

**AN EMPIRICAL APPROACH TO AUTOMATED PERFORMANCE
MANAGEMENT FOR ELASTIC N-TIER APPLICATIONS IN
COMPUTING CLOUDS**

A Dissertation
Presented to
The Academic Faculty

by

Simon J. Malkowski

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

Georgia Institute of Technology
May 2012

**AN EMPIRICAL APPROACH TO AUTOMATED PERFORMANCE
MANAGEMENT FOR ELASTIC N-TIER APPLICATIONS IN
COMPUTING CLOUDS**

Approved by:

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

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

Dr. Leo Mark
School of Computer Science
Georgia Institute of Technology

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

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

Date Approved: March 16th, 2012

To My Parents

ACKNOWLEDGEMENTS

It is with great pleasure that I give credit where credit is due and sincerely thank the numerous people who made significant contributions to this dissertation. First and foremost, Dr. Calton Pu tirelessly guided me throughout my entire graduate school career as patient teacher and understanding advisor. Hanwei Chen, Dr. Markus Hedwig, Deepal Jayasinghe, Yasuhiko Kanemasa, Motoyuki Kawaba, Jack Li, Dr. Dirk Neumann, Jason Parekh, Junhee Park, Dr. Akhil Sahai, Qingyang Wang, and Masao Yamamoto directly contributed to this dissertation by co-authoring my research papers. Last but not least, Dr. Ling Liu, Dr. Leo Mark, Dr. Shamkant B. Navathe, and Dr. João E. Ferreira served on my dissertation committee and helped me to shape this work in its entirety.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF SYMBOLS OR ABBREVIATIONS	xiv
GLOSSARY	xvi
SUMMARYxviii
I INTRODUCTION	1
II RELATED WORK	8
2.1 Experimental Analysis of Elastic System Scalability and Consolidation	8
2.2 Modeling and Detection of Non-trivial Performance Phenomena in Elastic Systems	10
2.3 Automated Control and Configuration Planning of Elastic Systems	11
III A STUDY OF PARALLEL DATABASE SERVER SCALABILITY	13
3.1 Introduction	14
3.2 Experimental Setup	16
3.2.1 Benchmark Applications	17
3.2.2 Database Replication Techniques	19
3.2.3 Hardware Setup	19
3.2.4 Dataset Magnitude and Data Visualization	20
3.3 Evaluation of Hardware Choices	24
3.4 Evaluation of DBMS Choices	32
3.5 Evaluation of Middleware Choices	39
3.6 Evaluating Implementation-specific Bottlenecks	41
3.6.1 Long Read-queries in Low-end MySQL Scenario	42
3.6.2 Self-join Limitation in PostgreSQL Scenario	46

3.7	Related Work	50
3.8	Future Work	52
3.9	Conclusion	53
3.10	Appendix	54
3.10.1	Code Generation	54
3.10.2	Network Topology and Deployment	55
3.10.3	Experimental Cycle	56
3.10.4	Data Point Collection	56
3.10.5	Single-bottlenecks vs. Multi-bottlenecks	57
IV	CHALLENGES AND OPPORTUNITIES IN CONSOLIDATION	59
4.1	Introduction	60
4.2	Experimental Setup	62
4.3	Experimental Consolidation Study	65
4.3.1	Non-monotonic Response Time in Consolidation	67
4.3.2	Hypotheses and Correlation Analysis	73
4.3.3	Software Resource Allocation in Consolidation	77
4.4	Related Work	79
4.5	Conclusion	82
V	STATISTICAL BOTTLENECK DETECTION	83
5.1	Introduction	83
5.2	Intervention Analysis	84
5.2.1	Assumptions	85
5.2.2	The Detection Model	86
5.2.3	Determining an Intervention Point	88
5.2.4	Metric Selection Scheme	89
5.2.5	Impact Assessment	90
5.3	Experimental Evaluation	91
5.3.1	Bottleneck Detection in the 1/1/1 Configuration	93

5.3.2	Performance Comparison of the Analysis	97
5.4	Related Work	98
5.5	Conclusion	99
VI	OBSERVATION AND ANALYSIS OF MULTI-BOTTLENECKS	100
6.1	Introduction	100
6.2	Simple Classification of Multi-bottlenecks	103
6.3	Statistical Data Interpretation	110
6.4	Experimental Setup	111
6.5	Concurrent Bottlenecks	114
6.6	Oscillatory Bottlenecks	117
6.6.1	Within-node Dependences	118
6.6.2	Between-Node Dependences	120
6.7	Related Work	124
6.8	Conclusion	125
VII	A TOOL FOR CLOUD CONFIGURATION PLANNING	127
7.1	Introduction	127
7.2	Background	130
7.2.1	Service Level Agreements	130
7.2.2	Experimental Infrastructure	132
7.2.3	Single-bottlenecks vs. Multi-bottlenecks	132
7.3	Empirical Configuration Planning	134
7.4	Tool Implementation	137
7.5	Case Study	139
7.5.1	Setup	139
7.5.2	Results	141
7.6	Related Work	147
7.7	Conclusion	149

VIII	AUTOMATED CONTROL FOR ELASTIC N-TIER WORKLOADS	150
8.1	Introduction	150
8.2	Adaptation Architecture	154
8.2.1	Meta-model	156
8.2.2	Horizontal Scale Model	156
8.2.3	Empirical Model	157
8.2.4	Workload Forecast Model	160
8.2.5	Adaptation Workflow	161
8.3	Experimental Evaluation	161
8.3.1	Elastic Application Testbed	161
8.3.2	IAE Implementation Details	164
8.3.3	Experimental Results	165
8.3.4	Synthetic Workload Results	165
8.3.5	Empirical Scaling Results	171
8.3.6	Real Workload Results	173
8.4	Related Work	174
8.5	Conclusion	176
IX	CONCLUSION	177
	REFERENCES	182

LIST OF TABLES

1	Details of the benchmark application setup and the corresponding software configuration choices.	18
2	Details of the hardware setup in the Emulab cluster and sample system topology.	21
3	Data set size and complexity for RUBBoS experiments.	22
4	Distribution of SQL response times in the first database server of the 1/2/1/9ML C-JDBC scenario for 8,000 users scenario with browse-only workload.	46
5	Summary of experimental setup (i.e., hardware, operating system, software, and virtualization environment).	64
6	The workload of Sys_A is increased monotonically at a constant rate of 100 concurrent users while the workload of Sys_B is kept constant at 2,300 concurrent users during all experiments. The deployment corresponds to Figure 17(b).	66
7	Heuristic approximation of the intervention point.	95
8	Results of the bottleneck detection process.	95
9	Ranking function value against interval width.	96
10	Ranking function value against different crossover points.	96
11	Ranking function value against the step width.	97
12	Results of the bottleneck detection process.	98
13	Details of the experimental setup on the Emulab cluster.	113
14	Software setup.	140
15	Hardware setup.	140
16	SLA Model	144
17	Profit optimal configurations.	146

LIST OF FIGURES

1	A web-facing elastic three-tier application.	3
2	Major quality of service determinants.	4
3	Overview of core dissertation research in the area of automated performance management for elastic n-tier applications in computing clouds.	5
4	Maximum system throughput and bottleneck map for RUBBoS read/write workload with C-JDBC and MySQL on normal nodes.	26
5	Maximum system throughput and bottleneck map for RUBBoS read/write workload with wait-all C-JDBC on normal nodes and MySQL on low-cost nodes.	28
6	Maximum system throughput and bottleneck map for RUBBoS read/write workload with wait-first C-JDBC on normal nodes and MySQL on low-cost nodes.	29
7	Resource utilization densities for 1/2/1/1ML RUBBoS with read/write workload, “wait-all” C-JDBC, and MySQL.	30
8	Resource utilization densities for 1/2/1/9ML RUBBoS with read/write workload, C-JDBC, and MySQL.	31
9	Maximum system throughput and bottleneck map for RUBBoS browse-only workload with C-JDBC and MySQL on normal nodes.	34
10	Maximum system throughput and bottleneck map for RUBBoS browse-only workload with C-JDBC and PostgreSQL on normal nodes.	35
11	Maximum system throughput and bottleneck map for RUBBoS browse-only workload with C-JDBC on normal nodes and MySQL on low-cost nodes.	37
12	Maximum system throughput and bottleneck map for RUBBoS read/write workload with MySQL Cluster on normal nodes.	40
13	Detailed analysis of the 1/2/1/9ML C-JDBC scenario for 8,000 users scenario with browse-only workload.	44
14	Detailed analysis of the utilization densities in the 1/2/1/9ML C-JDBC scenario with browse-only workload.	45
15	Time series analysis of 1/2/1/5P scenario for 9,000 users with a browse-only workload.	47
16	Detailed analysis of 1/2/1/5P scenario for 9,000 users with a browse-only workload.	48

17	Example of a dedicated (a) and a consolidated (b) 3-tier application system deployment, presented as mappings of software servers to physical hardware nodes.	63
18	Throughput (a) and response time (b) of 3-tier systems Sys_A and Sys_B. The deployment of the two consolidated systems corresponds to Figure 17(b).	68
19	Trace snapshots (10 seconds) of the CPU utilizations in DB node of the backend consolidation scenario with constant Sys_B workload of 2,300 users and Sys_A workloads of 1,600 users (a), 2,000 users (b), and 2,300 users (c).	70
20	The number of concurrent requests (processing and waiting) in Sys_B tiers (same experiment runs as in Figure 19).	71
21	Correlation analysis based hypothesis testing of Sys_A properties. Pearson's correlation coefficients are calculated based on CPU utilization, number of concurrent jobs, and response time traces collected in the DB node at 0.1s granularity.	75
22	Correlation analysis based hypothesis testing of Sys_B properties. Pearson's correlation coefficients are calculated based on CPU utilization, number of concurrent jobs, and response time traces collected in the DB node at 0.1s granularity.	76
23	The number of concurrent ViewStory requests in Sys_B tiers (same experiment runs as in Figures 19(b) and 20(b)).	78
24	Comparison of three-tier system performance for various software resource allocations in both the consolidated (a), (b) and the dedicated (c) deployment scenarios show in Figure 17.	80
25	SLO-satisfaction and usage metric graphs for 1/1/1 RUBiS experiment configuration.	94
26	1/2/1/1L RUBBoS experiment with read/write workload.	102
27	Simple multi-bottleneck classification.	106
28	Average throughput and response time of a 1/6/1/1L RUBiS experiment with browse-only workload an throttled database network bandwidth (2 Mbps).	115
29	Detailed bottleneck analysis of a 1/6/1/1L RUBiS experiment with browse-only workload an throttled database network bandwidth (2 Mbps).	116
30	Probability distribution (i.e., frequency of resource saturation states) among the fourteen different bottleneck resource saturation states. in 1/6/1/1L RUBiS experiment with browse-only workload an throttled database network bandwidth (2 Mbps).	117

31	Bottleneck analysis of a 1/2/1/1L RUBBoS experiment with read/write workload.	119
32	Probability distribution (i.e., frequency of resource saturation states) among the four different bottleneck resource saturation states in 1/2/1/1L RUBBoS experiment with read/write workload.	120
33	Average throughput and response time of a 1/1/1/8L RUBBoS experiment with read/write workload.	121
34	Density analysis of 1/1/1/8L RUBBoS experiment with read/write workload.	122
35	Detailed bottleneck analysis of a 1/1/1/8L RUBBoS experiment with read/write workload.	123
36	Profit model schema.	131
37	Simple multi-bottleneck classification.	134
38	Schematic data refinement process in the CloudXplor Tool.	135
39	Schema of CloudXplor Tool.	137
40	Screenshot of current implantation of the CloudXplor Tool.	138
41	Throughput of a RUBBoS system with 1 web, 2 application, and 4 data nodes servers under read/write workload.	142
42	Maximal throughput of a RUBBoS system with 1 web and 2 application servers under read/write workload.	142
43	Response time distributions of a RUBBoS system with 1 web, 1 application, 1 MySQL, and 2 data node servers under read/write workload.	143
44	Revenue and cost analysis of a RUBBoS system under read/write workload.	145
45	Profit and cost analysis of a RUBBoS system under read/write workload. . .	146
46	Elastic four-tier application.	151
47	Major quality of service determinants.	152
48	Integrative Adaptation Engine architecture.	155
49	Extensible meta-model that implements the EM, the WFM, and the HSM with ODS access.	157
50	Decision-making workflow inside the multi-model controller.	162
51	ODS initialization based on a synthetic workload with a uniform interaction distribution.	167
52	ODS initialization based on a synthetic workload with a uniform interaction distribution.	168

53	ODS initialization based on a synthetic workload with increased application server load.	169
54	ODS initialization based on a synthetic workload with increased application server load.	170
55	EM-based scaling with uniform interaction distribution (ODS initialized according to Figures 51 and 52).	171
56	EM-based scaling with increased app. server load (ODS initialized according to Figures 53 and 54).	172
57	EM-based scaling with uniform workload and changing ratio of Business Logic Requests.	173
58	Assimilated <code>Wikipedia.com</code> workload.	173
59	IAE application scaling based on the workload in Figure 58 and balanced interactions.	174

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, p. 62.
C-JDBC	Clustered Java Database Connectivity, an open source database cluster middleware that allows Java applications to transparently access a cluster of databases, p. 13.
CloudXplor	A research prototype of a novel tool for configuration planning in clouds based on an iterative and interactive refinement process of empirical data, p. 7.
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, p. 3.
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, p. 55.
Emulab	A network testbed that provides a wide range of computing environments for development and evaluation of computer systems, p. 3.
Linux	A Unix-like computer operating system, distributed and developed as open source software, p. 17.
MySQL	An open source relational database management system that runs as a server providing multi-user access to a number of databases, p. 13.
MySQL Cluster	An open source real-time transactional relational database with high availability, "shared-nothing" distributed architecture with no single point of failure, p. 13.
PostgreSQL	An open source object-relational database management system targeting enterprise class database environments, p. 13.
RUBBoS	Rice University Bulletin Board System, an open source n-tier application benchmark modeled after the website http://slashdot.org/ , p. 7.

- RUBiS** Rice University Bidding System, an open source n-tier application benchmark modeled after the website <http://www.ebay.com/>, p. 9.
- SysViz** A trace analysis tool, developed at the Fujitsu Laboratories, able to reconstruct entire traces of executed transactions in n-tier application systems, p. 61.
- Tomcat** Apache Tomcat is an open source application server based on an implementation of Java Servlet and JavaServer Pages technologies, p. 62.

SUMMARY

Achieving a high degree of efficiency is non-trivial when managing the performance of large web-facing applications such as e-commerce websites and social networks. While computing clouds have been touted as a good solution for elastic applications, many significant technological challenges still have to be addressed in order to leverage the full potential of this new computing paradigm. In this dissertation I argue that the automation of elastic n-tier application performance management in computing clouds presents novel challenges to classical system performance management methodology that can be successfully addressed through a systematic empirical approach. I present strong evidence in support of my thesis in a framework of three incremental building blocks: Experimental Analysis of Elastic System Scalability and Consolidation, Modeling and Detection of Non-trivial Performance Phenomena in Elastic Systems, and Automated Control and Configuration Planning of Elastic Systems. More concretely, I first provide a proof of concept for the feasibility of large-scale experimental database system performance analyses, and illustrate several complex performance phenomena based on the gathered scalability and consolidation data. Second, I extend these initial results to a proof of concept for automating bottleneck detection based on statistical analysis and an abstract definition of multi-bottlenecks. Third, I build a performance control system that manages elastic n-tier applications efficiently with respect to complex performance phenomena such as multi-bottlenecks. This control system provides a proof of concept for automated online performance management based on empirical data.

CHAPTER I

INTRODUCTION

Achieving a high degree of efficiency is non-trivial when managing the performance of large web-facing applications such as e-commerce websites and social networks. For example, their workloads are characterized by sudden and periodic variations that are particularly difficult to forecast accurately [36]. In fact, these non-stationary workloads may have peak loads that exceed the sustained load on the order of several magnitudes [2, 16]. If such *elastic* applications are deployed in a dedicated datacenter, the service provider will face a challenging trade-off. On the one hand, satisfying Quality of Service (QoS) objectives such as short response time during peak loads will keep a large number of nodes at low utilization levels most of the time. On the other hand, provisioning mainly for the sustained load by keeping a relatively high utilization of a small number of nodes will lead to saturation and loss of business during the occasional (but often critical) peak loads.

Computing clouds have been touted as a good solution for elastic applications because asynchronous peak loads allow sharing of the same nodes using techniques such as consolidation; however, while the enormous potential of this new paradigm has been widely accepted, many significant technological challenges remain to be addressed in order to make cloud computing a “permanent game changer”. For instance, while achieving an economical sharing of cloud resources at sustained loads *and* satisfying QoS objectives at peak loads is clearly of great interest for both cloud providers and cloud users, the necessary management automation is still an open and actively studied (commercial and academic) research problem [59, 75, 76, 97, 106]. Moreover, important research on issues such as dynamic application scalability, performance benchmarking, and distributed Database Management Systems (DBMSs) that was generated before the dawn of this transformational

technology [17] has to be revisited and adapted to cloud environments [27, 55, 56].

Preliminary research results on application performance management in clouds show that one of the main difficulties—not specific to any of the aforementioned sub-problems—is scenario heterogeneity [20]. Different providers offer differently tailored services with different technological solutions, different capabilities, different price models, and different guarantees. Therefore, observable performance phenomena are highly variable, depending on both the platform (e.g., hardware and software resource utilization levels) and the applications involved (e.g., specific combination of co-located applications) [54, 82]. In mainframe-based datacenters, a classic approach would have been the tuning of hardware and software resources to maximize the throughput of stable applications and workloads [89]. However, in cloud environments, the underlying hardware is typically chosen for its good cost / performance ratio. Consequently, it is not a viable option to tune or design a balanced hardware configuration for any particular set of applications or workloads.

In this dissertation, I show that a promising approach to addressing the multi-level complexity of the problem at hand is the use of large-scale empirical data (i.e., acquired by means of observation or experimentation) in a bottom-up approach to system performance analysis and system performance management. More concretely, in contrast to classical top-down performance modeling methodologies (e.g., analytical modeling) in which experimental data are primarily used for initial model instantiation and as a means of approach validation, the prominence of empirical data in my approach differs inherently. Consequently, my thesis statement is formulated as follows.

Thesis Statement: *The automation of elastic n-tier application performance management in computing clouds, which is mandated by their scale and complexity, presents novel challenges to classical system performance management methodology that can be successfully addressed through a systematic empirical approach to system performance management.*

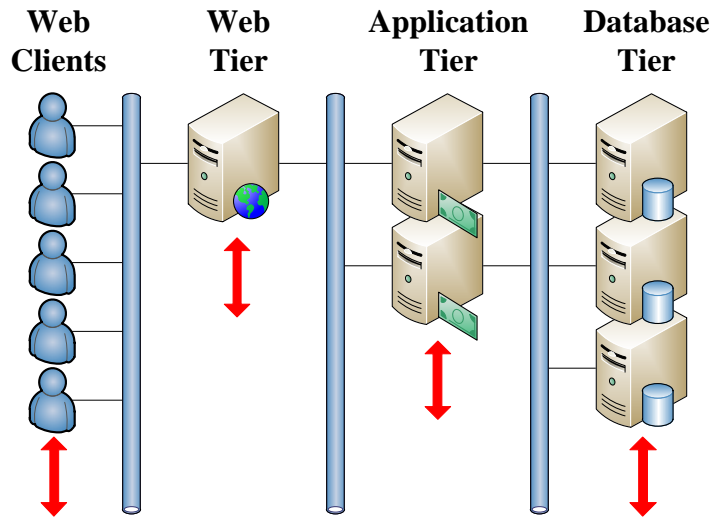


Figure 1: A web-facing elastic three-tier application.

The scope of my dissertation are web-facing elastic n-tier applications such as shown in Figure 1. These web-facing n-tier applications are popular due to their low-maintenance modular architecture but also notoriously difficult to manage efficiently under high system loads. More precisely, this research targets elastic n-tier applications that are deployed in computing clouds such as EC2 and Emulab. Figure 1 illustrates a common three-tier application example with web servers, application servers, and database servers. The main advantage of such elastic n-tier applications is their potential for on-demand performance scalability through the addition of computing resources (e.g., component servers deployed on additional hardware nodes) to overloaded tiers. However, due to the dependencies among the component servers, it is non-trivial to scale and consolidate n-tier applications in a computing clouds while maintaining constantly high QoS for all possible workload scenarios [59, 66–68, 81, 97].

There are numerous factors that may significantly affect the QoS of a system and need to be taken into account when scaling and consolidating a complex n-tier system. Figure 2 depicts some of these factors, which we call QoS *determinants*. Important QoS determinants include (but are not limited to) hardware and software components (e.g., type of storage), system state (e.g., data size and distribution), and workload (e.g., workload

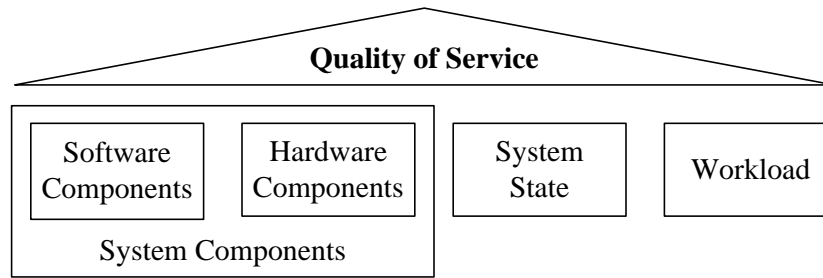


Figure 2: Major quality of service determinants.

level and transaction mix). In summary, continuous performance modeling of elastic n-tier applications is a complex non-linear analysis problem with multiple levels because of dependencies among component servers, potentially uncontrollable QoS determinants, and dynamic environment changes.

