 2005, Proceedings*, vol. 3771 of *Lecture Notes in Computer Science*, Springer, 2005.

[271] Roscoe, A. W., Hoare, C. A. R., and Bird, R., *The Theory and Practice of Concurrency*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997.

[272] Roser, C., Nakano, M., and Tanaka, M., "Shifting bottleneck detection," *Simulation Conference, 2002. Proceedings of the Winter*, vol. 2, pp. 1079–1086 vol.2, Dec. 2002.

[273] Roser, C., Nakano, M., and Tanaka, M., "Simulation test bed for manufacturing analysis: comparison of bottleneck detection methods for agv systems," in *WSC '03: Proceedings of the 35th conference on Winter simulation*, pp. 1192–1198, Winter Simulation Conference, 2003.

[274] Russell, N. C., *Foundations of Process-Aware Information Systems*. PhD thesis, Queensland University of Technology, Brisbane, Australia, December 2007.

[275] Sahai, A., Pu, C., Gueyoung Jung, Q. W., and Swint, G., "Towards automated deployment of built-to-order systems," in *IFIP/IEEE Distributed Systems: Operations and Management (DSOM '05)*, pp. 109–120, 2005.

[276] Sahai, A., Singhal, S., Machiraju, V., and Joshi, R., "Automated generation of resource configurations through policies," in *in Proceedings of the IEEE 5 th International Workshop on Policies for Distributed Systems and Networks*, pp. 7–9, 2004.

[277] Sahai, A., Singhal, S., and Udupi, Y. B., "A classification-based approach to policy refinement," in *In Proc. of The Tenth IFIP/IEEE IM*, 2007.

[278] Sailer, A., Head, M. R., Kochut, A., and Shaikh, H., "Graph-based cloud service placement," in *Proceedings of the 2010 IEEE International Conference on Services Computing*, SCC '10, (Washington, DC, USA), pp. 89–96, IEEE Computer Society, 2010.

[279] Salaün, G., Bordeaux, L., and Schaerf, M., "Describing and reasoning on web services using process algebra," in *ICWS* [33], pp. 43–51.

[280] Salaün, G., Ferrara, A., and Chirichiello, A., "Negotiation among web services using lotos/cadp," in Zhang [352], pp. 198–212.

[281] Salaün, G., Kramer, J., Lang, F., and Magee, J., "Translating fsp into lotos and networks of automata," in Davies and Gibbons [122], pp. 558–578.

[282] Salaün, G. and Serwe, W., "Translating hardware process algebras into standard process algebras: Illustration with chp and lotos," in Romijn *et al.* [270], pp. 287–306.

[283] Saran, H. and Vazirani, V. V., "Finding k-cuts within twice the optimal," in *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pp. 743–751, 1991.

[284] Schulz, G., *The Green and Virtual Data Center*. Boston, MA, USA: Auerbach Publications, 2009.

[285] Shahabuddin, J., Chrungoo, A., Gupta, V., Juneja, S., Kapoor, S., and Kumar, A., "Stream-packing: Resource allocation in web server farms with a qos guarantee," in *Proceedings of the 8th International Conference on High Performance Computing*, HiPC '01, (London, UK, UK), pp. 182–191, Springer-Verlag, 2001.

[286] Shan, M.-C., Dayal, U., and Hsu, M., eds., *Technologies for E-Services, 5th International Workshop, TES 2004, Toronto, Canada, August 29-30, 2004, Revised Selected Papers*, vol. 3324 of *Lecture Notes in Computer Science*, Springer, 2005.

[287] Shasha, D. E., *Database tuning: a principled approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992.

[288] Singh, A., Korupolu, M., and Voruganti, K., "Zodiac: efficient impact analysis for storage area networks," in *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, (Berkeley, CA, USA), pp. 6–6, USENIX Association, 2005.

[289] Singh, A., Hayward, B., and Anderson, D., "Green IT takes center stage." Springboard Research, 2007.

[290] Sitaraman, S. and Venkatesan, S., "Forensic analysis of file system intrusions using improved backtracking," in *IWIA '05: Proceedings of the Third IEEE International Workshop on Information Assurance*, (Washington, DC, USA), pp. 154–163, IEEE Computer Society, 2005.

[291] Skowron, A., Agrawal, R., Luck, M., Yamaguchi, T., Morizet-Mahoudeaux, P., Liu, J., and Zhong, N., eds., *2005 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2005), 19-22 September 2005, Compiegne, France*, IEEE Computer Society, 2005.

[292] Snyder, B., "Server virtualization has stalled, despite the hype," *InfoWorld*, December 2010.

[293] Sobel, W., Subramanyam, S., Sucharitakul, A., Nguyen, J., Wong, H., Klepchukov, A., Patil, S., Fox, O., and Patterson, D., "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," 2008.