My research in automated performance management of elastic n-tier applications in computing clouds can be arranged in a framework of three incremental building blocks to provide strong support for my thesis statement. This framework and the corresponding technical chapters are illustrated in Figure 3. Moreover, the figure provides an overview of my core research publications in this area and their mapping within this dissertation. The foundational building block addresses experimental analysis of elastic system scalability and consolidation (Chapters 3 and 4). The middle building block investigates modeling and detection of non-trivial performance phenomena in elastic systems (Chapters 5 and 6). Finally, the top building block evaluates concrete approaches to automated control and configuration planning of elastic systems (Chapters 7 and 8).

At its highest abstraction level, Figure 3 illustrates the incremental nature of my dissertation research—despite the distinct differences among the building blocks themselves. In other words, in this dissertation I argue that successful identification and abstraction of non-trivial phenomena is facilitated by a thorough experimental system performance analysis. Analogously, the automation of system performance management can be effectively

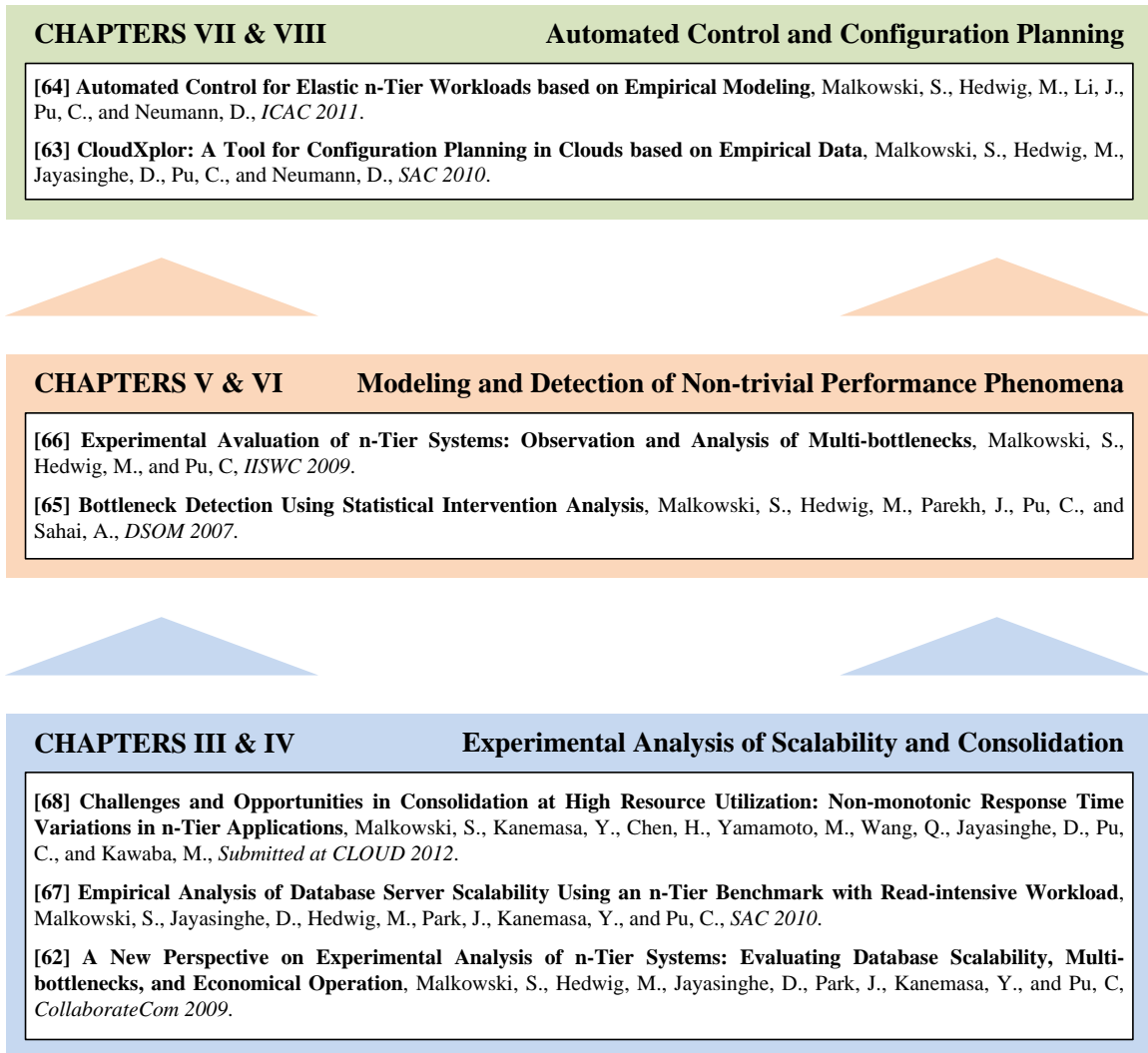


Figure 3: Overview of core dissertation research in the area of automated performance management for elastic n-tier applications in computing clouds.

addressed based on a representative set of abstractly formulated system characteristics that were derived empirically.

The experimental analysis of elastic system scalability and consolidation at large-scale enables the exploration and the understanding of complex system state spaces without rigid a priori assumptions. My research in this area focuses on experimental analysis of n-tier systems [62] with particular attention to database server scalability [67] and challenges and opportunities in consolidation at high resource utilization [68]. Furthermore, during the

course of my research, I have also investigated topics such as the comparison of similarities and differences between software and hardware resource allocation [102] and the analysis of the performance and scalability of n-tier systems when they are migrated among different clouds [47]. In this dissertation, I provide a study of scale-up and scale-out scenarios with up to nine replicated and / or partitioned database servers on dedicated hardware nodes [62,67] as a proof of concept for large-scale experimental system performance studies. While the analysis of multi-bottleneck phenomena in n-tier applications such as oscillating resource saturation [62,67] illustrates the opportunities in discovering complex performance phenomena in empirical data generated through scalability experiments, the analysis of non-monotonic response time variations in co-located n-tier application systems [68] illustrates the opportunities in discovering complex performance phenomena in empirical data generated through consolidation experiments.

The modeling and detection of non-trivial performance phenomena in elastic systems allows non-trivial performance characteristics that have been discovered in empirical data to be identified, abstracted, and formalized in order to ensure generic applicability of the research results. My research in this area includes bottleneck detection through statistical analysis [65] and the abstraction of multi-bottleneck phenomena [66], which may be particularly difficult to detect due to multi-modal resource utilization profiles. My bottleneck detection approach for n-tier applications using statistical intervention modeling [65] provides a proof of concept for formulating successful performance phenomena detection based on large-scale experimental data, and the presented classification of multi-bottlenecks [66] illustrates the opportunities in abstract classification of complex performance phenomena based on experimental evaluation of n-tier application scalability.

The automated control and configuration planning of elastic systems aims to design and build control components and tools for (fully) automated systems that operate efficiently and deliver high performance under constant QoS. My research in this area addresses configuration planning in clouds [63] and automated control based on an empirical model [64].

Moreover, during the course of my research, I have also explored topics such as automated adaptive provisioning [43, 44], datacenter investment support [42], and multi-level control based on control theory [105]. In this dissertation I present a tool for configuration planning in clouds through iterative refinement of empirical performance data—CloudXplor [63]. This tool provides a proof of concept for offline cloud configuration management based on large-scale experimental data. Furthermore, I introduce my implementation of an architecture and a multi-model controller for managing elastic n-tier applications in clouds based on an empirical model [64]. This implementation and analysis work provides a proof of concept for automated online performance management for elastic n-tier cloud applications based on empirical data.

The rest of this dissertation is organized in eight chapters, based on the framework structure that was established in Figure 3. Following an initial overview of related work in Chapter 2, my experimental analysis of elastic system scalability and consolidation is organized in Chapters 3 and 4. First, I detail an experimental study of parallel database server scalability in n-tier systems using the Rice University Bulletin Board System (RUBBoS) benchmark in Chapter 3 [62, 67] and then Chapter 4 presents a study of the challenges and opportunities in consolidation of n-tier application systems at high resource utilization in clouds [68]. My work on modeling and detection of non-trivial performance phenomena in elastic systems is organized in Chapters 5 and 6. First, I introduce my work on bottleneck detection using statistical intervention analysis in Chapter 5 [65] and then Chapter 6 presents my work on abstraction and analysis of multi-bottlenecks in n-tier applications [66]. My work on the implantation and the analysis of automated control and configuration planning components for elastic systems is organized in Chapters 7 and 8. First, I discuss configuration planning in clouds based on empirical data in Chapter 7 and then Chapter 8 proposes an empirical modeling approach to automating the control of elastic n-tier workloads [63, 64]. Finally, Chapter 9 concludes this dissertation with an overview of my main research contributions and an outlook into potential future work.

CHAPTER II

RELATED WORK

This chapter presents an initial overview of related work, organized according to the framework structure that was established in Figure 3. Section 2.1 discusses related work from the area of experimental analysis of elastic system scalability and consolidation. Subsequently, Section 2.2 introduces related approaches in modeling and detection of non-trivial performance phenomena in elastic systems. Finally, Section 2.3 presents related work in automated control and configuration planning of elastic systems. While these three sections serve as a good starting point for contextualizing the contributions made in this dissertation, all subsequent chapters that detail technical contributions (i.e., Chapters 3–8) feature comprehensive related work sections that are dedicated to each specific technical topic.

2.1 Experimental Analysis of Elastic System Scalability and Consolidation

A central problem in enterprise system studies is that scalability evaluations traditionally only address scaled load scenarios to determine best achievable performance (e.g., [29]). Concretely, experimentation methodologies with 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 solution impact in the real world [27]. Moreover, it is well understood that transactional replication can exhibit unstable scale-up behavior [39], and it is not sufficient to characterize replication performance in terms of peak throughput [27].

The evaluation of database replication techniques often falls short of real representability. There are two common approaches to evaluation with benchmarks. Performance is

either tested through microbenchmarks [78] or through web-based distributed benchmark applications such as TPC-W, RUBiS [6], and RUBBoS [15]. These benchmark applications are modeled on real applications (e.g., Slashdot.org or Ebay.com), offering real production system significance. Therefore, this approach has found a growing group of advocates [29, 30, 35, 79, 81].

Many related research papers on application consolidation in cloud environments (e.g., [38, 69]) assume linear consolidation performance and apply optimization techniques to improve the location of various tasks. For example, a recent approach applies linear programming to improve consolidated application performance through dynamic VM migration [37]. Similarly, most papers that model and solve consolidation as a bin-packing optimization problem assume linear consolidation. Clearly, the experimental study of consolidation performance in this dissertation does not invalidate these good results, but this work helps to delimit the applicability of such results that assume linear performance in consolidation.

There are several related papers that measure performance interference in consolidation and find significant variability (e.g., [53, 82]). All these papers show that a careful co-location of applications that use complementary resources is suitable to mitigate performance loss and achieve speedups equivalent to linear consolidation performance. Given the large number of system configuration parameters and resources in each component of a computer system, the current manual identification of consolidation interference sources (e.g., [53, 74]) is becoming unwieldy and expensive. This dissertation advocates methods that apply statistical analysis tools and generate automated analysis workflows. After the statistical tools have found the most likely bottleneck resources, the correlation is analyzed in detail. This is a non-trivial process since one often finds multi-modal resource utilization graphs in complex systems due to dependencies among the components [66].

Please refer to Sections 3.7 and 4.4 for a more comprehensive treatment of related work in this area.

2.2 Modeling and Detection of Non-trivial Performance Phenomena in Elastic Systems

Traditional performance analysis in computer systems presumes models based on expert knowledge, employs standard statistical methods, and parameterizes them based on a certain experimentation design [46, 58]. Queuing models have been common practice in many research efforts dealing with performance prediction [94, 98]. Although these approaches have been applied very successfully, they may suffer from their rigid assumptions when handling all evolution of large applications. The availability of extensive instrumentation data profiles [94] or constant mean inter-arrival times of request [98] do not generally hold in practice since actual parameters vary widely in real-world applications. The characteristic load non-stationarity in n-tier systems has actually been previously exploited for performance prediction by Stewart et al. [93], who also explain how their anomaly detection can be used to invoke a bottleneck detection process such as the one presented in this dissertation.

A different approach to performance analysis in computer systems are statistically induced models; for example, Cohen et al. apply a tree-augmented Naive Bayesian network to discover correlations between system-level metrics and performance states, such as SLO satisfaction and SLO failure [31]. Powers et al. also use machine learning techniques to analyze performance [80]. However, rather than detecting bottlenecks in the current system, they predict whether the system will be able to withstand the load in the following hour. Similarly, we have performed a comparative study of machine learning classifiers to investigate performance patterns in our research group [7]. Our goal was to compare the performance of several well-known machine learning algorithms as classifiers in terms of bottleneck detection, and finally to identify the classifier that best detects bottlenecks in multi-tier applications. Several other studies are based on dynamic queuing models combined with predictive and reactive provisioning [97]. Their contribution allows an enterprise system to increase capacity in bottleneck tiers during flash crowds.

In contrast to the approach in this dissertation, previous bottleneck detection research in

computer systems has built upon an extremely detailed understanding of the systems (e.g., invasively instrumented central system [21]) or an analysis confined to a small resource subset (e.g., network traffic or software configurations [83]). Methods for bottleneck detection and analysis have also been discussed in literature on simulation in areas such as industrial production [84]. This work emphasizes a technique for dealing with shifting bottleneck behavior based on resource utilization. The notion of ordering bottlenecks by severity with the help of resource demand distributions has been introduced by Luthi [61]. He provides a formal proof that approximating distributions with histograms yields a higher precision than conventional methods. Unlike the empirical work presented in this dissertation, both these approaches remain very generic without inference of domain specific phenomena.

Please refer to Sections 5.4 and 6.7 for a more comprehensive treatment of related work in this area.

2.3 Automated Control and Configuration Planning of Elastic Systems

In parallel to the technical complexity, economic considerations have recently become a key issue in IT system operation. This new awareness on sustainable operation modes can be seen in the emerging trend of Green IT [90]. Although this term is primarily ecologically motivated, it is also closely related to efficiency in the domain of datacenter operation. For IT operation, environmental goals are largely congruent with economic objectives because the major Green IT scope is to increase efficiency [88]. One option is the enhancement of isolated units such as power supplies [101]. Another more holistic approach is utilization optimization. For instance, virtualization and consolidation efforts are well-established concepts in this area [88]. However, these concepts may not always be trivially applicable. Large information systems with volatile workload processes, for example, might have too complex performance patterns, which requires a thorough an in-depth understanding of the system behavior in order to satisfy QoS [43].

To the best of my knowledge, there are only few papers on the integration of multiple automated control models in n-tier application performance management. Urgaonkar et al.'s work [97] is an example that combines reactive control and queuing-based prediction. Despite the small number of previous papers, there is little doubt that combining multiple models to overcome their individual limitations can be beneficial. More generally, although there are no automated control mechanisms for cloud management that use empirical models directly, my work on multi-model controllers can be seen as an integration (with extensions) of three component technologies: control systems, queuing models, and automated learning.

Another paper that includes a multi-model controller uses an integral control technique called proportional thresholding for automated elastic storage in clouds [59]. Their scaling of the storage service is managed through a meta-controller combining a control system-based data rebalance controller and a horizontal scale controller. Their paper makes some simplifying assumptions, e.g., storage service QoS depends only on CPU utilization as the sole control metric. They also statically set the target CPU utilization to 20%, which may be reasonable for storage service, but is relatively low for cloud node utilization. Other control system-based approaches differ from my work because they assume the availability of continuously scaling resources in non-distributed environments [75, 76].

Please refer to Sections 7.6 and 8.4 for a more comprehensive treatment of related work in this area.

CHAPTER III

AN EXPERIMENTAL STUDY OF PARALLEL DATABASE SERVER SCALABILITY IN N-TIER SYSTEMS USING THE RUBBOS BENCHMARK

One of the key requirements for efficient operation of parallel n-tier systems is accurate prediction of scalability. This is a significant challenge for classic analytical models due to non-stationary workload and complex dependencies. We have adopted an empirical approach to this challenge through a large-scale measurement study of parallel database server scalability using a five-tier RUBBoS benchmark system. Gathering data at this scale—over 525 GB of log data from over 6,000 experiments—was facilitated through automated experimentation based on code generation. This paper presents several interesting findings from the analysis of our dataset. More specifically, we analyze and contrast bottleneck migration patterns when scaling-out from one to nine replicated database servers, scenarios where low-cost hardware achieves similar performance as normal hardware, and scaling characteristics of different DBMSs (MySQL and PostgreSQL) and different distribution middlewares (C-JDBC and MySQL Cluster). The main contribution of this work is to improve the understanding of performance scalability of parallel replicated database servers. First, we provide concrete scalability results that systematically explore a large configuration space to identify and explain regions with non-trivial scalability characteristics. Second, our analysis confirms the existence of challenging bottleneck phenomena such as multi-disk oscillatory bottlenecks that are caused by complex system dependencies. For future work, these findings show the feasibility and usefulness of measurement-based analysis in studying the scalability of n-tier systems.

3.1 Introduction

Performance scalability of database servers is a constant challenge for researchers and practitioners [27]. As real world data and applications grow in size and volume, the demands on database servers also grow; e.g., in mission-critical n-tier application systems. These modern n-tier applications differ from traditional mainframe-based online transaction processing due to non-stationary workloads and high configuration complexity with explicit dependencies among the web, application, and database servers. In practice, such system dependencies are very hard to describe exhaustively a priori. Consequently, these new requirements become serious challenges for classic analytical performance models, which are typically derived based on strong assumptions (e.g., request independence, static workloads, and complete resource dependency set) [46]. An alternative approach to analytical models is empirical study through large-scale measurement of n-tier application systems. Such bottom-up approaches have become increasingly feasible and practical due to the growing availability of datacenters and cloud environments as well as automated experiment management tools based on code generation.

In this paper, we improve the understanding of performance scalability of parallel replicated database servers in n-tier systems through extensive measurement analysis using a five-tier benchmark system (RUBBoS [5]). Our measurements achieve a significant scale, rarely seen in previous work—525 GB of log data in over 6,300 experiments. These data cover a set of systematically chosen (but model-independent) configurations to support the identification of generalizable characteristics derived from observed results (e.g., response time, throughput, and their relationship to resource utilization). In contrast, analytical models in classic performance evaluation validate their predictions through experiments on a small number of representative sample configurations for each model.

The scale of our experiments is made possible by a combination of extensive hardware availability and automated experiment management tools. Our hardware environment is the Emulab testbed [3], a shared infrastructure for experimental research. The actual

experiments are enabled by a customized set of experiment management tools that automate the entire experimental process through code generation [7, 85]. Starting from configuration specification, our tools configure, deploy, and run experiments in a pipeline. The analysis of experimental results is also partially automated in this experimental benchmark cycle. In contrast, traditional ways to run distributed system experiments manually are notoriously hard to manage because scripting tasks become increasingly tedious and error-prone at larger scale.

Our experiments cover scale-up and scale-out scenarios with up to nine replicated and / or partitioned database servers. Each server in our experiments was deployed on dedicated hardware nodes (i.e., one server per node). The system deployment scenarios were altered by varying experimental parameters such as hardware components, software components, and configuration settings (e.g., replication/partitioning strategy). More concretely, we used two relational database management systems (MySQL and PostgreSQL), two database replication middlewares (C-JDBC and MySQL Cluster), two data consistency policies (“wait-all” and “wait-first”), workloads ranging from 1,000 to 13,000 concurrent users, two client interaction patterns (browse-only and read/write), and two different hardware node types (normal and low-cost).

The first main contribution of this work consists of concrete performance scalability observations that systematically cover the configuration space of up to a dozen servers. We present performance and bottleneck migration patterns when scaling-out from one to nine database servers, and analyze these scalability characteristics based on different configuration and design choices. We find scenarios where low-cost hardware achieves similar performance as normal hardware. We identify conditions under which relaxed data consistency requirements do not increase system performance. Furthermore, we confirm that the MySQL DBMS achieves significantly higher scalability through its modest MyISAM engine. However, we also find that PostgreSQL servers are able to match MySQL’s performance beyond a certain number of replicas under specific workload conditions because

the hardware bottleneck shifts to the clustering middleware tier.

The second main contribution of this work are the uncovered complex bottleneck phenomena that would escape classic analytical approaches. We identify and analyze implementation-specific bottleneck phenomena, which may, for instance, cause a small number of queries to have several orders of magnitude longer self-join execution times in the PostgreSQL DBMS. Another example are oscillatory bottlenecks between different database servers that are caused by certain data consistency settings in the C-JDBC middleware. We show that this specific dependency may be resolved at the cost of weaker data consistency, which greatly improves the scalability in this particular scenario.

The remainder of this paper is structured as follows. Section 3.2 outlines our experimental setup. In Section 3.2.4 we discuss the magnitude of the presented dataset and introduce the graphical representations that were used to visualize our findings. Sections 3.3, 3.4, 3.5, and 3.6 contain the actual evaluation of our dataset following the axes of hardware-specific, DBMS-specific, middleware-specific, and implementation-specific phenomena, respectively. Related work from the fields of system performance evaluation and distributed database server scalability analysis is summarized in Section 3.7. Section 3.8 provides an overview of potential extensions to our approach and other future work directions. A short summary of our most important findings and our conclusions are discussed in Section 3.9. Appendix 3.10 provides background on code generation, network topology and deployment, the experiment cycle, data point collection, experimental infrastructure, and bottlenecks.

3.2 Experimental Setup

In this section we outline the setup used in our experiments. The main design choices that determine our experimental setup are the benchmark application, the corresponding software choices, the software configuration, the chosen database replication technology, and the hardware setup.

3.2.1 Benchmark Applications

Among n-tier application benchmarks, RUBBoS has been used in numerous research efforts due to its real production system significance. Readers familiar with this benchmark can skip this subsection and solely consider Tables 1(a) and 1(b). The two tables outline the concrete choices of software components used in our experiments and their most important configurations, respectively.

RUBBoS [5] is an n-tier e-commerce system modeled on bulletin board news sites similar to Slashdot. The benchmark can be implemented as four-tier (client emulator, web server, application server, and database server) or five-tier (addition of clustering middleware such as C-JDBC) system. In general, this application places high load on the backend. The generated workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. The benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes. In this paper we used both the browse-only and the read/write workloads for our experiments.

Typically, the performance of benchmark application systems depends on many configurable settings (including software and hardware). To facilitate the interpretation of experimental results, we chose base configurations that are as generic as possible to guarantee reasonable performance throughout all scenarios. While this task is not trivial and requires a significant amount of domain knowledge, tuning system performance for each specific scenario is beyond the scope of this study. The most important deviations from standard hardware or software settings are spelled out when used (refer to Table 1). In the RUBBoS benchmark, each experiment trial consists of three periods: ramp-up, run, and ramp-down. In our experiments, the trials consisted of an 8-minute ramp-up, a 12-minute run period, and a 30-second ramp-down. System level performance measurements (e.g., CPU or network bandwidth utilization) were taken during the run period using Linux accounting log utilities (i.e., `iostat`). Please refer to Appendix 3.10.4 for more details on data point collection.

3.2.2 Database Replication Techniques

In this subsection we briefly introduce the two different database replication techniques that we have used in our experiments. Please refer to the cited sources for a comprehensive introduction.

C-JDBC [30] is an open source database cluster middleware, which provides Java application’s access to a cluster of databases transparently through a JDBC driver. The database can be distributed and replicated among several nodes. C-JDBC balances the queries among these nodes. C-JDBC also handles node failures and provides support for check-pointing and hot recovery. The C-JDBC server implements two update propagation strategies for replicated databases: “wait-all” (write request is completed when all replica respond) and “wait-first” (write request is considered completed upon the first replica response). The wait-all strategy is equivalent to the commonly called “write-all” replication strategy, and the wait-first strategy is similar to the primary-secondary replication strategy.

MySQL Cluster [4] is a real-time open source transactional database designed for fast, always-on access to data under high throughput conditions. MySQL Cluster utilizes a “shared nothing” architecture, which does not require any additional infrastructure and is designed to provide 99.999% data availability with no single point of failure. In our experiments we used the “in-memory” version of MySQL Cluster. Nonetheless, MySQL Cluster can also be configured to use disk-based data as well. By concept, MySQL Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. It uses the NDBCLUSTER storage engine to enable running several nodes with MySQL servers in parallel.

3.2.3 Hardware Setup

The experiments used in this paper were run in the Emulab testbed [3] with two types of servers. Table 2(a) contains a summary of the used hardware. We distinguish the two node types as “normal” and “low-cost” nodes. Note that due to the network card constraints,

normal nodes were connected over 1,000 Mbps while low-cost nodes were limited to 100 Mbps links. Despite this bandwidth limitation, none of the presented configuration scenarios resulted in a network bottleneck. All experiments were carried out by allocating each server to a dedicated physical node. In the initial setting all components were normal hardware nodes. As an alternative, database servers were also hosted on low-cost machines. Such hardware typically entails a compromise between cost advantage and performance loss, which may be hard to resolve without actual empirical data.

Throughout the paper we use a four-digit notation #W/#A/#C/#D to denote the number of servers in each tier. In the case of C-JDBC, the numbers represent web servers, application servers, clustering middleware servers, and database servers. The database management system type is either “M” or “P” for MySQL or PostgreSQL, respectively. If the server node type is low-cost, the configuration is marked with an additional “L”. The same notation is used for MySQL Cluster configurations with a slightly different meaning. If MySQL Cluster is used, the third number (i.e., “C”) denotes the number of SQL nodes and the fourth number (i.e., “D”) denotes the number of data nodes. An illustrative sample topology of a C-JDBC experiment with two client nodes, one web server, two application servers, one clustering middleware server, and three database servers on low-cost nodes (i.e., 1/2/1/3L) is shown in Table 2(b). It should be noted that we solely use one web server in all our experiments because most of the RUBBoS system load consists of dynamic web pages, which do not stress the web tier.¹ Similarly, we use solely one management node to control all MySQL Cluster experiments because the database cluster control workload is sufficiently small.



3.2.4 Dataset Magnitude and Data Visualization

We have run a large number of experiments over a wide range of configurations and workloads as the basis of this work. Table 3 quantifies the approximate magnitude of the data and the complexity of the experiments used to collect the data in our RUBBoS cycles.

¹Dynamic content request are always accompanied by static content request, the latter can, however, be considered as light-weight requests.

Table 2: Details of the hardware setup in the Emulab cluster and sample system topology.

(a) Emulab hardware node setup.

Type	Components	
Normal 	Processor	Xeon 3GHz (64-bit)
	Memory	2GB
	Network	6 x 1Gbps
	Disk	2 x 146GB (10,000rpm)
Low-cost 	Processor	PIII 600Mhz (32-bit)
	Memory	256MB
	Network	5 x 100Mbps
	Disk	13GB (7,200rpm)

(b) Sample C-JDBC topology (1/2/1/3L).

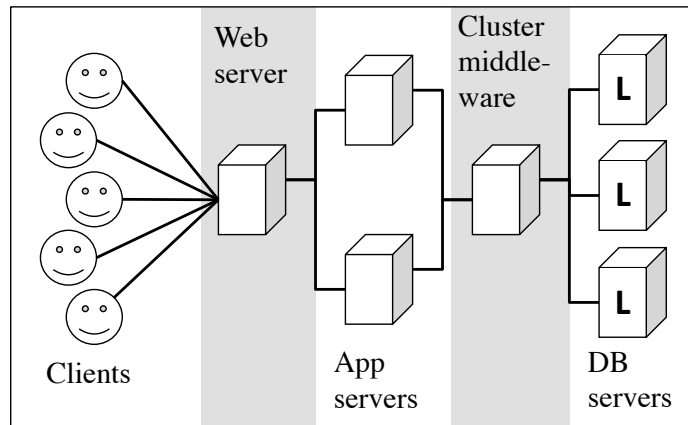


Table 3: Data set size and complexity for RUBBoS experiments.

Data metrics	Data magnitude
Number of experiments	6,318
Number of nodes (concurrent and consecutive)	91,598
Number of stored data points	3,334.1M
Total log data file size	525.6 GB

Solely for the study shown in this paper, we have used examined 6,318 experiments, which produced around 525 GB of data. For each of the four main configuration types, the total experimental data output averaged at over 277,841,000 one second granularity system metric data points (e.g., network bandwidth and memory utilization) in addition to higher-level monitoring data (e.g., response times and throughput). In the course of conducting our experiments, we have used (both concurrently and consecutively) 91,598 testbed hardware nodes. Our data warehouse is filled with around 3,334,000,000 system metric data points. In order to investigate system behavior as low as SQL query level, we have modified (if necessary) the original source code of software components to record detailed accounting logs. Because an empirical analysis of the experimental results showed that detailed logging can affect overall performance up to 8.5 percent, we also implemented effective sampling algorithms to minimize the logging performance impact. We should mention that all system performance measurements in this paper (i.e., throughput and response time) were collected without such detailed logging turned on. The latter was solely used for specific scenario analysis.

Beyond providing aggregated information that are well suited as performance characteristic reference, another one of our goals was to conduct an in-depth phenomena analysis. However, when dealing with such a large magnitude of data as presented in this publication, efficient and clear visualization by itself becomes one of the challenges that need to be resolved. For this reason, we adopted three specific types of graphical representations that facilitate visual analysis at different aggregation levels.

- We used three-dimensional bar graphs (e.g, left side of Figure 4) to visually summarize our maximum throughput results from a bird’s eye perspective. These throughput graphs depict the highest achievable throughput (z-axis) for each tested configuration in any given scenario. In most cases, each scenarios evaluated a workload span between 1,000 and 13,000 users. The x-axis represents the number of deployed database servers in the case of C-JDBC and the number of SQL nodes in the case of MySQL Cluster. The y-axis represents the number of application servers. Shading was used to indicate the throughput level from zero (dark) to the maximal achieved throughput within each scenario (light). The tick scale on the colorbar (right side of the throughput graph) explicitly shows the highest maximum throughput and the lowest maximum throughput that were achieved in the depicted scenarios.
- We used bottleneck shifting maps (e.g, right side Figure 4) to combine the maximum throughput results for each configuration with the analysis of the corresponding resource utilization data. These maps explicitly depict, in a uniform symbolic representation, the bottleneck shifting characteristics that correspond to the maximum throughput for each scenario. The various bottlenecks (“BN”) are shaded differently and labeled in accordance with their physical resource and tier. We adhered to the naming convention of “App”, “CM”, and “DB” for application, cluster middleware, and database tiers, respectively. The axes in the bottleneck maps correspond to the first two dimensions of the respective maximum throughput graph for each scenario.
- We used three-dimensional kernel regression densities to detail discovered resource utilization phenomena (e.g, hidden implementation-specific bottlenecks) at a fine granularity level. Figure 7(a) is an example of such a “low aggregation” figure. These density graphs allow to zoom into summarized resource utilization behavior without over-aggregating the data. This is particularly important if the underlying distribution is multi-modal, which we have observed in many cases (e.g., Figure 7(b)).

We used Gaussian kernel regression to generate these smoothed densities (z-axis) to better depict the changing resource utilization distribution (x-axis) as the workload grows (inverted y-axis). Note that an early version of such graphical representations have appeared in our previous work [66] to diagnose particularly challenging multi-bottlenecks.

3.3 Evaluation of Hardware Choices

In general, our experimental evaluation details RUBBoS scale-out experiments in which we increase the number of database and application tier nodes gradually to find the bottlenecks at node level (for each configuration). In particular, this section evaluates the impact of choosing different hardware node types (i.e., normal and low-cost nodes) for the backend. More concretely, we traverse an experiment space with the following constraints. The benchmark application runs on a combination of C-JDBC and MySQL servers. The clients generate a read/write mix workload. Database servers are hosted on either normal or low-cost nodes (homogeneously); whereas, all other servers are hosted on normal nodes. The system sizes range between one to three application servers and one to nine database servers. In other words, the experiments start from 1/1/1M and range to 1/3/1/9M. For each configuration, we increase the workload (i.e., number of concurrent users) from 1,000 up to 13,000 in steps of 1,000.

We should point out that due to smart memory management components (e.g., buffer manager) in typical database management systems (including MySQL and PostgreSQL), the database server uses adaptive approaches to fully utilize all available physical memory. Therefore, “raw” memory bottlenecks such as high paging activity are avoided. In this study, we analyze bottlenecks directly observable at the operating system level such as high CPU or disk utilization. The evaluation of impact of memory constraints on database servers is a topic for future research.

The first dataset evaluates an experiment set with MySQL servers deployed on normal

nodes. The graph on the left side in Figure 4 depicts the highest achievable throughput for this scenario. The highest maximum throughput is 1,478 ops/s for 1/2/1/2M, and the lowest maximum throughput is 1,122 ops/s for 1/1/1/8M. There are three groups with leveled maximum throughput that can be identified in the throughput graph. The first group (database server bottleneck) consists of configurations 1/2/1/1M and 1/3/1/1M. The second group (application server bottleneck) consists of all single application server configurations. The third group (clustering middleware bottleneck) has the highest maximum throughput level and consists of all other six configurations with at least two application servers and two database servers. The second graph on the right side in Figure 4 shows the corresponding bottleneck map. The underlying resource analysis yielded the insight that all bottlenecks are CPU bottlenecks, and the clustering middleware eventually becomes the primary bottleneck when scaling the system. Consequently, increasing the number of database and application servers is effective only in cases with relatively small numbers of servers.

While full data replication always provides the best support for read-only workload, it requires consistency management with the introduction of updates in a read/write mixed workload. In this set of experiments, we used the C-JDBC clustering middleware to provide consistency management. However, despite relaxing consistency constraints and changing the default wait-all strategy to wait-first replication (see Subsection 3.2.2 for details) the maximum throughput and bottleneck maps remained identical. This, seemingly surprising, phenomenon is caused by the delegation of transactional support in MySQL to the storage server, which is MyISAM by default. MyISAM does not provide full transactional support; specifically, it does not use Write Ahead Logging (WAL) to write changes and commit record to disk before returning a response. MySQL processes update transactions in memory if the server has sufficient hardware resources. Therefore, caused by the fast responses, the result are very small differences between wait-all and wait-first. Both cases have identical characteristics and contain the three aforementioned groups. Consequently, we have omitted the inclusion of the “duplicate” figure.

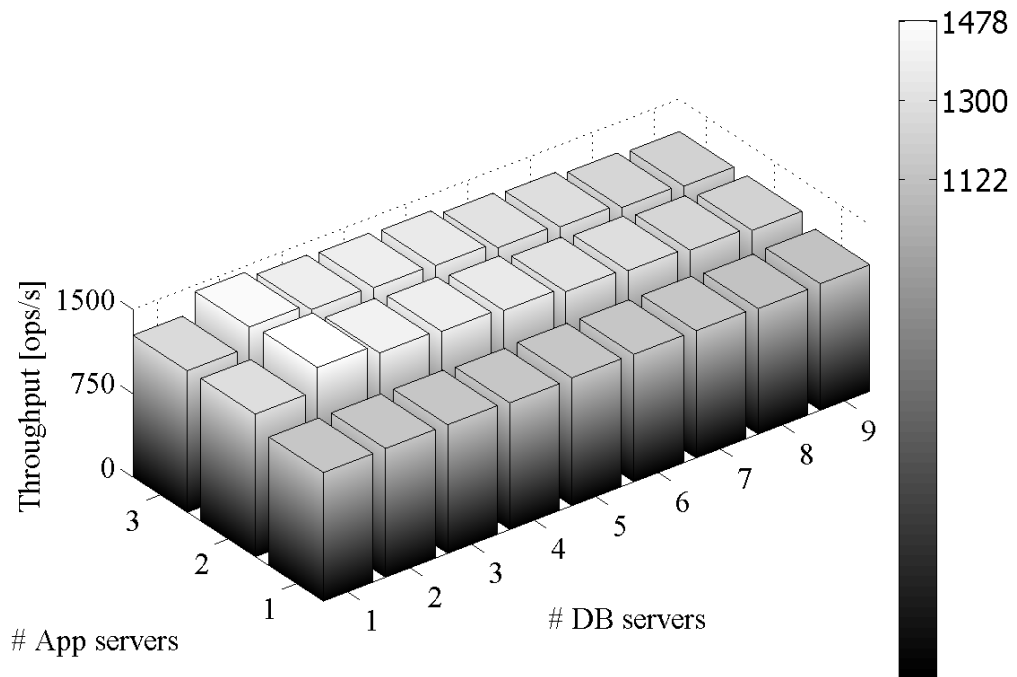


Figure 4: Maximum system throughput and bottleneck map for RUBBoS read/write workload with C-JDBC and MySQL on normal nodes.

In the following we compare the previous experiment set to a scenario with MySQL on low-cost hardware. Because of the limited memory on low-cost machines (see Table 2(a)), the throughput graphs in Figures 5 and 6 are dominated by disk and CPU utilization in the database tier. Initially, in all configurations with a single database server, the maximum throughput is mainly limited by a bottleneck in the database CPU. Figure 7 shows the corresponding database server CPU and disk utilization densities in the 1/2/1/1ML configuration. Note that no replication strategy is necessary in the case of single database servers (i.e., 1/1/1/1M to 1/3/1/1M). Both the CPU density and the disk density in Figures 7(a) and 7(b) have two modes each for higher workloads. This characteristic resource consumption corresponds to a constant shifting between resource saturation and underutilization. If both utilizations were combined into a single maximum utilization value, the density profile would display a stable saturation behavior. Both Figure 5 and Figure 6 show that this bottleneck can be resolved by replicating the database node. In other words, adding a database server increases the maximum throughput.

The single disk bottleneck becomes an oscillatory bottleneck for more than one database server with the wait-all replication strategy (Figure 5) because write queries have to await the response of all MySQL servers. An oscillatory bottleneck means that at any arbitrary point in time (usually) only one disk is saturated and that the bottleneck location changes constantly. Once a third database server has been added, the CPU bottleneck is resolved completely, and the maximum throughput remains invariant to any further hardware additions. Although the saturated resources (i.e. database disks) are located in the database tier, the C-JDBC configuration prohibits the increase of maximum throughput through additional hardware in the backend. Figure 8(a) shows that in the case of nine database nodes, the bottleneck has been further distributed among the database disks. Solely a slight mode at the high percentile values indicates an infrequent saturation. Such bottlenecks are particularly hard to detect with average utilization values. Their cause is twofold—on the one hand, all write queries are dependent on all database nodes, and on the other, the load-balancer successfully

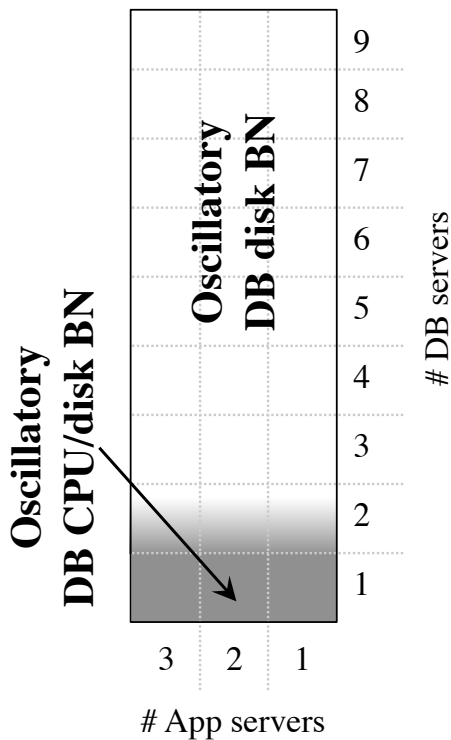
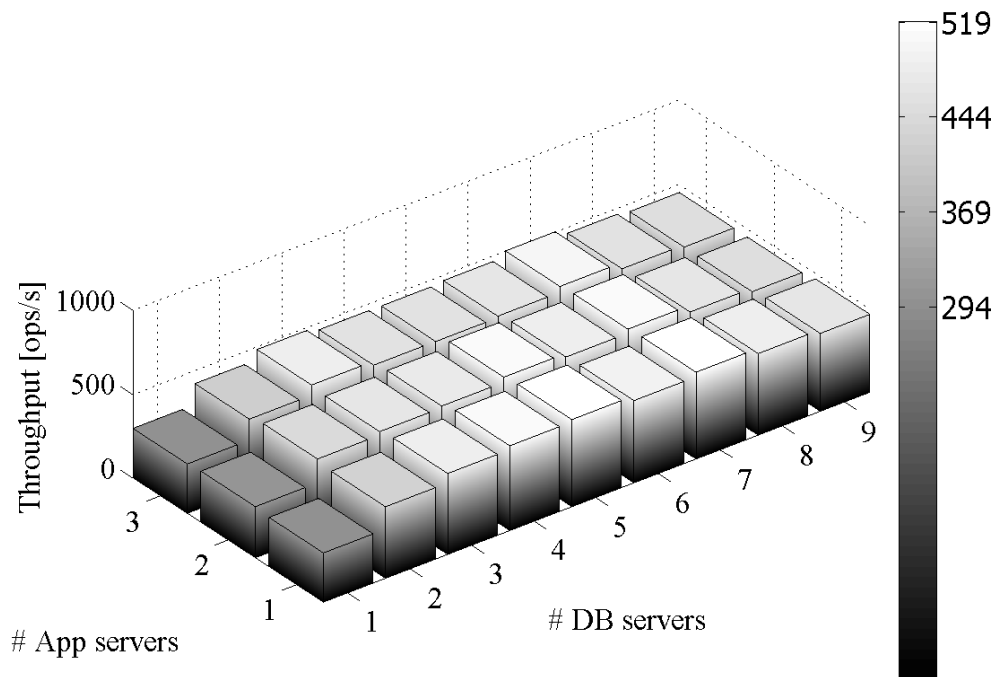


Figure 5: Maximum system throughput and bottleneck map for RUBBoS read/write workload with wait-all C-JDBC on normal nodes and MySQL on low-cost nodes.