[294] Sobel, W., Subramanyam, S., Sucharitakul, A., Nguyen, J., Wong, H., Klepchukov, A., Patil, S., Fox, O., and Patterson, D., "CloudStone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," 2008.

[295] Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., and Kamath, S., "Automatic virtual machine configuration for database workloads," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 953–966, ACM, 2008.

[296] Srinath Perera, Chathura Herath, J. E. E. C. A. R. D. J. S. W. and Daniels, G., "Axis2, middleware for next generation web services," in *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, (Washington, DC, USA), pp. 833–840, IEEE Computer Society, 2006.

[297] Stewart, C., Kelly, T., and Zhang, A., "Exploiting nonstationarity for performance prediction," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 31–44, 2007.

[298] Stewart, C. and Shen, K., "Performance modeling and system management for multi-component online services," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, (Berkeley, CA, USA), pp. 71–84, USENIX Association, 2005.

[299] Swint, G. S., Pu, C., Jung, G., Yan, W., Koh, Y., Wu, Q., Consel, C., Sahai, A., and Moriyama, K., "Clearwater: extensible, flexible, modular code generation," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, (New York, NY, USA), pp. 144–153, ACM, 2005.

[300] Ta, D., Zhou, S., Cai, W., Tang, X., and Ayani, R., "Network-aware server placement for highly interactive distributed virtual environments," in *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, (Washington, DC, USA), pp. 95–102, IEEE Computer Society, 2008.

[301] Takecian, P. L., "ACP e LOTOS: um estudo comparativo baseado em conceitos de BPEL e padrões de controle de fluxo," Master's thesis, University of São Paulo, São Paulo, Brazil, June 2008.

[302] Tang, C., Steinder, M., Spreitzer, M., and Pacifici, G., "A scalable application placement controller for enterprise data centers," in *Proceedings of the 16th international conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 331–340, ACM, 2007.

[303] ter Beek, M. H., Bucchiarone, A., and Gnesi, S., "Web service composition approaches: From industrial standards to formal methods," in *ICIW* [41], p. 15.

[304] CISCO, "Introduction to cisco ios netflow - a technical overview," white paper, October 2007.

[305] GATech, "Research Network Operations Center." `www.rnoc.gatech.edu`.

[306] Thereska, E. and Ganger, G. R., "Ironmodel: robust performance models in the wild," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 1, pp. 253–264, 2008.

[307] Thereska, E., Salmon, B., Strunk, J., Wachs, M., Abd-El-Malek, M., Lopez, J., and Ganger, G. R., "Stardust: tracking activity in a distributed storage system," in *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '06/Performance '06, (New York, NY, USA), pp. 3–14, ACM, 2006.

[308] Turner, K. J., "Representing and analysing composed web services using CRESS," *J. Network and Computer Applications*, vol. 30, no. 2, pp. 541–562, 2007.

[309] Urgaonkar, B., Shenoy, P., Chandra, A., and Goyal, P., "Dynamic provisioning of multi-tier internet applications," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pp. 217–228, 2005.

[310] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A., "An analytical model for multi-tier internet services and its applications," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 291–302, 2005.

[311] Urgaonkar, B., Shenoy, P., and Roscoe, T., "Resource overbooking and application profiling in shared hosting platforms," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 239–254, 2002.

[312] Urgaonkar, B., Shenoy, P., and Roscoe, T., "Resource overbooking and application profiling in a shared internet hosting platform," *ACM Trans. Internet Technol.*, vol. 9, no. 1, pp. 1–45, 2009.

[313] Vahidov, R. and Neumann, D., "Situated decision support for managing service level agreement negotiations," in *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, (Washington, DC, USA), p. 52, IEEE Computer Society, 2008.

[314] van Breugel, F. and Koshkina, M., "Models and verification of BPEL." Unpublished Draft, September 2006. http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf.

[315] van der Aalst, W. M. P. and Berens, P. J. S., "Beyond workflow management: product-driven case handling," in *GROUP '01: Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work*, (New York, NY, USA), pp. 42–51, ACM, 2001.

[316] van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. J. M. M., "Workflow mining: a survey of issues and approaches," *Data Knowl. Eng.*, vol. 47, no. 2, pp. 237–267, 2003.

[317] van der Aalst, W. and ter Hofstede, A., "http://www.workflowpatterns.com," June 2008.

[318] van der Aalst, W. M. P., "Verification of workflow nets," in *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, (London, UK), pp. 407–426, Springer-Verlag, 1997.

[319] van der Aalst, W. M. P., "The application of petri nets to workflow management," *Journal of Circuits, Systems, and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[320] van der Aalst, W. M. P., "Pi calculus versus petri nets: let us eat humble pie rather than further inflate the pi hype." Unpublished Discussion Paper, 2003. http://tmitwww.tm.tue.nl/staff/wvdaalst/pi-hype.pdf.