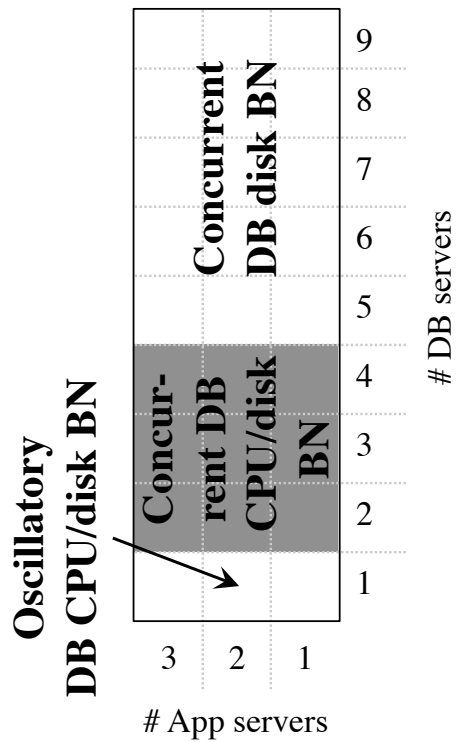
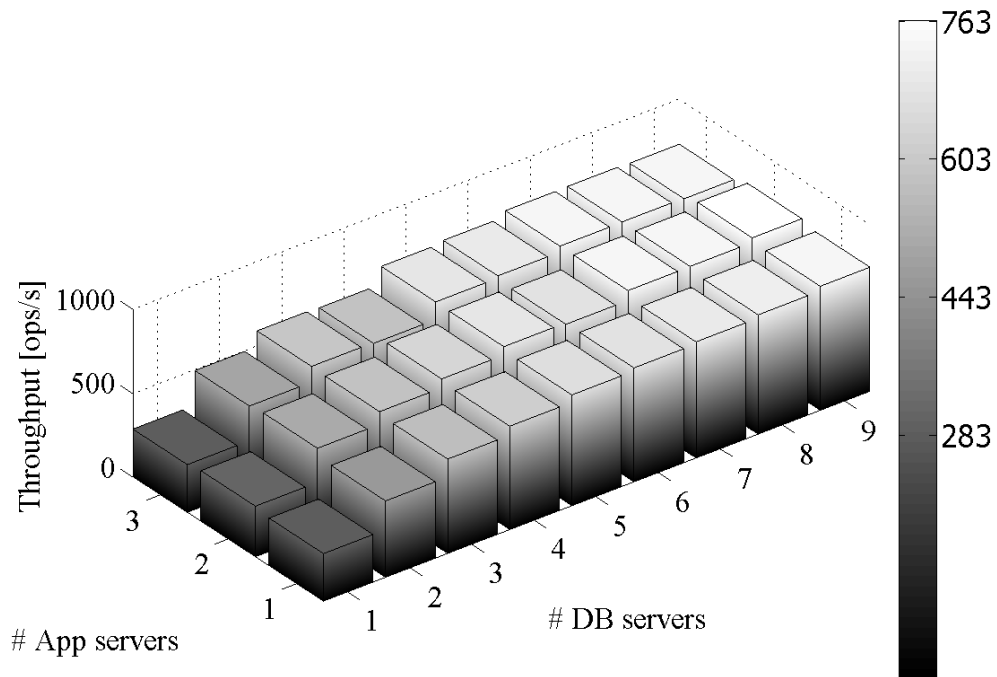
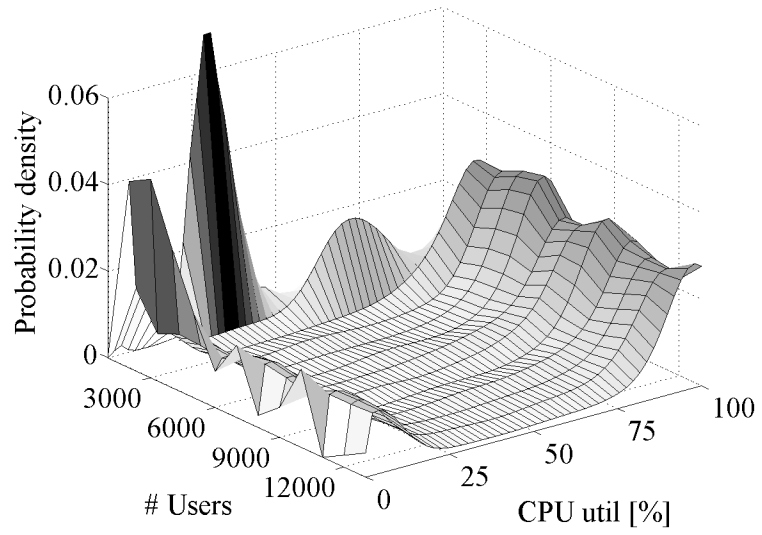
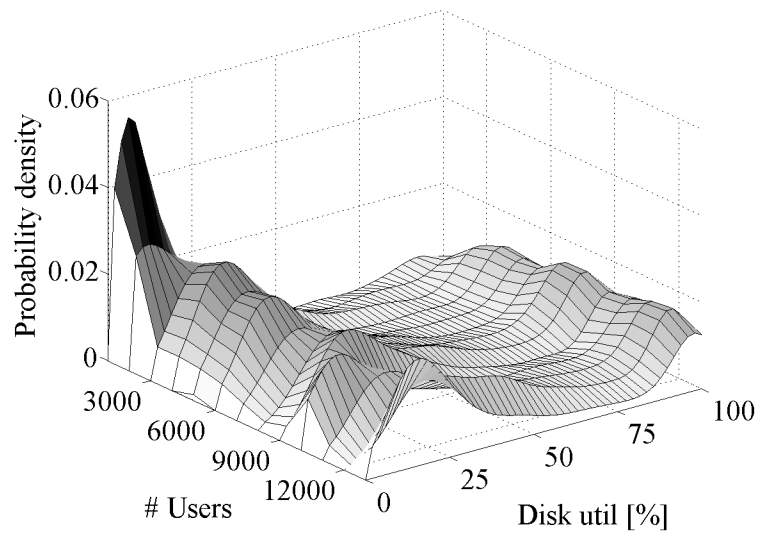


Figure 6: Maximum system throughput and bottleneck map for RUBBoS read/write workload with wait-first C-JDBC on normal nodes and MySQL on low-cost nodes.

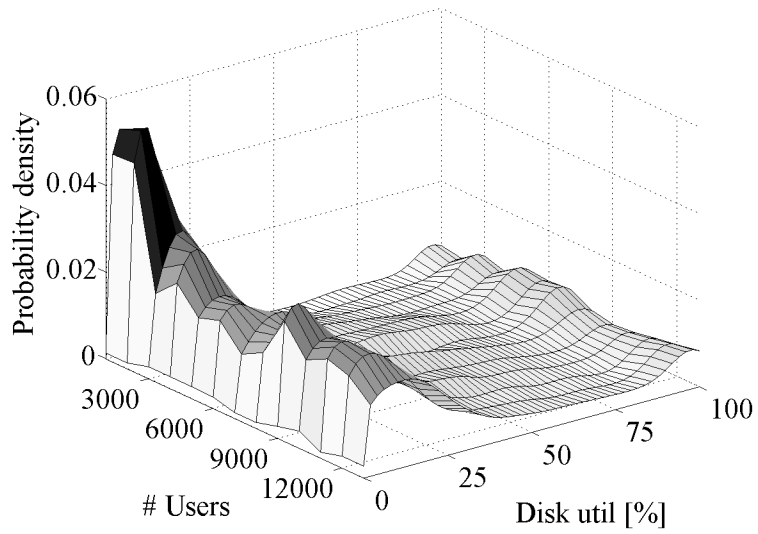


(a) Database server CPU.

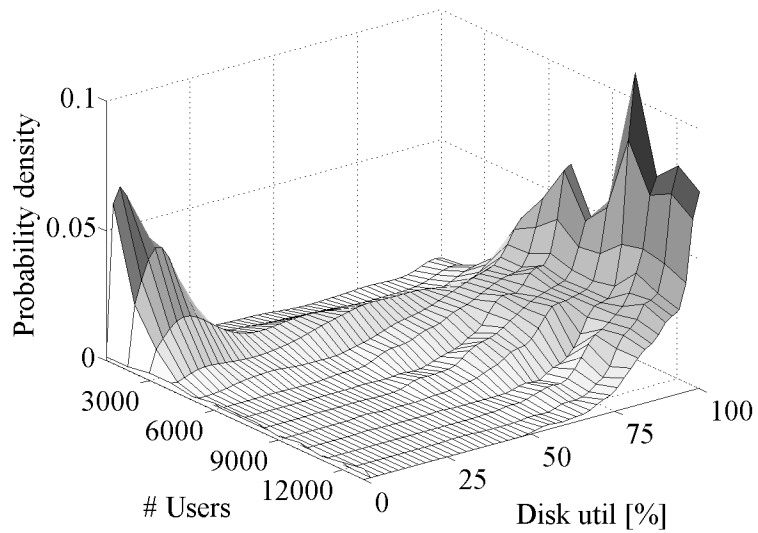


(b) Database server disk.

Figure 7: Resource utilization densities for 1/2/1/1ML RUBBoS with read/write workload, “wait-all” C-JDBC, and MySQL.



(a) First database server disk with wait-all replication strategy.



(b) First database server disk with wait-first replication strategy.

Figure 8: Resource utilization densities for 1/2/1/9ML RUBBoS with read/write workload, C-JDBC, and MySQL.

distributes the load in case one database server is executing a particularly long query.

In contrast to the wait-all case, the wait-first replication strategy enables growing maximum throughput when scaling out the database server (Figure 6). Instead of oscillatory disk bottlenecks the configurations with more than one database server have concurrent bottlenecks in the database disks (Figure 6). After the database server has been replicated five times the database disks are the only remaining bottleneck. The difference between oscillatory and concurrent bottlenecks is clearly visible in the comparison of Figures 8(a) and 8(b). Both show the 1/2/1/9ML configuration for read/write workload. While the wait-first replication strategy (Figure 8(b)) results in a clearly saturated resource, the wait-all replication strategy (Figure 8(a)) results in an infrequently utilized database disk. The actual system bottleneck lays in the combined behavior of all database disks. Due to replication overhead, the maximum throughput growth diminishes once very high database replication states are reached. In comparison to the normal hardware results, the low-cost configurations reach an overall lower throughput level for read/write workloads.

3.4 Evaluation of DBMS Choices

Unlike the previous section, which focused on evaluating the experiment space along the axis of hardware configuration, this section evaluates two different database management systems. More concretely, we alter the experiments in the previous section to evaluate differences and commonalities between MySQL and PostgreSQL database servers. Although write request to the database create an interesting idling behavior in the DB nodes (due to the PostgreSQL log manager [62]), we focus on browse-only workload results for the purpose of this section. In fact, the browse-only mix workload used in this section results in a significantly higher degree of scalability than in the case of read/write workload. Similarly to the previous experiments, database servers are hosted on either normal or low-cost nodes (homogeneously); whereas, all other servers are hosted on normal nodes. The system sizes range between one to three application servers and one to nine database servers, and the

workload ranges from 1,000 up to 13,000 in steps of 1,000.

Similarly to the previous section, the first dataset evaluates an experiment set with MySQL servers deployed on normal nodes. Figure 9 shows the highest achievable throughput (Z-axis) for experiments with varying configurations. We can again recognize three groups as in the read/write workload case. The first group (database server bottleneck) consists of configurations 1/1/1/1M, 1/2/1/1M, and 1/3/1/1M, which show similar maximum throughput for one database server (X-axis). The second group (application server bottleneck) consists of all remaining configurations with one application server (Y-axis). These configurations have a similar maximum throughput level, which is significantly higher than the throughput level of the first group. The third group (clustering middleware bottleneck) has the highest maximum throughput level and consists of the other 16 configurations (i.e., at least two application servers and two database servers).

Figure 9 is well suited to illustrate some of the typical bottlenecks observed under browse-only workload throughout our experiment set. In general, the RUBBoS benchmark places high load on the database server in the browse-only mode. Note that in this set of scale-out experiments, we have chosen full replication of database servers (i.e., any DB server can answer every query). Full data replication provides the best support for read-only workload, but it requires consistency management with the introduction of updates in a read/write mixed workload (see Section 3.3). An alternative approach (i.e., data partitioning) is evaluated in Section 3.5.

The summary of the results with the PostgreSQL DBMS is shown in Figure 10. The figure shows that PostgreSQL and MySQL share some similar throughput characteristics (e.g., the three groups from Figure 9) but different scalability characteristics. The first group contains all configurations with up to three database servers except the 1/1/1/3P case. The group is the result of a database server CPU bottleneck. The second group results from an application server CPU bottleneck and contains all configurations with a single application server and between three and nine database servers. The third group is caused by the

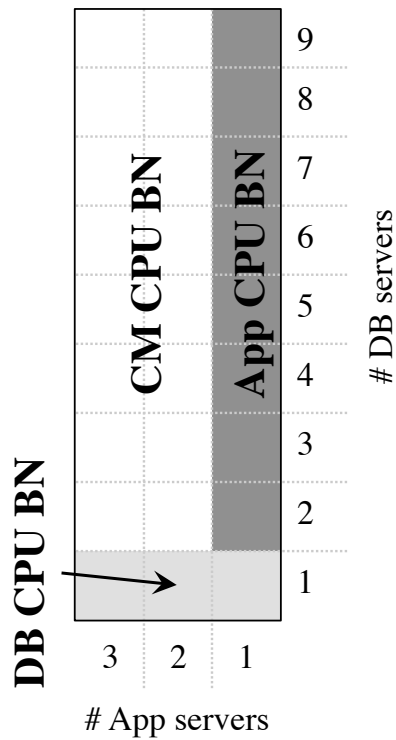
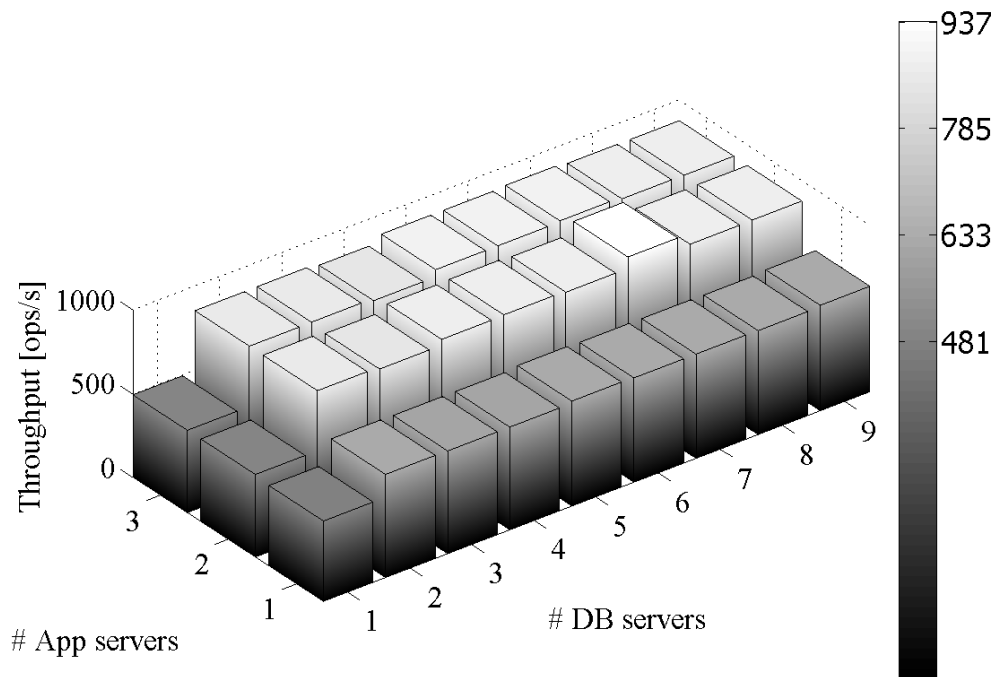


Figure 9: Maximum system throughput and bottleneck map for RUBBoS browse-only workload with C-JDBC and MySQL on normal nodes.

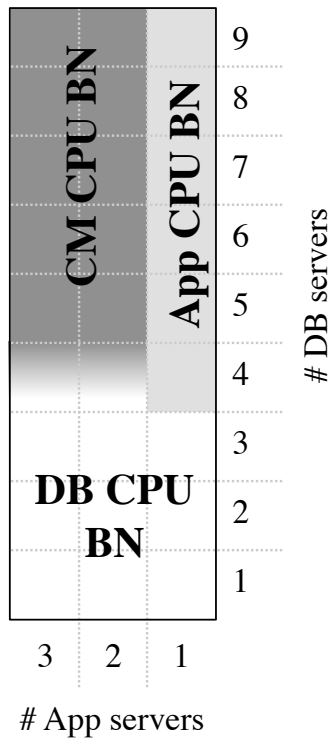
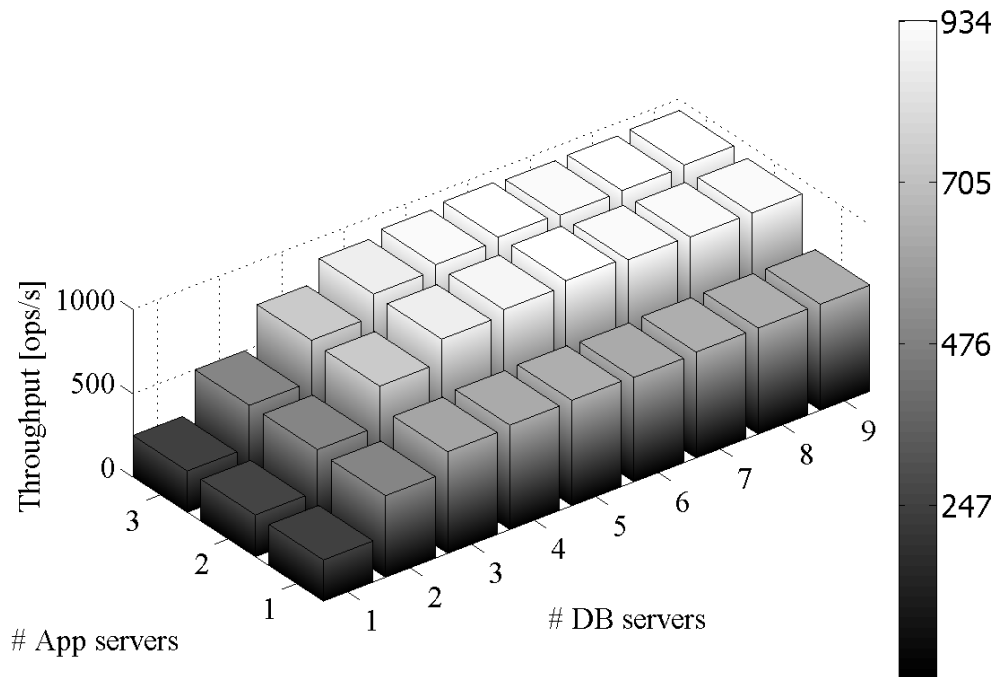


Figure 10: Maximum system throughput and bottleneck map for RUBBoS browse-only workload with C-JDBC and PostgreSQL on normal nodes.

clustering middleware CPU bottleneck for more than three database servers and more than one application server. Interestingly, for four database servers, the two configurations with more than one application server can be attributed to both groups because they concurrently saturate the CPUs in the C-JDBC and PostgreSQL tiers.

In both previously discussed DBMS scenarios the overall system eventually reaches a similar maximum throughput level of around 930 interactions per second. However, bottleneck and performance characteristics differ significantly between the two DBMSs for lower replication numbers. More concretely, PostgreSQL is solely able to match around 50 percent of the MySQL throughput when the backend is not replicated. When more database nodes are added the throughput difference shrinks (i.e., the performance of PostgreSQL improves relatively). This is caused by the MySQL system becoming bottlenecked outside the database tier for all configurations with more than one database node (C-JDBC bottleneck). Figure 9 indicates that after adding the fourth database, the PostgreSQL system matches the throughput level of MySQL. In general, these figures clearly illustrate the consequences of the clustering middleware (i.e., C-JDBC) bottleneck and how its saturation eventually limits the entire system, regardless of the DBMS.

In the following we analyze data from similar RUBBoS scale-out experiments with MySQL on low-cost hardware. Methodologically, these data offer valuable insights for practitioners when compared to both MySQL on normal nodes and PostgreSQL on normal nodes. The former comparison illustrates the performance penalty (*ceteris paribus*) of using cheaper hardware. The latter comparison investigates a scenario that leverages the more complex database engine of PostgreSQL when compared to MySQL's simpler MyISAM through hardware scale-up. Given our previous explanations and the transitivity of these results, the following analysis compares the low-cost results to the baseline of MySQL on normal nodes.

The summary of the experimental results for low-cost hardware (i.e., maximum throughput and bottleneck behavior) is shown in Figure 11. The figure reveals that maximum

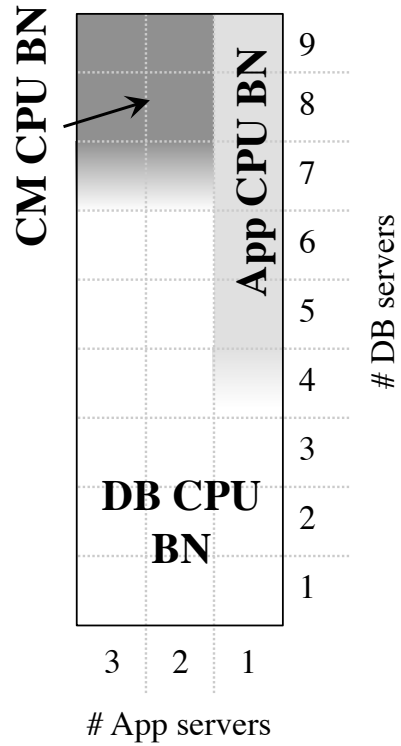
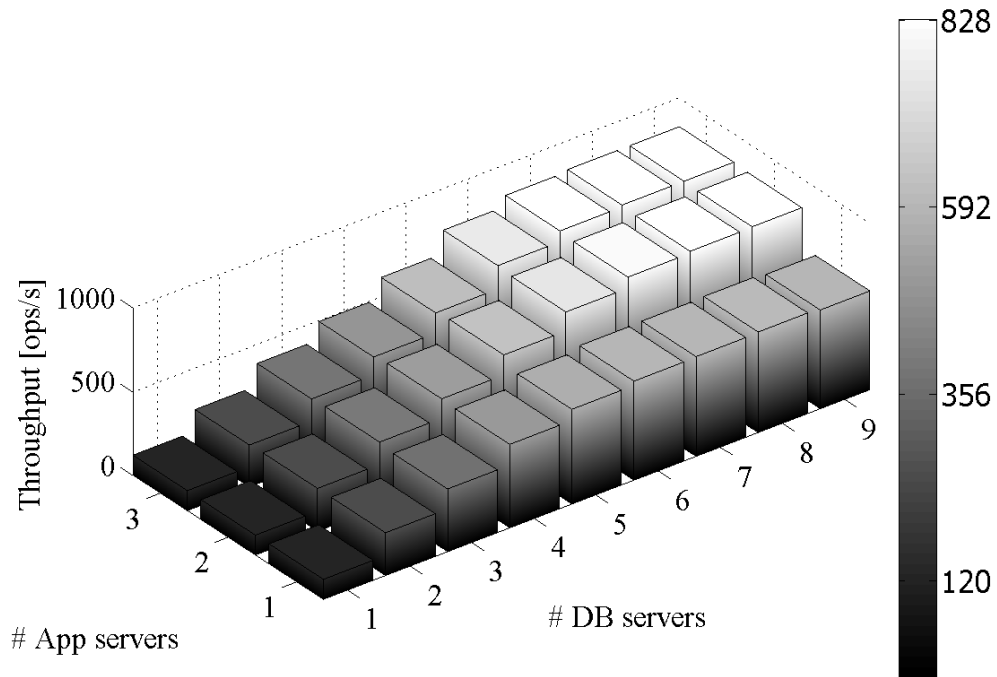


Figure 11: Maximum system throughput and bottleneck map for RUBBoS browse-only workload with C-JDBC on normal nodes and MySQL on low-cost nodes.

throughput of small configurations (i.e., low replication degree) can simply be increased by replicating the backend. Consistent with our previous discussion, this is caused by database CPU bottlenecks in all configurations with less than four database servers. These bottlenecks can be resolved by adding a server node to the database tier. However, the performance gain of additional database servers in configurations with just one application server diminishes significantly in the 1/1/1/4ML case because the primary bottleneck starts to migrate to the application server tier. For all higher database replication states (i.e., at least five database servers and only one application server) the maximum throughput remains stable. For all configurations with at least two application servers, the system performance is scalable up to seven database nodes. After that point the primary bottleneck starts shifting from the backend tier to the clustering middleware. Once the CPU of the C-JDBC server determines the system throughput, the addition of database and application servers has no significant performance effect anymore.

The performance comparison of normal and low-cost nodes in the C-JDBC MySQL scenarios shows that for low replication numbers, the low-cost nodes are not able to match the performance of the normal nodes. In case of a single database node, the performance ratio can be as low as 30 percent. However, similarly to the previous comparison, a rapid relative performance improvement is observable when adding more backend nodes. With more than seven database servers in the two application server case, the low-cost nodes come within a ten percent range of the normal hardware performance. These performance characteristics can again be explained with the underlying bottleneck behavior. More generally, if the same primary bottleneck determines the system throughput, the performance implications of different submodules may diminish significantly.

In essence, these results explicitly reveal the performance map that is necessary to assess the utility of one hardware choice over the other. When combined with cost estimates of these configurations, these data can directly be applied in configuration planning, which is a good use-case example for our approach [63]. In fact, these results also document the threat

of assuming a single-class workload in performance models. Under such an assumption some of the observed characteristics (e.g., maximum throughput delta) are not explicable. Therefore, the results presented in this section can typically not be obtained analytically.

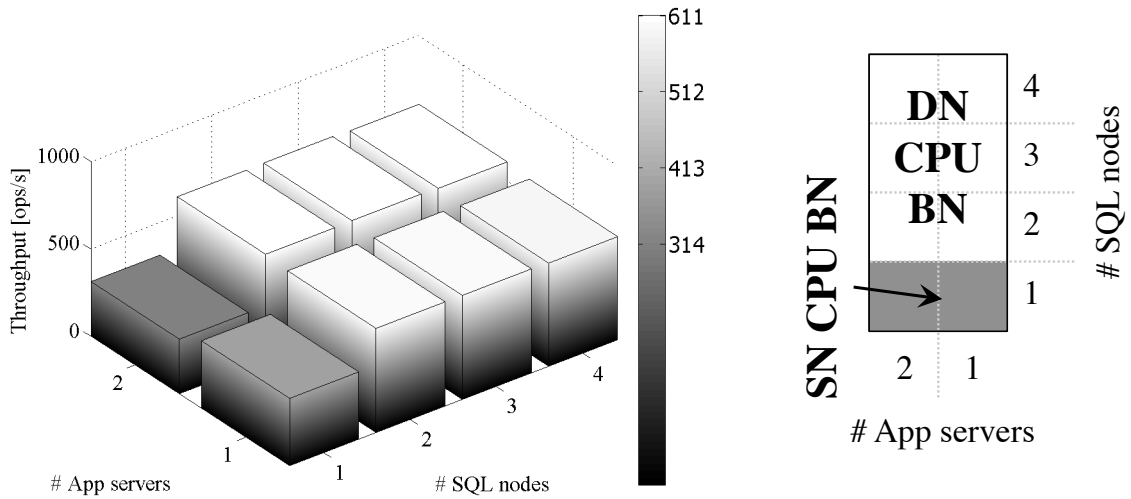
3.5 Evaluation of Middleware Choices

In order to explicitly investigate the effects of different middleware components that implement different replication technologies, we have conducted a similar set of experiments as discussed in the previous sections with MySQL Cluster in lieu of C-JDBC. In other words, this section evaluates the impact of choosing different approaches to database replication. It should also be noted that in our configurations MySQL Cluster was configured as “in-memory” DBMS (see Section 3.2.2).

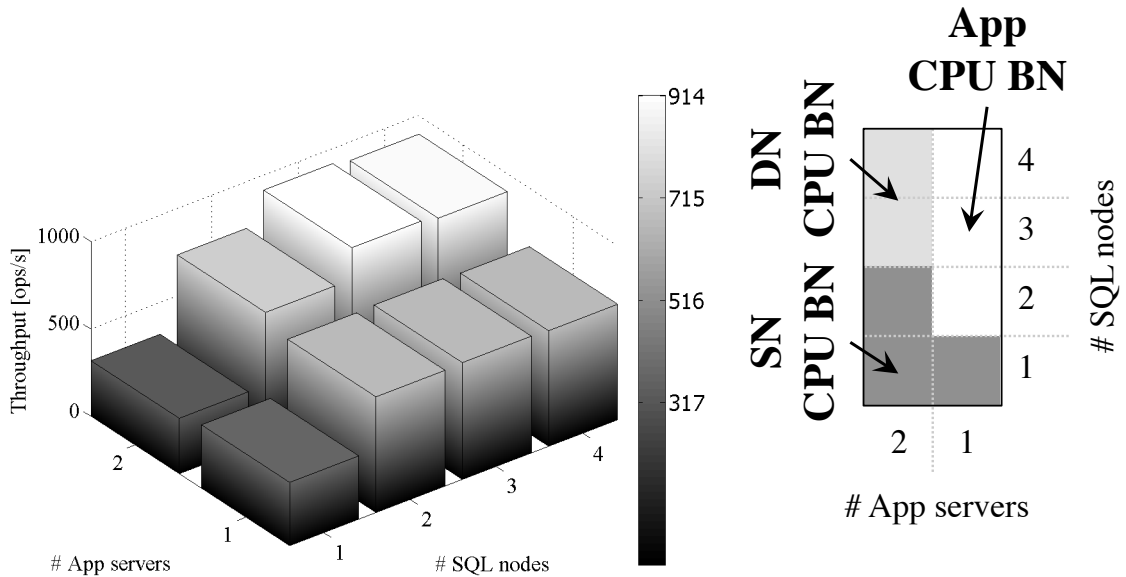
Concretely, we traverse an experiment space with the following constraints in this section. The benchmark is deployed with a MySQL Cluster backend. The clients generate a read/write mix workload. All backend nodes are hosted on normal hardware. The system sizes range between one to two application servers and up to nine nodes in the database tier. For each configuration, we increase the number of concurrent users from 1,000 up to 13,000 in steps of 1,000.

In the first set of experiments, we have run the system without partitioning the database, which implies two data nodes² and one management node. The system was scaled-out by increasing the number of MySQL servers from one to four and varying between one or two application servers. The summary of the results is shown in Figure 12(a). As illustrated in the graph, the maximum throughput that can be achieved is seemingly low compared to the same number of nodes with C-JDBC MySQL (see Figure 4). The main reason for this characteristic is evident from the bottleneck analysis in Figure 12(a). The bottleneck map shows that the MySQL data nodes rapidly become the system bottleneck when scaling-out.

²MySQL Cluster has a restriction that the number of data nodes cannot be increased without partitioning the dataset.



(a) MySQL Cluster configuration with two data nodes.



(b) MySQL Cluster configuration with four data nodes.

Figure 12: Maximum system throughput and bottleneck map for RUBBoS read/write workload with MySQL Cluster on normal nodes.

To achieve higher system throughput, this bottleneck needs to be alleviated through replication. Therefore, we have partitioned the database into two parts using the MySQL Cluster default partitioning mechanism and repeated the set of experiments with four data nodes. The performance summary is shown in Figure 12(b). The comparison of Figures 12(a) and 12(b) reveals that increasing the number of data nodes does, in fact, increase the maximum system throughput significantly. The bottleneck analysis in Figure 12(b) illustrates how splitting the database causes the bottlenecks to shift from the data nodes to the application servers initially. When the number of SQL nodes is increased further, the bottlenecks shift back to the data nodes. Consequently, our empirical analysis reveals that the system is not bottlenecked in the MySQL servers in the case of read/write workload, which would have been a potential a priori assumption. Instead, the data node CPUs eventually become the throughput-determining factor. In order to get throughput levels that are comparable to the C-JDBC MySQL experiments, it is necessary to partition the data further in order to use more than 4 data nodes. Most notably, such scale-out requires significantly more hardware nodes than have been used in the baseline experiments.

3.6 Evaluating Implementation-specific Bottlenecks

As the number of hardware nodes increases, physical resources become abundant, and system bottlenecks migrate. All previously discussed experiment set are examples of this phenomenon, in one way or the other. In contrast, implementation-specific bottlenecks are caused by certain design and configuration choices, which make database scale-out performance a highly non-linear function of number of servers. In the following we show two examples of database management system and clustering middleware implementation and configuration settings that cause a small number of queries to have five to six orders of magnitude longer execution times. We show that these problematic queries have a highly significant effect on overall system performance, despite their extremely small numbers. Such queries were difficult to identify since they are similar to normal queries,

both syntactically and semantically. Considerable effort and database management system expertise were required for the findings described in this section.

In Subsection 3.6.1 we analyze “heavy read queries” in the C-JDBC 1/2/1/9ML configuration under browse-only workload and find overly long read queries on the MySQL low-end nodes to cause a problematic load-balancing effect in the clustering middleware. In Subsection 3.6.2 we examine “hidden” bottlenecks in a C-JDBC 1/2/1/5P configuration under browse-only workload, and identify their cause as an self-join limitation in the query-optimizer of PostgreSQL.

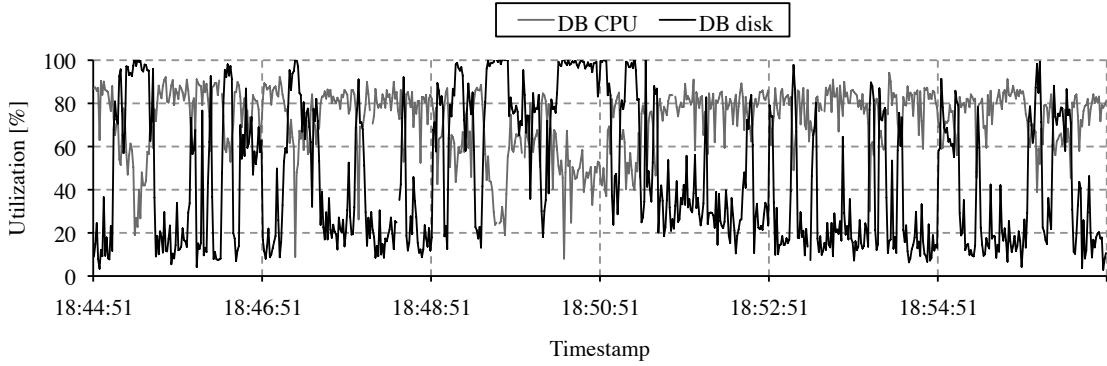
3.6.1 Long Read-queries in Low-end MySQL Scenario

Complex queries on low-end nodes require optimization in order to perform well. To optimize certain queries which search a big table (“old_comments”), MySQL has two specific requirements. First the length of the search strings for the column “subject” may not exceed ten characters, and second, the specified offset must be zero. The first requirement is due to a faulty indexing scheme in RUBBoS. However, the following example query violates both requirements.

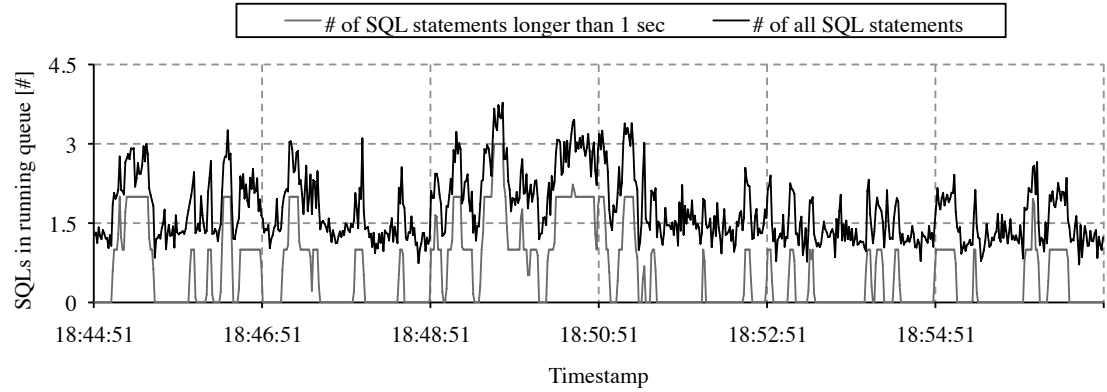
```
SELECT
    old_comments.id, old_comments.story_id,
    old_comments.subject, old_comments.date,
    users.nickname
FROM old_comments, users
WHERE
    subject LIKE 'independently%' AND
    old_comments.writer = users.id
ORDER BY date DESC
LIMIT 25 OFFSET 25;
```

If a query cannot be optimized, the entire table is retrieved for brute force query processing. In the case of low-end nodes and browse-only workload, this leads to extremely long response time because the table is retrieved from disk. In Figure 13(a) the effect of these queries is identifiable in disk I/O intervals at 100 percent bandwidth utilization. However, despite the intuitive assumption that such disk I/O is the direct cause for the correlated drop in CPU utilization, we find the actual reason for low overall throughput to lay in the load-balancing policy of C-JDBC. The latter allocates load to the database servers according to the number of running SQLs on each node. If a certain server is assumed loaded, the load-balancer moves to the next higher replica number. (This strategy usually achieves higher performance than round-robin.) When “heavy read queries” arrive at the database management system, the long disk access times cause the average number of SQLs in the running queue to increase (see Figure 13(b)). Although this does not imply that the CPU is overloaded, the load-balancing algorithm in the clustering middleware distributes the load to other databases. Consequently, the number of executed SQLs drops rapidly (Figure 13(c)), which is strongly correlated with the CPU consumption in Figure 13(a).

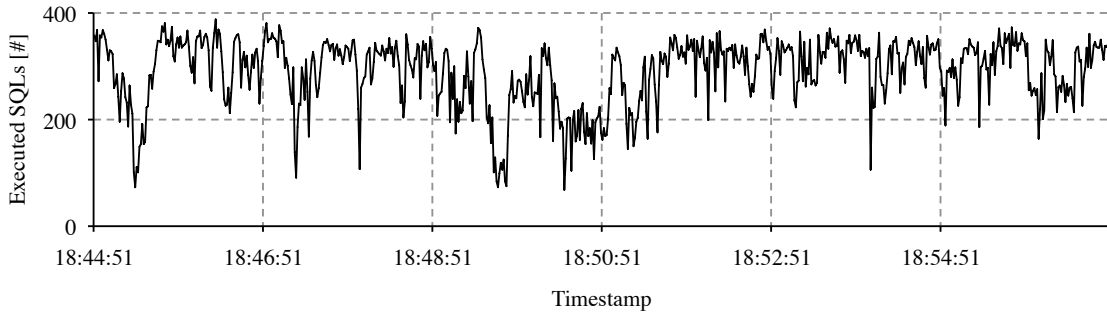
One conclusion from these observations is that the load-balancing of multiple databases is a difficult problem when more than one resource, such as the CPU and the disk, are saturated. A sophisticated algorithm is necessary for a load-balancer to handle more than one independent resources efficiently, especially in the case of non-stationary request streams. Monitoring the number of active SQLs is not sufficient to resolve such more complex bottleneck scenarios. Table 4 shows that the number of such problematic queries is small (compared to total SQL counts) in our experiment. However, their processing times are so large that the overall performance is skewed significantly. This characteristic behavior only occurs if the database is not the primary system bottleneck. In the presented scenario, this means that the C-JDBC node’s CPU is the primary bottleneck (Figure 14(a)). The database servers (predominantly) operate under unsaturated conditions at around 80 percent CPU utilization (Figure 14(b)) and low disk utilization (Figure 14(c)).



(a) Time series of the first database server resource utilization.

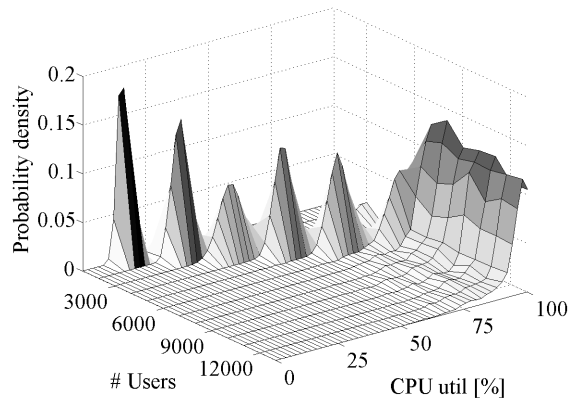


(b) Time series of the approximated number of SQLs in the first database server.

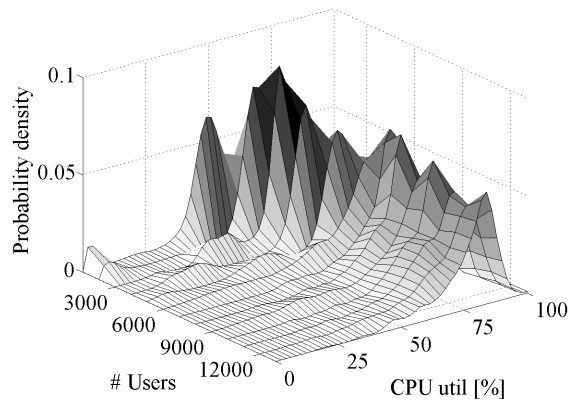


(c) Time series of number of executed SQLs in the database server.

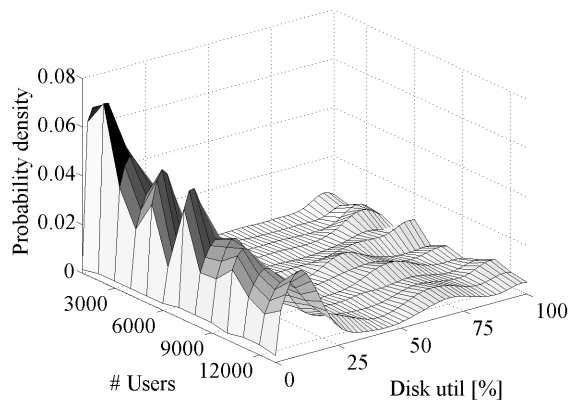
Figure 13: Detailed analysis of the 1/2/1/9ML C-JDBC scenario for 8,000 users scenario with browse-only workload.



(a) Cluster middleware CPU utilization density.



(b) First database server CPU utilization density.



(c) First database server disk utilization density.

Figure 14: Detailed analysis of the utilization densities in the 1/2/1/9ML C-JDBC scenario with browse-only workload.

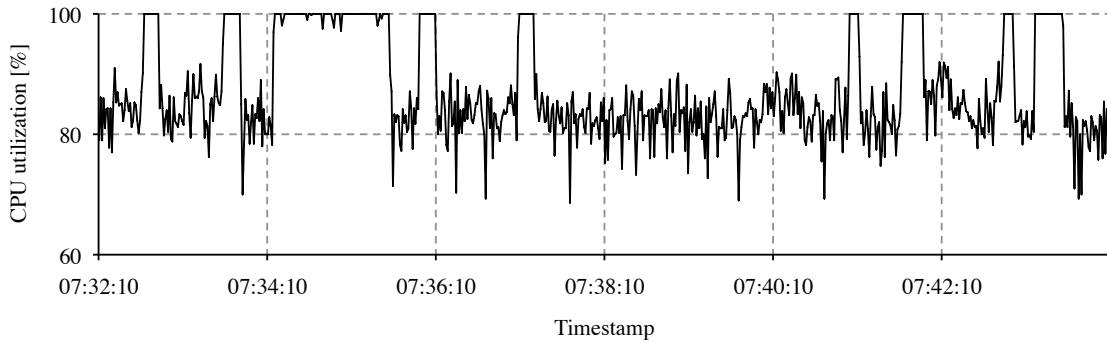
Table 4: Distribution of SQL response times in the first database server of the 1/2/1/9ML C-JDBC scenario for 8,000 users scenario with browse-only workload.