[321] van der Aalst, W. M. P., Dumas, M., Ouyang, C., Rozinat, A., and Verbeek, E., "Conformance checking of service behavior," *ACM Trans. Internet Techn.*, vol. 8, no. 3, 2008.

[322] van der Aalst, W. M. P., Dumas, M., and ter Hofstede, A. H. M., "Web service composition languages: Old wine in new bottles?," in *EUROMICRO* [32], pp. 298–307.

[323] van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., Russell, N., Verbeek, H. M. W. E., and Wohed, P., "Life after bpel?," in Bravetti *et al.* [89], pp. 35–50.

[324] van der Aalst, W. M. P. and ter Hofstede, A. H. M., "YAWL: yet another workflow language," *Inf. Syst.*, vol. 30, no. 4, pp. 245–275, 2005.

[325] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P., "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[326] Van Renesse, R., Birman, K. P., and Vogels, W., "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, pp. 164–206, May 2003.

[327] Vassiliadis, P., "A survey of extract-transform-load technology," in *Integrations of Data Warehousing, Data Mining and Database Technologies*, pp. 171–199, 2011.

[328] Velte, T., Velte, A., and Elsenpeter, R. C., *Green IT: Reduce Your Information System's Environmental Impact While Adding to the Bottom Line*. New York, NY, USA: McGraw-Hill, Inc., 2009.

[329] Verma, A., Ahuja, P., and Neogi, A., "pmapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, (New York, NY, USA), pp. 243–264, Springer-Verlag New York, Inc., 2008.

[330] Wainwright, R. L. and Haddad, H., eds., *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, ACM, 2008.

[331] Walker, E., "Benchmarking Amazon EC2 for high-performance scientific computing," *LOGIN*, vol. 33, pp. 18–23, Oct. 2008.

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

[333] Wang, Q., Kanemasa, Y., Li, J., Jayasinghe, D., Kawaba, M., and Pu, C., "Response time reliability in cloud environments: An empirical study of n-tier applications at high resource utilization," in *SRDS*, pp. 378–383, 2012.

[334] Wang, Q., Malkowski, S., Jayasinghe, D., Xiong, P., Pu, C., Kanemasa, Y., Kawaba, M., and Harada, L., "The impact of soft resource allocation on n-tier application scalability," in *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, (Washington, DC, USA), pp. 1034–1045, IEEE Computer Society, 2011.

[335] Wang, Y., Carzaniga, A., and Wolf, A. L., "Four enhancements to automateddistributed system experimentation methods," in *Proceedings of the 30th international conference on Software engineering*, ICSE '08, (New York, NY, USA), pp. 491–500, ACM, 2008.

[336] Wang, Y., Rutherford, M. J., Carzaniga, A., and Wolf, A. L., "Automating experimentation on distributed testbeds," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, (New York, NY, USA), pp. 164–173, ACM, 2005.

[337] Wang, Y., Zhao, Q., and Zheng, D., "Bottlenecks in production networks: An overview," *Journal of Systems Science and Systems Engineering*, vol. 14, no. 3, pp. 347–363, 2005.

[338] Ward, C., Aravamudan, N., Bhattacharya, K., Cheng, K., Filepp, R., Kearney, R., Peterson, B., Shwartz, L., and Young, C., "Workload migration into clouds challenges, experiences, opportunities," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 164–171, 2010.

[339] Weikum, G., König, A. C., and Dessloch, S., eds., *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, ACM, 2004.

[340] Weske, M., *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[341] White, S., "Business process modeling notation (BPMN)." Business Process Management Initiative (BPMI)–Version 1.0–BPMI.org, 2004.

[342] Wu, C., Hamada, M., and Wu, C., *Experiments: planning, analysis, and parameter design optimization*. Wiley New York, 2000.

[343] XenSource, "A performance comparison of commercial hypervisors." Technical report, 2007.

[344] Xiong, K. and Perros, H., "Service performance and analysis in cloud computing," in *Services - I, 2009 World Conference on*, pp. 693–700, 2009.

[345] Xiong, P., *Dynamic Monitoring, Modeling and Management of Performance and Resources for Applications in Cloud*. PhD thesis, Georgia Institute of Technology, Atlanta, Georgia, USA, 2012.

[346] Xiong, P., Chi, Y., Zhu, S., Moon, H. J., Pu, C., and Hacigumus, H., "Intelligent management of virtualized resources for database systems in cloud environment," in *Proceedings of IEEE International Conference on Data Engineering (ICDE) 2011*, 2011.

[347] Xiong, P., Wang, Z., Jung, G., and Pu, C., "Study on performance management and application behavior in virtualized environment," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pp. 841–844, 2010.