Resp. time	Count	Frac. [%]
0.01ms–0.1ms	522	0.250
0.1ms–1ms	42,866	20.449
1ms–10ms	154,505	73.887
10ms–100ms	10,727	5.130
100ms–1s	446	0.213
1s–10s	31	0.015
10s–100s	14	0.007
Total	209,111	100.000

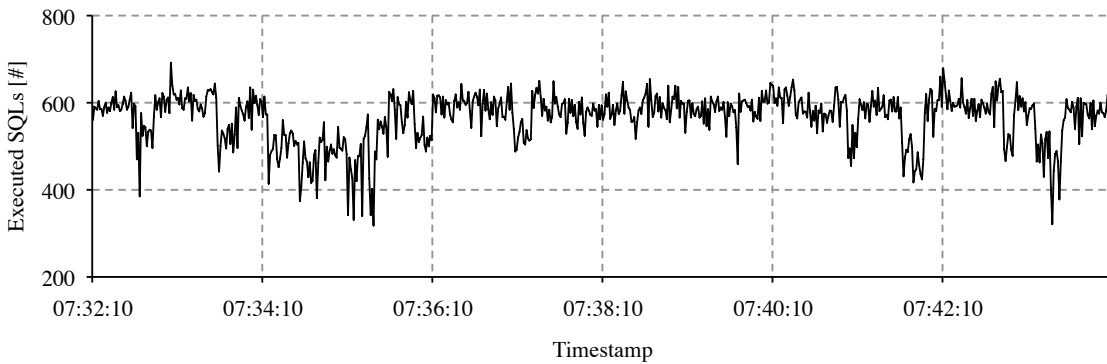
3.6.2 Self-join Limitation in PostgreSQL Scenario

In this subsection we analyze the performance of C-JDBC PostgreSQL on normal nodes. Unlike the previous case, the hardware in the database tier possesses enough memory to cause almost no disk I/O in the browse-only case. However, a different problematic query, which results in occasional database server CPU utilization saturation (Figure 15(a)) and a corresponding drop in executed SQLs (Figure 15(b)), arises. These utilization peaks influence the CPU utilization density of the database servers significantly. This generates a density mode at the high percentile end of the utilization spectrum that becomes visible when the x-axis is inverted (Figure 16(a)). While in the previous section the clustering middleware was unable to resolve such a load balancing issue, the number of active SQLs in the running queue (Figure 15(c)) is a good indicator of actual CPU load in this scenario. Therefore, the load is distributed successfully among the other database servers, and the database CPU is not the primary bottleneck in this configuration and load scenario.

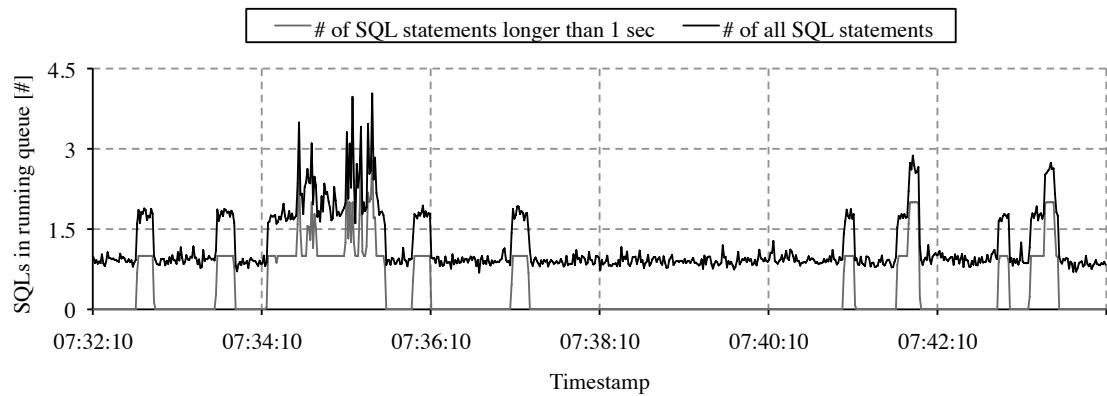
In order to infer what causes the observed non-stationary behavior, it was necessary to use the SQL logging facilities of the database server. The response time distribution (see Table 16(b)) shows a diminishing number of queries, which are responsible for the overall degradation of average system performance. A careful log analysis reveals that all long-lasting read queries, which take more than two seconds, are of the same type. An example of such a query is shown at the end of this subsection.



(a) Time series of the first database server CPU utilization.

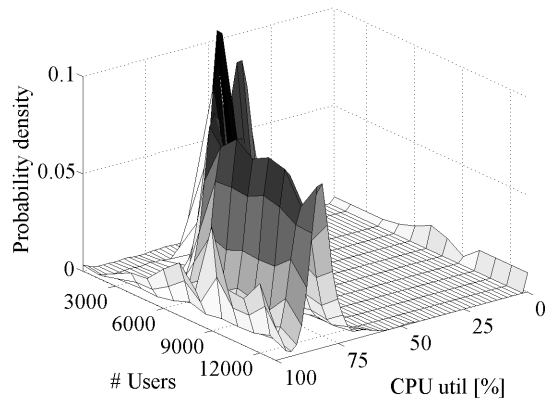


(b) Time series of number of executed SQLs in the database server.



(c) Time series of the approximated number of SQLs in the first database server.

Figure 15: Time series analysis of 1/2/1/5P scenario for 9,000 users with a browse-only workload.



(a) First database server CPU utilization density.

Resp. time	Count	Frac. [%]
0.1ms–1ms	235,487	57.800
1ms–10ms	167,788	41.183
10ms–100ms	4,085	1.003
100ms–1s	31	0.008
1s–10s	21	0.005
10s–100s	7	0.002
Total	407,419	100.000

(b) Distribution of SQL response times in first database server.

Figure 16: Detailed analysis of 1/2/1/5P scenario for 9,000 users with a browse-only workload.

In fact, the query optimizer in PostgreSQL appears to handle queries with more than nine times self-joins (i.e., queries which join the same table more than nine times as in the example above) differently in our experimental configuration. The processing of such queries causes the saturation of the database CPU. It should be noted that the query optimizer of PostgreSQL can handle up to nine times self-joins very well. In the case of nine times self-joins, the average response time is only 17 ms. Naturally, this type of self-join query also happens in experiments with MySQL. In that case, MySQL executes such queries as quick as other queries even though they are more than nine times self-joins. In the scenario at hand, the C-JDBC detects the “heavy query” overload situation and distributes further requests among the other database server replicas. Therefore, the number of executed SQLs (see Figure 15(b)) drops rapidly, and the CPU is able to reduce the queue size (see Figure 15(c)) after the inefficient query has been processed. In general, this kind of problematic queries can constitute a serious performance threat, in case several of them reach the database tier at the same time. If they occur solely sequentially, the load-balancer is able to resolve this problem.

```
SELECT
```

```
    t11.id, t11.parent, t11.childs, t11.rating, t11.date,  
    t11.subject, t11.comment, users.nickname
```

```
FROM
```

```
users, comments AS t0, comments AS t1, comments AS t2,  
comments AS t3, comments AS t4, comments AS t5,  
comments AS t6, comments AS t7, comments AS t8,  
comments AS t9, comments AS t10, comments AS t11
```

```
WHERE
```

```
t11.parent = t10.id AND t10.parent = t9.id AND  
t9.parent = t8.id AND t8.parent = t7.id AND  
t7.parent = t6.id AND t6.parent = t5.id AND
```

```
t5.parent = t4.id AND t4.parent = t3.id AND  
t3.parent = t2.id AND t2.parent = t1.id AND  
t1.parent = t0.id AND t0.parent = 0 AND  
t11.writer=users.id AND t11.story_id=286;
```

3.7 Related Work

Cloud computing, Green IT, server consolidation, and virtualization have become ubiquitous terminology in the times of ever-growing complexity in large-scale computer systems and increasing awareness of ecological implications. However, many hard problems remain to be solved on the way to an optimal balance between performance, economy, and ecology. Specifically, scaling systems along the two axes of performance and availability is particularly difficult with regard to database servers. Database replication is widely accepted as common mechanism of choice when scale-up potential is exhausted although replication often falls short of real-world user needs. This leads to a continuous twofold trend. New replication solutions continue to emerge and real-world database clusters are often comparably small (i.e., less than 5 replicas) [27]. In this paper we address these shortcomings explicitly.

A central problem in enterprise system studies is that scalability evaluations traditionally only address scaled load scenarios to determine best achievable performance (e.g., [29]). Concretely, experimentation methodologies with 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 solution impact in the real world [27]. Moreover, it is well understood that transactional replication can exhibit unstable scale-up behavior [39], and it is not sufficient to characterize replication performance in terms of peak throughput [27].

There exist many popular solutions to database replication, and the most common

commercial technique is master-slave replication. Some replication product examples are IBM DB2 DataPropagator, replication in Microsoft SQL Server, Sybase Replication Server, Oracle Streams, replication in MySQL. Academic prototypes that use master-slave replication are Ganymed [79] and Slony-I [1]. The latter is a popular replication system supporting cascading and failover. However, in this work we focus on multi-master architectures. Some commercial products that use this technique are MySQL Cluster and DB2 Integrated Cluster. Academic multi-master prototypes are C-JDBC [30], which is used in this work, Tashkent [35], and Middle-R [78]. Further efforts focus on adaptive middleware to achieve performance adaptation in case of workload variations [73].

The evaluation of database replication techniques often falls short of real representability. There are two common approaches to evaluation with benchmarks. Performance is either tested through microbenchmarks [78] or through web-based distributed benchmark applications such as TPC-W, RUBiS, and RUBBoS [15]. These benchmark applications are modeled on real applications (e.g., Slashdot.org or Ebay.com), offering real production system significance. Therefore this approach has found a growing group of advocates [29, 30, 35, 79, 81].

Traditional performance analysis in computer systems presumes models based on expert knowledge, employs standard statistical methods, and parameterizes them based on a certain experimentation design [46, 58]. Queuing models have been common practice in many research efforts dealing with performance prediction [94, 98]. Although these approaches have been applied very successfully, they suffer from their rigid assumptions when handling all evolution of large applications. The availability of extensive instrumentation data profiles [94] or constant mean inter-arrival times of request [98] do not hold in general since actual parameters vary widely in real applications. The characteristic load non-stationarity in n-tier systems has been exploited for performance prediction by Stewart et al. [93], who also explain how their anomaly detection can be used to invoke a bottleneck detection process.

Most recent literature on the topic of database replication deals with specific implementation aspects such as transparent techniques for scaling dynamic content web sites [13]. The experimental evaluation in such works often remains narrow in focus and is solely used to confirm a priori defined properties. The goals of our approach are different in several ways. We investigate system performance from a much broader perspective and do not define concrete validation goals a priori. Moreover, this work builds on or is closely linked to several of our previous publications. We first introduced our large-scale observation-based approach to performance characterization of distributed n-tier applications with main focus on the RUBiS benchmark [81]. Correspondingly, the notion of bottleneck detection through data analysis based on heuristics was coupled with the obtained dataset [65]. These findings branched out in three directions. First, our insights in bottleneck detection required a formalization of multi-bottleneck phenomena in complex systems [66]. Second, augmenting our bottleneck detection methodology with a statistical workload prediction framework led to a Green IT provisioning model [43]. Third, focusing on performance in terms of phenomena and targeting scalability issues of the backend led to the large RUBBoS dataset presented in this paper. While this paper provides a complete overview of our exceptionally rich dataset by presenting the most interesting findings, we have previously addressed read/write workload [62] and browse-only workload [67] in preliminary versions of this work. Unlike these two works, this paper focuses on the complete picture and provides a holistic perspective of in-depth empirical database server scalability analysis. A significant amount of previously unpublished explanations, data, and analyses have been included in this paper to further strengthen its unique contribution. A good exemplary application of findings and data, such as presented in this paper, is empirically-based configuration planning [63].

3.8 Future Work

There are several possible directions for follow-up research based on the promising results in this paper. Methodologically, our approach has covered new ground, and the shown results

clearly corroborate the value of empirical database server analysis. Nonetheless, the full formalization of the presented methodology will still require substantial effort. Although our performance analysis has discovered many interesting characteristics of the analyzed systems, further exploration at different aggregation levels will be required in order to grasp the full information contained in our dataset. In other words, we believe that further study (e.g., through refined analysis of our data or new experimental data) may bring new understanding of database server scale-out in n-tier applications. In addition, the presented bottleneck phenomena illustrate how our analysis is able to reveal “hidden” and unexpected system characteristics. One potential direction for future work is the discovery of new and previously unseen bottleneck phenomena similarly to the ones shown in this work. Finally, multi-bottleneck detection still presents many challenges to administrators of large and complex systems. Further work is necessary to facilitate multi-bottleneck detection, diagnosis, and automated alleviation.

3.9 Conclusion

We used techniques and tools developed for automated experiment management, including automatically generated scripts for execution and measurement, to study parallel database server scalability in the RUBBoS n-tier benchmark. Based on this dataset and the subsequent evaluation, we have generated a broad reference of scalability characteristics. Moreover, these data facilitated in-depth analysis of particularly interesting bottleneck characteristics. The identification of non-obvious single- and multi-bottlenecks showed how migratory bottleneck phenomena can arise in real systems and result in surprising yet significant performance effects.

In general, the explicit analysis of varying system configurations yielded the identification of performance characteristics that would escape classic analytical approaches. An example is the finding of infrequent bottlenecks caused by PostgreSQL’s differentiated processing of SQL queries with more than nine times self-joins, which seems to require

several orders of magnitude longer CPU time. Another example are oscillatory bottlenecks introduced by the wait-all replication strategy, which waits for all the replicated database servers to complete an update before considering the update finished. Although no single hardware bottleneck exists in the system, as the number of database servers increased, the replicated database resources show a combined bottleneck behavior. We showed that such cases may be particularly challenging to detect and analyze successfully.

Our results suggest that techniques used in the “tuning” of database management systems to improve performance may be useful in the identification of internal bottlenecks that can take a critical role in achieving service level objectives in mission critical applications. While our data analysis shows that there are many similarities between database and application/web server scalability, our results also clearly document that distributed database servers have significantly higher sophistication and complexity that require further in-depth evaluation and analysis similar to our approach.

3.10 Appendix

In this appendix we provide background that is of particular importance for the understanding of our approach. Subsection 3.10.1 discusses the use of code generation. Our network topology and deployment specifics are the topic of Subsection 3.10.2. Subsection 3.10.3 introduces our experiment cycle, and Subsection 3.10.4 provides more specifics on data point collection in our experiment context. Finally, in Subsection 3.10.5, we touch upon our taxonomy of bottleneck phenomena.

3.10.1 Code Generation

The main focus of our research is the identification of representative phenomena with potentially wide applicability. Running experiments in n-tier system with actual workloads and analyzing the data, introduces a set of new challenges compared to workloads simulation. To improve the soundness (precision) and correctness (accuracy) of our experimental data we have run a very high number of experiments over a wide range of configurations and

workloads. A typical experiment consist of different types of servers and different types of application running on distributed environment, thus deploying, configuring, monitoring and data collecting are inherently difficult task. Although the deployment, configuration, execution, and analysis scripts contain a high degree of similarity, the differences among them are subtle and important due to the dependencies among the varying parameters. Maintaining these scripts by hand is a notoriously expensive and error-prone process. In order to enable experimentation at this large scale, we used an experimental infrastructure created for the Elba project to automate n-tier system configuration management. The Elba approach [81] divides each automated staging iteration into steps such as converting policies into resource assignments [86], automated code generation [50], benchmark execution, and analysis of results.

3.10.2 Network Topology and Deployment

In our experiments we consider as many possible configuration settings as possible and analyze the system based on those settings. Running this large number of experiments, we have found that network topology may also effect performance results for the n-tier systems. Therefore, we have modified the network topology in accordance with the type of configuration and real representation. In most of the cases we have used a hierarchical mesh network.

In experimental testing, when we want to improve the empirical data to be more deterministic, the testing environment and environmental conditions become key considerations. To have more deterministic experimental data for each experiment, we start the experiments using fresh a operating system snapshot. We then deploy applications and configure them using our automated code generation tools. With help of automated code-generation these tedious tasks (e.g., deploy, configure) are done in the push of a button.

3.10.3 Experimental Cycle

Our experimental cycle consists of numerous stages, and each stage consists of number of different steps. In the first stage, we boot-up the nodes with the fresh operating system image then we tailor our code generator input template to meet the specific need of the experiment both hardware and software wise. Our automated code generator will then generate instruction and configuration scripts that are used toward driving the experiment. In the second stage, upon finishing the generation, initial steps are launched automatically. Considering the dependency structure between applications, appropriate applications may be deployed in parallel to each tier. After all preparation steps (e.g., configuring and starting) are terminated, we pick the first workload from the list and execute the browse-only scenario first. We collect data, stop all the servers, restart them, and run the corresponding read/write scenario. Next, we reset all the servers to their initial settings and run the next workload; likewise, we iterate these steps until we finish all the required workloads. During the final stage, we dump all data and upload them into our data warehouse for analysis.

3.10.4 Data Point Collection

The empirical dataset that is used in this work is part of an ongoing effort for generation and analysis of n-tier performance data. We have already run a very high number of experiments over a wide range of configurations and workloads in various environments. When we run our experiments, we collect as much data as possible without affecting the system behavior significantly (i.e., transparent monitoring). For this purpose, we use various lightweight techniques to collect system and application data points. In most cases, we use primitive system APIs or lightweight tools build on those APIs for system data collection. Additionally, based on the scope of the experiments, we adjust the data collection interval and the corresponding number of data points. We have been using sar, iostat, and dstat monitors for system data collections. In most of the reported experiments, we have been using one second granularity for the data snapshot interval.

When collecting application data such as thread pool and database pool utilization, we have been using lightweight API calls supported by the JVM. To cope with the scalability of our experiments, we have slightly modified the client emulators of the benchmark application to record bare minimum information that is sufficient for our data analysis.

3.10.5 Single-bottlenecks vs. Multi-bottlenecks

The content presented in this subsection has previously been introduced by Malkowski et al. [66] and provides the basis of the bottleneck terminology in this paper. The abstract definition of a *system bottleneck* (or bottleneck for short) can be derived from its literal meaning as the key limiting factor for achieving higher system throughput. Due to this intuition, similar formulations have often served as foundation for bottleneck analysis in computer systems. But despite their popularity, these formulations are based on assumptions that do not necessarily hold in practice. Because of queuing theory, the term bottleneck is often used synonymously for single-bottleneck. In a single-bottleneck cases, the saturated resource typically exhibits a near-linearly load-dependent average resource utilization that saturates at one hundred percent past a certain workload. However, the characteristics of bottlenecks may change significantly if there is more than one bottleneck resource in the system. This is the case for many real n-tier applications with heterogeneous workloads. Therefore, we explicitly distinguish between single-bottlenecks and the umbrella-term *multi-bottlenecks* [65,66].

Because system resources may be causally dependent in their usage patterns, multi-bottlenecks introduce the classification dimension of *resource usage dependence*. Additionally, greater care has to be taken in classifying bottleneck resources according to their *resource saturation frequency*. Resources may saturate for the entire observation period (i.e., fully saturated) or for certain parts of the period (i.e., partially saturated). In summary, the classification that forms the basis of the multi-bottleneck definition distinguishes between *simultaneous*, *concurrent* and *oscillatory* bottlenecks. In comparison to other bottlenecks,

resolving oscillatory bottlenecks may be very challenging. Multiple resources form a combined bottleneck, and it can only be resolved in union. Therefore, the addition of resources does not necessarily improve performance in complex n-tier systems. In fact, determining regions of complex multi-bottlenecks through modeling may be an intractable problem. Consequently, multi-bottlenecks require measurement-based experimental approaches that do not oversimplify system performance in their assumptions.

Classic bottleneck analysis employs average values to investigate resource saturation states. Such results are then used to support claims concerning service-level performance, throughput characteristics, and hardware demands. However, average numbers can mask the variations introduced by non-stationarity in workloads, which are an inherent characteristic of n-tier systems. We address this statistical threat through kernel density estimation to display the actual distribution of resource utilization values. Densities intuitively indicate the percentage of time each resource spent at a given utilization level. Peaks at high utilization imply that the resource was occasionally fully saturated.

CHAPTER IV

CHALLENGES AND OPPORTUNITIES IN CONSOLIDATION AT HIGH RESOURCE UTILIZATION: NON-MONOTONIC RESPONSE TIME VARIATIONS IN N-TIER APPLICATIONS

A central goal of cloud computing is high resource utilization through hardware sharing; however, utilization often remains modest in practice due to the challenges in predicting consolidated application performance accurately. We present a thorough experimental study of consolidated n-tier application performance at high utilization to address this issue through reproducible measurements. Our experimental method illustrates opportunities for increasing operational efficiency by making consolidated application performance more predictable in high utilization scenarios. The main focus of this paper are non-trivial dependencies between SLA-critical response time degradation effects and software configurations (i.e., readily available tuning knobs). Methodologically, we directly measure and analyze the resource utilizations, request rates, and performance of two consolidated n-tier application benchmark systems (RUBBoS) in an enterprise-level computer virtualization environment. We find that monotonically increasing the workload of an n-tier application system may unexpectedly spike the overall response time of another co-located system by 300 percent despite stable throughput. Based on these findings, we derive a software configuration best-practice to mitigate such non-monotonic response time variations by enabling higher request-processing concurrency (e.g., more threads) in all tiers. More generally, this experimental study increases our quantitative understanding of the challenges and opportunities in the widely used (but seldom supported, quantified, or even mentioned) hypothesis that applications consolidate with linear performance in cloud environments.

4.1 Introduction

Server consolidation (or simply *consolidation*) through hardware virtualization technology is a fundamental technique for achieving economical sharing of computing infrastructures as computing clouds. Consolidated servers run on the same physical node in dedicated Virtual Machines (VMs) to increase overall node utilization, which increases profit by reducing operational costs such as power consumption. As computing nodes grow in power (e.g., number of cores per node) with Moore’s law, there is an increasing amount of unused hardware per node. This trend of steadily decreasing utilization of physical nodes makes consolidation a technique of rapidly growing importance.

In contrast to the conceptual simplicity of consolidation in clouds, leveraging the full potential of this technology has presented significant challenges in practice. In fact, enterprise computing infrastructures continue to struggle with surprisingly low resource utilization—between 4 and 18 percent average utilization [33, 91]. While the exact reasons why the utilization levels of datacenters have not significantly improved in the last decade are the subject of controversial debates, there is ample anecdotal evidence from engineers and analysts that traditional applications make it difficult to optimize IT resource allocation [9].

Where a classic approach uses (ad hoc) system tuning to find and eliminate resource bottlenecks to resolve the discrepancy between concept and practice [46], we argue that the dynamic nature of clouds necessitates more methodical experimental approaches to predicting and analyzing the performance of consolidated applications. A major motivation for our argument is the existence of a phase transition in the performance and effectiveness of consolidation algorithms (e.g., [37]) in regions of higher resource utilization. While it is generally accepted that under low utilization consolidation wins by adding more applications, consolidation loses when too many applications are co-located, causing a loss of Quality of Service (QoS) and revenues. We believe this discontinuity in system properties, between winning and losing, to be a fundamental phenomenon, not easily addressed through performance debugging.

We address the challenge of predicting consolidated n-tier application performance at high resource utilization through a detailed experimental study. Our main focus are non-trivial response time degradation effects, which are particularly critical for QoS and Service Level Agreements (SLAs), when co-locating multiple n-tier applications on shared hardware. Methodologically, we directly measure and analyze the request rates, resource utilization, and performance of two instances of a representative n-tier application benchmark (RUBBoS) in a popular enterprise-level computer virtualization environment using various system analysis tools (e.g., SysViz [8]). More specifically, we investigate how readily available n-tier application configuration knobs (e.g., the maximum number of threads in web servers or the number of available DB connections in application servers) impact response time and throughput in high utilization consolidation scenarios.

Our main contribution is an in-depth experimental study of the challenges and opportunities in performance management when consolidating n-tier applications in cloud environments at high utilization. We illustrate and explain non-trivial response time degradation phenomena, particularly and importantly, with respect to the onset of regions of higher resource saturation. As an example, our measurements show that the response time in a particular n-tier consolidation scenario may unexpectedly and non-monotonically increase by 300 percent and then decrease to its initial level (see Figure 18(b)). This response time spike appears as the workload of the co-located (i.e., competing for resources) n-tier application system is increased monotonically and despite linear throughput.

Based on our experimental findings, we derive a practical configuration best-practice for consolidated n-tier applications. We show that this guideline is able to mitigate some of the previously illustrated performance degradation effect and therefore allows leveraging the high utilization in this scenario into efficient and SLA-conform throughput. More concretely, we show that when consolidated n-tier application systems are configured with additional software resources (e.g., larger thread and connection pools), the risk of non-monotonic response time degradation under high resource utilization can be significantly lowered. This

configuration practice augments the inherent performance isolation properties of modern clouds and may lead to more stable performance in traditional applications when deployed in co-located scenarios, even as resource utilization grows beyond regions that are traditionally considered to guarantee SLAs.

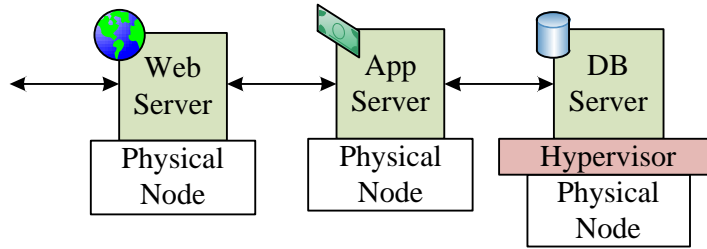
The remainder of this paper is structured as follows. In Section 4.2 we describe our experimental setup, detailing our n-tier application deployment and our testbed. Subsequently, Section 4.3 introduces our results on challenges and opportunities in consolidation of n-tier applications in clouds. Section 4.4 provides an overview of related work in this area. Finally, Section 4.5 concludes the paper with a brief result summary and impact considerations.

4.2 *Experimental Setup*

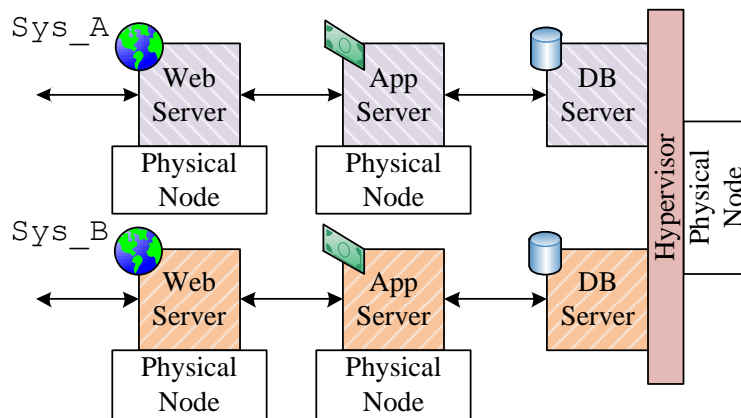
While consolidation as practice may be applied to any type of application, the focus of this paper are n-tier applications with LAMP (Linux, Apache, MySQL, and PHP) implementations. Typically, n-tier applications are organized as a pipeline of servers¹, starting from web servers (e.g., Apache), through application servers (e.g., Tomcat), and ending in database servers (e.g., MySQL). This organization, commonly referred to as n-tier architecture (e.g., 3-tier in Figure 17(a)), serves many important web-facing applications such as e-commerce, customer relationship management, and logistics. In our experiments, we use the popular n-tier application benchmark system RUBBoS [5]. Due to space constraints, we solely focus on results obtained with a browsing-only workload setting. Our default experiment ramp-up and run-times are 180 and 300 seconds, respectively.

At an abstract level, the deployment of n-tier applications in a cloud computing infrastructure can be modeled as a mapping between component servers and physical computing nodes. An application deployment is *dedicated* whenever the number of physical nodes is at least equal to the number of servers. Figures 17(a) exemplifies a dedicated n-tier deployment with one web server, one application servers, and one database server. In contrast, if the

¹In this paper *server* is used in the sense of computer programs serving client requests. Hardware is referred to as *physical computing node* or *node* for short.



(a) Dedicated deployment of a 3-tier application system with three software servers (i.e., web, application, and database) and three physical hardware nodes.



(b) Consolidated deployment of two 3-tier systems (Sys_A and Sys_B) with one server per tier and five physical hardware nodes in total. The DB servers are co-located in dedicated VMs on a single shared physical hardware node.

Figure 17: Example of a dedicated (a) and a consolidated (b) 3-tier application system deployment, presented as mappings of software servers to physical hardware nodes.

number of physical nodes is smaller than the number of servers, the deployment mapping is a *consolidation*, which requires at least two servers to be co-located on a single physical node (e.g., Figure 17(b)). In our experiments, we denote the first RUBBoS system as Sys_A and the second RUBBoS system as Sys_B, as illustrated in the figure. Unless otherwise stated, the default consolidation methodology in this paper is to affiliate (i.e., pin) both VMs to the same CPU core and limit both virtual CPUs to 1.36GHz (i.e., 60% of 2.27GHz) with a reservation of 0.00MHz and normal shares (i.e., both VMs have the same priority). Other important characteristics of our experimental testbed are summarized in Table 5.

Table 5: Summary of experimental setup (i.e., hardware, operating system, software, and virtualization environment).

CPU	Quad Xeon 2.27GHz * 2 CPU (HT)
Memory	12GB
HDD	SAS, 10,000RPM, 300GB * 2 (RAID1)
Network I/F	1Gbps * 2
OS	RHEL Server 5.3 (Tikanga), 32-bit
OS Kernel	2.6.18-128.el5PAE
Web Server	HTTPD-2.0.54
App Server	Apache-Tomcat-5.5.17
Connector	Tomcat-Connectors-1.2.28-src
DB Server	MySQL-5.0.51a-Linux-i686-glibc23
Java	JDK1.6.0_23
Monitoring Tools	SysViz [8] and custom VMware CPU monitor
Hypervisor	VMware ESXi v4.1.0
Guest OS	RHEL Server 5.3, 32-bit
Guest OS Kernel	2.6.18-8.el5

In this work we deeply analyze client request traces because a single web client request in an n-tier system may trigger multiple non-trivial interactions between different tiers. For this purpose we use SysViz—an n-tier system trace analysis tool developed at the Fujitsu Laboratories. SysViz reconstructs the entire trace of each transaction executed in the n-tier system, based on a comprehensive traffic message log, to calculate the intra-node delay of every request in any server in the system. More concretely, the SysViz tool is based on a passive 4-step network tracing process: (I) collect a complete IP packet log from the n-tier system; (II) translate the IP packet log to protocol messages (e.g., HTTP and AJP) exchanged between tiers; (III) extract identification data from each protocol message (e.g., URL of an Web request); (IV) reconstruct the trace of each transaction of the n-tier system using these identification data, pre-learned transaction-flow models, and a set of transaction matching algorithms [8].

In order to monitor the CPU utilization of each VM in our testbed environment sufficiently accurately and at sufficiently fine granularity (i.e., 0.1 seconds), we have developed a custom CPU monitoring tool. To do so, we have slightly modified the source code of our bare-metal hypervisor. While we have experimentally confirmed that our custom CPU monitor operates with very high accuracy and with negligible overhead, these empirical results are omitted here due to space constraints.

4.3 Experimental Consolidation Study

In this section we experimentally investigate the challenge of non-monotonic response time variations, which may unexpectedly appear in n-tier applications due to software configuration settings and CPU scheduling artifacts in high utilization consolidation scenarios. Based on our findings, we analyze opportunities in high resource utilization with software configuration settings that allow to mitigate the previously illustrated performance penalties. Consequently, our analysis strives to make high resource utilization scenarios more attractive in practice.

Table 6: The workload of Sys_A is increased monotonically at a constant rate of 100 concurrent users while the workload of Sys_B is kept constant at 2,300 concurrent users during all experiments. The deployment corresponds to Figure 17(b).

Experiment #	Workload [# users]		Total Runtime [s]
	Sys_A	Sys_B	
1	1,600	2,300	300
2	1,700	2,300	300
3	1,800	2,300	300
4	1,900	2,300	300
5	2,000	2,300	300
6	2,100	2,300	300
7	2,200	2,300	300
8	2,300	2,300	300
9	2,400	2,300	300

Software configuration, in the context of our work, refers to software settings that specify how many *software resources* such as DB connections in servlet programs in Tomcat or threads in Apache, Tomcat, and MySQL are allocated. Software resources are a natural extension of hardware resources (e.g., CPU and network) and inherently control how much request-processing concurrency is enabled in each tier. However, analogously to hardware resources, allocating the appropriate amount of software resources is a non-trivial challenge. In fact, previous research shows that an intricate balance between concurrency overhead and bottlenecks may necessitate the use of sophisticated configuration algorithms in dedicated deployment scenarios [102].

Methodologically, we consolidate two RUBBoS systems (i.e., Sys_A and Sys_B) in our experimental testbed (see Section 4.2) to investigate how increasing resource competition may affect the performance of n-tier systems. Concretely, we increase the workload of Sys_A from 1,600 to 2,400 concurrent users in steps of 100 users, and we keep Sys_B workload constant at 2,300 users, as shown in Table 6.

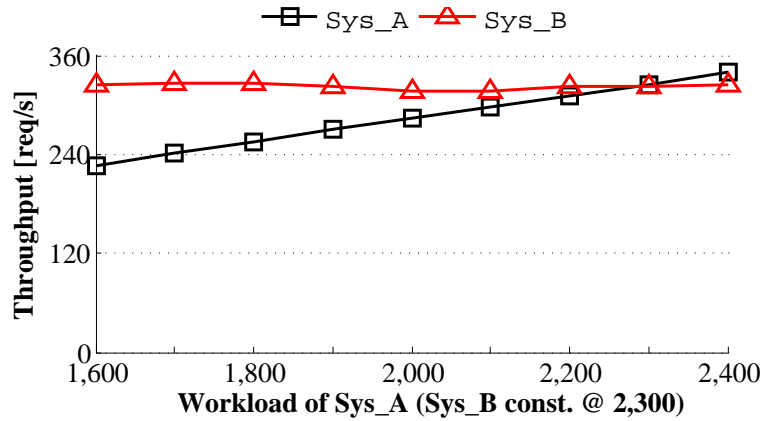
In order to study adverse software resource allocation effects, both systems are configured

with practical resource allocation settings that are derived from experimental results with dedicated deployments (e.g., Figure 24(c)). Sys_A is configured with a database connection pool size of 96 (12 DB connections per each of the 8 servlet programs in Tomcat). For the other software resources, we follow our previously published allocation algorithm and configure the system according to the RUBBoS-specific transaction-flow models to minimize concurrency overhead (e.g., Apache worker connection pool of 200 and 240 Tomcat threads) [102]. Sys_B is configured with a tuned DB connection pool size of 16 (2 DB connections per each of the 8 servlets in Tomcat), which is the best-performing software resource allocation in many dedicated deployment scenarios (e.g., Figure 24(c)). Please refer to Section 4.3.3 for further discussion of software resource allocation.

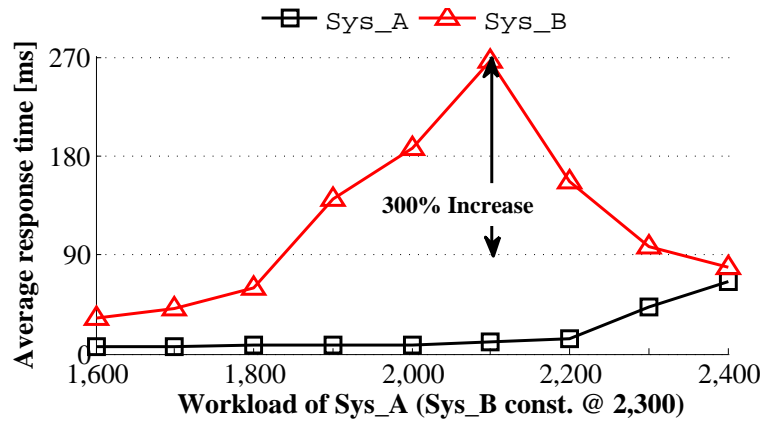
The rest of this section is organized in three parts. First, we introduce the challenging performance anomaly that is at the core of this paper in Section 4.3.1. In Section 4.3.2 we present and test hypotheses for the explanation of the performance anomaly. Finally, we discuss opportunities for high utilization consolidation of n-tier applications in clouds facilitated by means of appropriate software resource allocation in Section 4.3.3.

4.3.1 Non-monotonic Response Time in Consolidation

The performance anomaly, which is at the core of this paper, is depicted in Figure 18. The average throughput and response time of Sys_A and Sys_B are shown in Figures 18(a) and 18(b), respectively. Both throughput rates are largely linear and proportional to the system workloads. Meanwhile, the average response time graph of Sys_B reveals a significant peak (0.26s) at a Sys_A workload of 2,100 users. On first sight, this peak does not seem to be trivially explainable; in fact, this peak raises the question whether there are ways of mitigating such performance degradation to leverage the linear throughput in this high resource utilization scenario into profit. In the following, we illustrate why the response time performance of Sys_B deteriorates this significantly for a workload of 2,100 Sys_A users and then improves again (0.08s) as the workload of Sys_A increases to 2,400 users.



(a) Even under high total workload levels, the throughput of both systems remains proportional to their workload (i.e., linear). Sys_A throughput grows linearly, and Sys_B throughput is constant.



(b) Sys_B response time increases unexpectedly by 300% and then decreases to 30% of its peak value in contrast to Sys_A, which shows a normally expected monotonic response time increase.

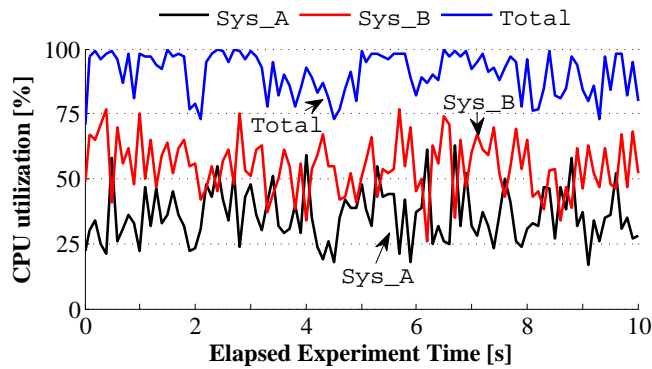
Figure 18: Throughput (a) and response time (b) of 3-tier systems Sys_A and Sys_B. The deployment of the two consolidated systems corresponds to Figure 17(b).

Furthermore, we discuss the implications of software configuration in consolidated n-tier applications in the context of this performance phenomenon.

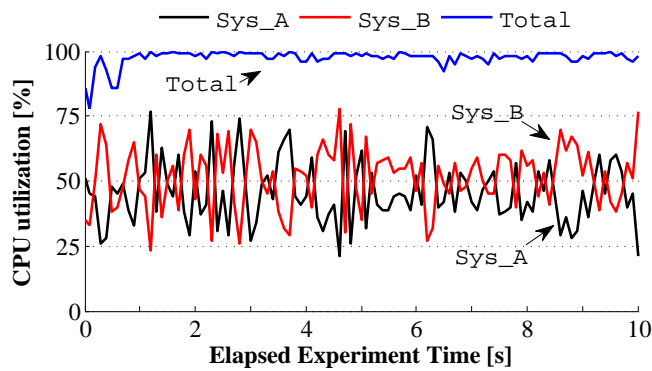
As a first step, we analyze the resource utilization in the shared DB node and the number of concurrent requests in each tier of Sys_B. We focus on the CPU utilizations because the main memory size of the VMs (4GB) is sufficient to eliminate disk I/O for the browsing-only RUBBoS workload, which is used in this set of experiments. We aim to analyze how the number of concurrent requests that queue up in the tiers of Sys_B relates to the utilization of the shared CPU core (i.e., the main physical bottleneck). Initially, we analyze this relationship qualitatively and zoom into 10 second time windows while aggregating the utilization metrics at 0.1s granularity. Figure 19 illustrates the VM CPU utilizations in the physical DB node as the workload of Sys_B is constant at 2,300 concurrent users, and the workload of Sys_A increases from 1,600 to 2,000 and then to 2,300 concurrent user. Figure 20 depicts the average number of concurrent requests (i.e., scheduled and waiting) in each tier of Sys_B (i.e., Web, App, and DB tiers).

As a first step, we analyze the resource utilization in the shared DB node and the number of concurrent requests in each tier of Sys_B. We focus on the CPU utilizations because the main memory size of the VMs (4GB) is sufficient to eliminate disk I/O for the browsing-only RUBBoS workload, which is used in this set of experiments. We aim to analyze how the number of concurrent requests that queue up in the tiers of Sys_B relates to the utilization of the shared CPU core (i.e., the main hardware bottleneck). Initially, we analyze this relationship qualitatively and zoom into 10 second trace windows while aggregating the utilization metrics at 0.1s granularity. Figure 19 illustrates the VM CPU utilizations in the physical DB node as the workload of Sys_B is constant at 2,300 concurrent users, and the workload of Sys_A increases from 1,600 to 2,000 and then to 2,300 concurrent user. Figure 20 depicts the average number of concurrent requests (i.e., scheduled and waiting) in each tier of Sys_B (i.e., Web, App, and DB tiers).

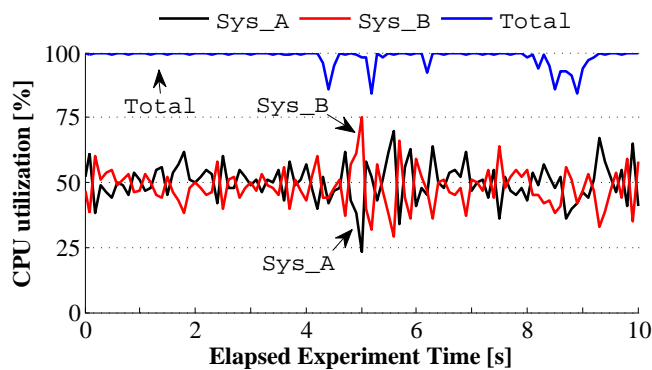
In Figure 19(a) Sys_A CPU utilization is significantly lower than Sys_B CPU utilization,



(a) Sys_A CPU utilization is significantly lower than Sys_B CPU utilization, which corresponds to the significantly lower workload of Sys_A.