[348] Xiong, P., Wang, Z., Malkowski, S., Jayasinghe, D., Wang, Q., and Pu, C., "Economical provisioning of multi-tier web applications in clouds: A unified controller for sla-based resource allocation and partitioning," in *The 31st Int'l Conference on Distributed Computing Systems (ICDCS 2011)*, 2011.

[349] Yalagandula, P. and Dahlin, M., "A scalable distributed information management system," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, (New York, NY, USA), pp. 379–390, ACM, 2004.

[350] Yan, F., Mountrouidou, X., Riska, A., and Smirni, E., "Toward automating work consolidation with performance guarantees in storage clusters," in *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '11, (Washington, DC, USA), pp. 326–335, IEEE Computer Society, 2011.

[351] Yigitbasi, N., Iosup, A., Epema, D., and Ostermann, S., "C-meter: A framework for performance analysis of computing clouds," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pp. 472–477, 2009.

[352] Zhang, L.-J., ed., *Web Services, European Conference, ECOWS 2004, Erfurt, Germany, September 27-30, 2004, Proceedings*, vol. 3250 of *Lecture Notes in Computer Science*, Springer, 2004.

[353] Zhang, Q., Cherkasova, L., and Smirni, E., "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, (Washington, DC, USA), p. 27, IEEE Computer Society, 2007.

[354] Zhang, R., Routray, R., Eyers, D., Chambliss, D., Sarkar, P., Willcocks, D., and Pietzuch, P., "IO tetris: Deep storage consolidation for the cloud via fine-grained workload analysis," in *4th International IEEE Conference on Cloud Computing (IEEE CLOUD)*, (Washington, DC, USA), IEEE, 07/2011 2011.

[355] Zhang, Y., Huang, G., Liu, X., and Mei, H., "Integrating resource consumption and allocation for infrastructure resources on-demand," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 75–82, 2010.

[356] Zhang, Z. H., Zhan, J. F., Li, Y., Wang, L., Meng, D., and Sang, B., "Precise request tracing and performance debugging for multi-tier service of black boxes," in *DSN '09: Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, (Washington, DC, USA), IEEE Computer Society, 2009.

[357] Zhao, L., Zaki, M. J., and Ramakrishnan, N., "Blosom: A framework for mining arbitrary boolean expressions over attribute sets," in *Proceedings of the 12th International Conference On Knowledge Discovery and Data Mining (KDD 2006)*, pp. 827–832, 2006.

[358] Zheng, Z., Lan, Z., Park, B.-H., and Geist, A., "System log pre-processing to improve failure prediction," in *DSN '09: Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, (Washington, DC, USA), IEEE Computer Society, 2009.

[359] Zuliane, D., Oikawa, M. K., Malkowski, S., Alcazar, J. P., and Ferreira, J. E., "The riverfish approach to business process modeling: Linking business steps to control-flow patterns," in Bertino and Joshi [76], pp. 179–193.

An Automated Approach to Create, Manage and Analyze Large- Scale Experiments for Elastic N-Tier Application in Clouds

Indika Deepal Jayasinghe

213 Pages

Directed by Dr. Calton Pu

Cloud computing has revolutionized the computing landscape by providing on-demand, pay-as-you-go access to elastically scalable resources. Many applications are now being migrated from on-premises data centers to public clouds; yet, the transition to the cloud is not always straightforward and smooth. An application that performed well in an on-premise data center may not perform identically in public computing clouds, because many variables like virtualization can impact the application's performance. By collecting significant performance data through experimental study, the cloud's complexity particularly as it relates to performance can be revealed. However, conducting large-scale system experiments is particularly challenging because of the practical difficulties that arise during experimental deployment, configuration, execution and data processing. In spite of these associated complexities, we argue that a promising approach for addressing these challenges is to leverage automation to facilitate the exhaustive measurement of large-scale experiments.

Automation provides numerous benefits: removes the error prone and cumbersome involvement of human testers, reduces the burden of configuring and running large-scale experiments for distributed applications, and accelerates the process of reliable applications testing. In our approach, we have automated three key activities associated with the experiment measurement process: *create*, *manage* and *analyze*. In create, we prepare the platform and deploy and configure applications. In manage, we initialize the application components (in a reproducible and verifiable order), execute workloads, collect resource monitoring and other performance data, and parse and upload the results to the data warehouse. In analyze, we process the collected data using various statistical and visualization techniques

to understand and explain performance phenomena. In our approach, a user provides the experiment configuration file, so at the end, the user merely receives the results while the framework does everything else. We enable the automation through code generation. From an architectural viewpoint, our code generator adopts the compiler approach of multiple, serial transformative stages; the hallmarks of this approach are that stages typically operate on an XML document that is the intermediate representation, and XSLT performs the code generation. Our automated approach to large-scale experiments has enabled cloud experiments to scale well beyond the limits of manual experimentation, and it has enabled us to identify non-trivial performance phenomena that would not have been possible otherwise.