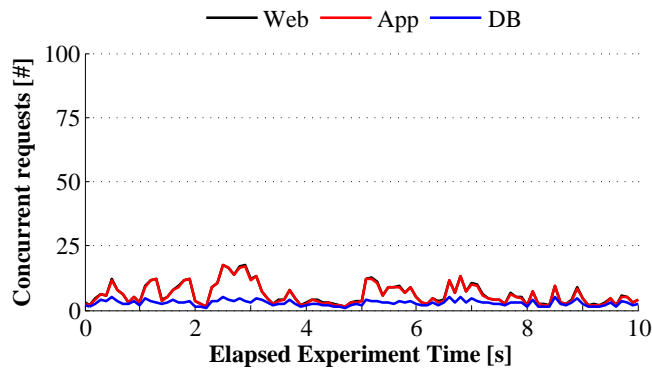


(b) Sys_A CPU utilization is only slightly lower than Sys_B, and the two graphs are mirror images, which suggests that one system waits for the other.

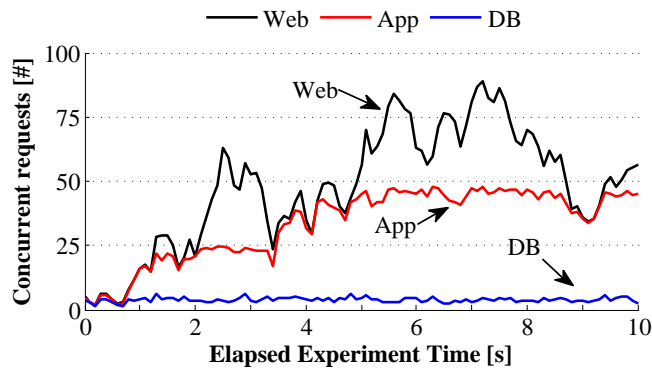


(c) The two utilizations are approximately equal on average, and the two graphs are mirror images, which suggests that one system waits for the other.

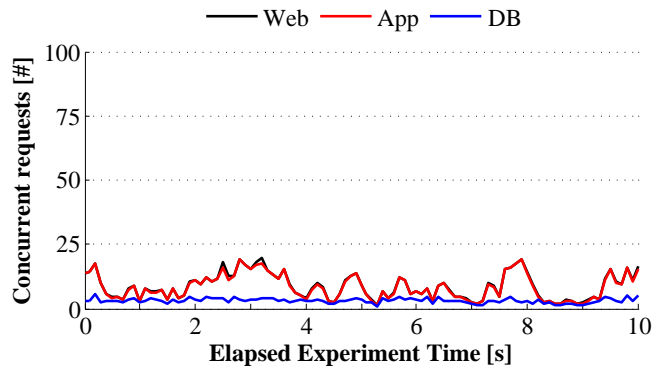
Figure 19: Trace snapshots (10 seconds) of the CPU utilizations in DB node of the backend consolidation scenario with constant Sys_B workload of 2,300 users and Sys_A workloads of 1,600 users (a), 2,000 users (b), and 2,300 users (c).



(a) The varying App queue length suggests a DB tier bottleneck with occasional DB connection pool (2 per servlet) saturation. There is no additional waiting time at the Web tier.



(b) Longer Web queues compared to App indicate an additional bottleneck in the App tier, suggesting that the thread pool in the App server is saturated with waiting requests.



(c) The disappearance of the additional waiting time in the Web tier suggests that the bottleneck in the App server has been resolved. The App queue length varies analogous to Figure 20(a).

Figure 20: The number of concurrent requests (processing and waiting) in Sys_B tiers (same experiment runs as in Figure 19).

which corresponds to the significantly lower workload of Sys_A (1,600 users) compared to Sys_B (2,300 users). Consequently, the occasionally varying App queue length in Figure 20(a) suggests a DB tier bottleneck with occasional DB connection pool (2 per servlet) saturation. In this scenario each Web request corresponds to one App request, which confirms that there is no additional waiting time for Sys_B requests at the Web tier. The result is the modest overall response time shown in Figure 18(b).

In Figure 19(b) the DB CPU utilization of Sys_A has increased significantly compared to Figure 19(a). On average, Sys_A DB CPU utilization is only slightly lower than Sys_B DB CPU utilization; moreover, the graphs are mirror images (i.e., negatively correlated), which suggests that one system waits for the other. In other words, Sys_B CPU utilization drops when Sys_A peaks and Sys_B CPU utilization peaks when Sys_A drops. Both utilizations fluctuate frequently and significantly around their mean utilization of 50%, and Figure 20(b) shows longer Web queues compared to App, which indicates an additional bottleneck in the App tier. This suggests that the thread pool in the App server is saturated with waiting requests, which is ultimately reflected by the significant overall response time increase in Figure 18(b).

Finally, the mean values of the two DB CPU utilizations are approximately equal in Figure 19(c), and the two graphs are mirror images; however, the utilizations fluctuate significantly less compared to Figure 19(b). Correspondingly, Figure 20(c) shows the disappearance of the additional waiting time in the Web tier. This suggests that the bottleneck in the App server has been resolved. Moreover, the App queue length varies analogous to Figure 20(a), which is reflected by the significant overall response time decrease shown in Figure 18(b).

Finally, the mean values of the two DB CPU utilizations are approximately equal in Figure 19(c), and the two graphs are mirror images; however, the utilizations fluctuate significantly less compared to Figure 19(b). Correspondingly, Figure 20(c) shows the disappearance of the additional waiting time in the Web tier. This suggests that the bottleneck

in the App server has been resolved. Moreover, the App queue length varies analogous to Figure 20(a), which is reflected by the significant overall response time decrease shown in Figure 18(b).

4.3.2 Hypotheses and Correlation Analysis

Our experimental data supports the hypothesis that the causal chain behind this performance anomaly is two-dimensional. In the first dimension, the overall response time of both systems is highly dependent on the priority that the DB node VMs are assigned by the hypervisor's CPU scheduler. In the second dimension, the relatively low software resource allocation in Sys_B, which is tuned for dedicated scenarios, amplifies waiting time increases in the DB tier of Sys_B and results in a disproportionately larger degradation of overall Sys_B response time. Both dimensions in combination explain the high peak of Sys_B response time in Figure 18(b).

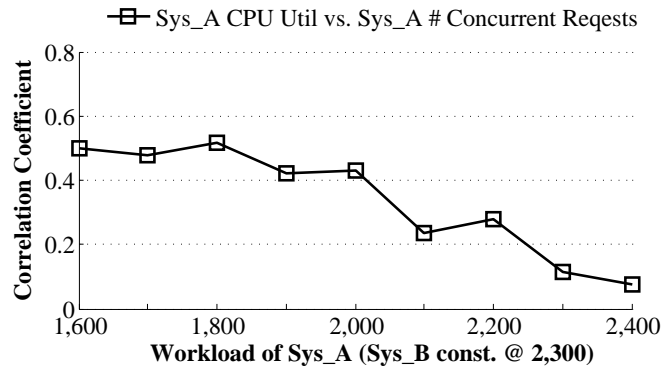
The first dimension is a result of the average-utilization-based CPU scheduling policy in the DB node hypervisor, and there are two different regions of system properties that are distinguishable in our experiments. For low workloads of Sys_A, the CPU scheduler in the DB node hypervisor preferentially allocates CPU to the Sys_A VM, based on its lower mean CPU utilization. As long as Sys_A workload is significantly lower than Sys_B workload, Sys_B DB CPU utilization drops whenever Sys_A DB CPU requirement peaks (Figure 19(b)). These fluctuations of Sys_A DB CPU requirement occur naturally due to the stochastic nature of the client request process. However, once Sys_A workload is increased significantly, the mean CPU utilizations become equal (Figure 19(c)), and the CPU scheduling priority also becomes equal. Consequently, the variation in Sys_B DB CPU utilization reduces significantly, and the long waiting queues in Sys_B disappear (compare Figures 20(b) and 20(c)).

The second dimension is a result of Sys_B not being able to process the queue of waiting CPU-intensive request of the type ViewStory sufficiently fast. This limitation is due to the

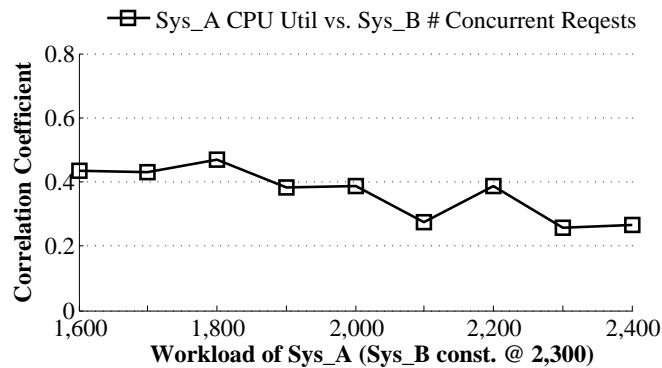
low software resource allocation in Sys_B. After peaks of Sys_A DB CPU utilization, Sys_B requires CPU and is scheduled by the hypervisor; however, the number of DB connections for long-processing ViewStory requests is too low to swiftly reduce the queue length of ViewStory requests in the App server (concurrency is limited to two connections per request type). Moreover, long-processing requests have to wait even longer at the DB CPU due to the large load of short-processing requests that are supplied to the database because they have a sufficient amount of available DB connections. Finally, when all threads in the App server are consumed by waiting ViewStory requests, all request types that arrive at the Web tier are put in a waiting queue, irrespective of type and available DB connections. This explains the long queues in Figures 20(b) and causes the disproportionately large response time increase in Figure 18(b).

To provide quantitative evidence for our hypotheses, we analyze the linear relationships between the main metrics in the preceding explanation (i.e., CPU utilization, number of concurrent jobs, and response time). Figures 21 and 22 show the corresponding correlation coefficient graphs for Sys_A and Sys_B, respectively. The coefficients are calculated based on traces at 0.1s granularity collected in ten independent experiment runs. More specifically, Figures 21(a) and 22(a) illustrate the correlation between the DB CPU utilization and the number of concurrent DB requests; Figures 21(b) and 22(b) show the correlation between the DB CPU utilization and the DB response time of the co-located VM; and Figures 21(c) and 22(c) show the correlation between the DB CPU utilization and the DB response time of the co-located VMs for Sys_A and Sys_B, respectively.

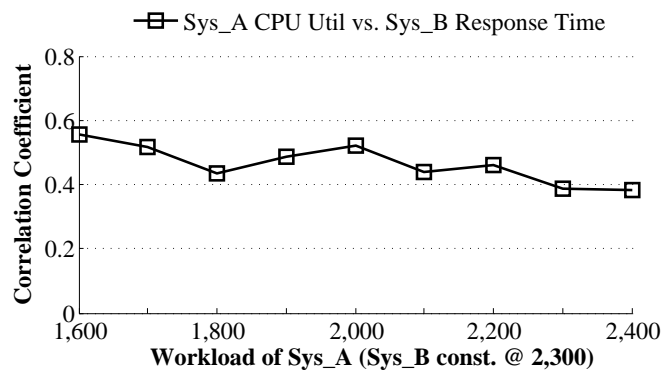
In Figure 21(a) the positive correlation suggests that Sys_A is scheduled with high priority, and the decreasing trend suggests a lower scheduling priority at higher workloads. This corresponds to the negative correlation in Figure 22(a), which indicates that Sys_B is scheduled with low priority. Sys_B scheduling priority reaches a minimum at Sys_A workload of 2,000 users. For higher Sys_A workloads the scheduling priority of Sys_B grows again. In Figure 21(b) the constantly high positive correlation suggests that Sys_B



(a) Positive correlation suggests that Sys_A is scheduled with high priority. The decreasing trend suggests a lower priority at higher workloads.

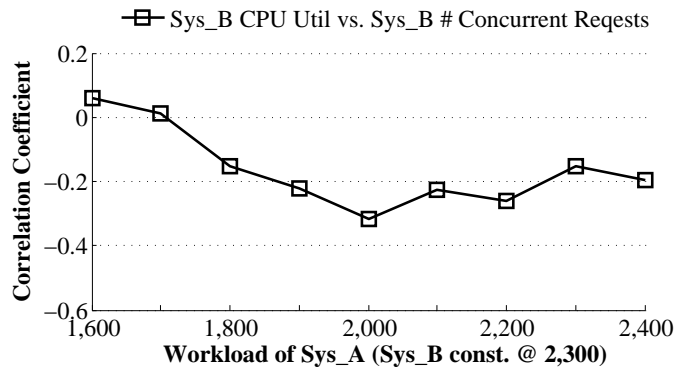


(b) Constantly high positive correlation suggests that Sys_B requests queue up when Sys_A is scheduled on the CPU for all analyzed workloads.

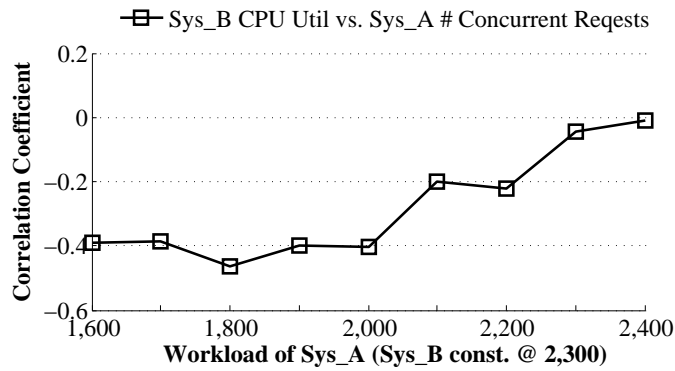


(c) Constantly high positive correlation suggests that Sys_B response time grows when Sys_A is scheduled on the CPU for all analyzed workloads.

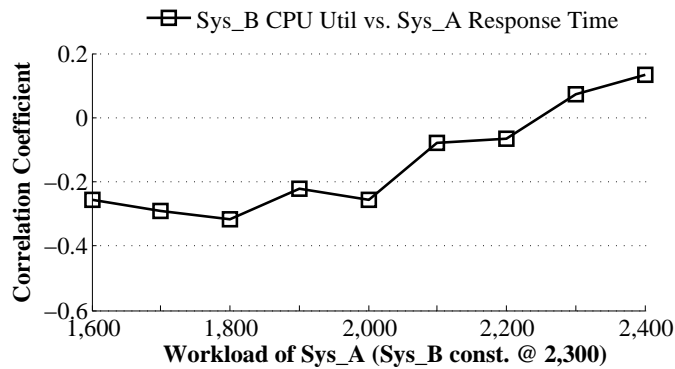
Figure 21: Correlation analysis based hypothesis testing of Sys_A properties. Pearson's correlation coefficients are calculated based on CPU utilization, number of concurrent jobs, and response time traces collected in the DB node at 0.1s granularity.



(a) Negative correlation suggests that Sys_B is scheduled with low priority. The priority reaches a minimum at Sys_A workload of 2,000 users, past which the scheduling priority grows again.



(b) Negative correlation suggests that Sys_B CPU utilization drops when the number of Sys_A requests increases. Past Sys_A workload of 2,000 users, this dependence diminishes rapidly.



(c) Negative correlation suggests that Sys_B CPU utilization drops when the response time of Sys_A grows due to request queues. Past Sys_A workload of 2,000 users, this dependence diminishes.

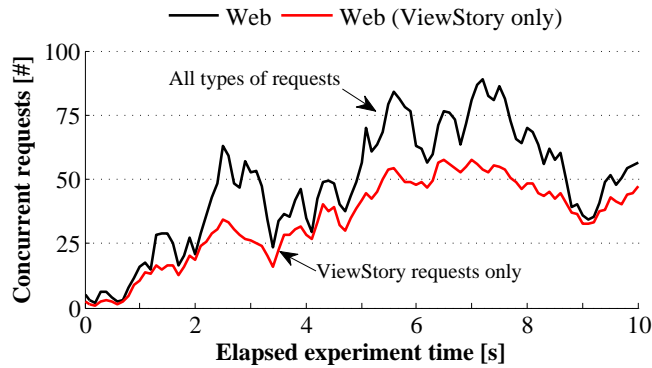
Figure 22: Correlation analysis based hypothesis testing of Sys_B properties. Pearson's correlation coefficients are calculated based on CPU utilization, number of concurrent jobs, and response time traces collected in the DB node at 0.1s granularity.

requests queue up when Sys_A is scheduled on the CPU for all analyzed workloads. Moreover, the constantly high positive correlation in Figure 21(c) indicates that Sys_B response time grows when Sys_A is scheduled on the CPU for all analyzed workloads. In contrast, the negative correlation in Figure 22(b) suggests that Sys_B CPU utilization drops when the number of Sys_A requests increases. For Sys_A workloads higher than 2,000 users, this dependence diminishes rapidly. Finally, the negative correlation in Figure 22(c) suggests that Sys_B CPU utilization drops when the response time of Sys_A grows due to request queues. For Sys_A workloads higher than 2,000 users, this dependence diminishes.

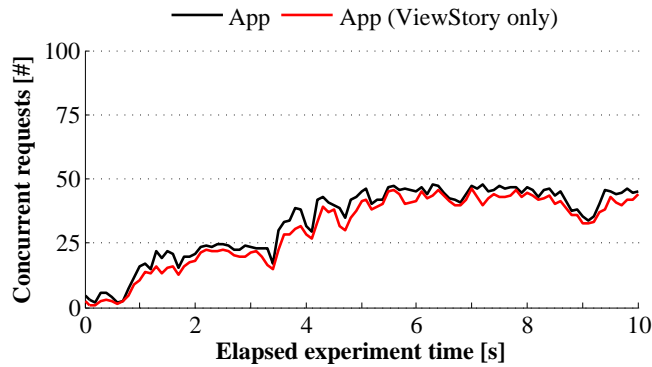
To provide quantitative evidence for our assertions about the role of long-processing requests, we illustrate the prominence of ViewStory request in all tiers of Sys_B (Figure 23). The traces in this figure correspond to the scenario shown in Figure 20(b). There is a significant amount of queued requests in the Web tier of Sys_B that are not of the type ViewStory (Figure 23(a)). This suggests that the overall response time of Sys_B increases significantly due to a bottleneck in the App tier of Sys_B. The overall response time is amplified because all request types queue up in the Web server and suffer from the long-processing ViewStory waiting time—even though they might be of a type that is normally short-processing. In contrast, most queued requests in the App tier are of type ViewStory (Figure 23(b)). This suggests that most threads in the App tier are blocked on queued ViewStory requests, which explains the request backlog in the Web tier. Finally, the number of DB connections in the App server (i.e., two per servlet) determines the maximum DB request concurrency; thus, the number of requests in Sys_B DB tier remains stable (Figure 23(c)).

4.3.3 Software Resource Allocation in Consolidation

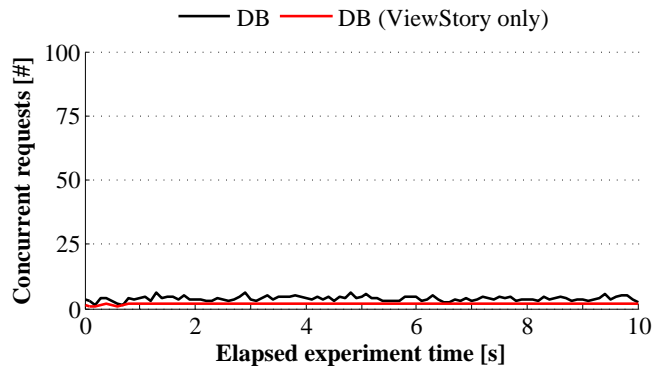
In the following we compare system response time in various software resource allocation scenarios in both consolidated and dedicated deployments. While Figures 24(a) and 24(b) show the response time of Sys_A and Sys_B in a consolidated deployment (see Figure 17(b)),



(a) All types of Sys_B requests are queued, which suggests that the overall response time of Sys_B increases significantly due to a bottleneck in the App tier of Sys_B.



(b) Predominantly, ViewStory requests are queued suggesting that most threads in the App tier are blocked with queued ViewStory requests, which causes the request backlog in the Web tier.



(c) The number of DB connections in the App server (i.e., 16) determines the maximum DB request concurrency; thus, the number of requests in the Sys_B DB tier remains stable.

Figure 23: The number of concurrent ViewStory requests in Sys_B tiers (same experiment runs as in Figures 19(b) and 20(b)).

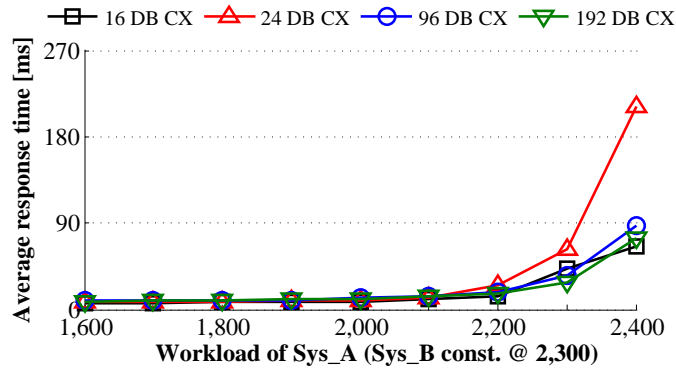
Figure 24(c) shows the response time for a dedicated deployment (see Figure 17(a)).

Very little variation is apparent for the different software configuration settings in Figure 24(a) Sys_A, which suggests that a concurrency setting that is at least equal to the co-located system (i.e., 16 DB connections in Sys_B) is dominant for consolidated scenarios. On the other side, Figure 24(b) Sys_B shows very high variation for different software configurations, which suggests that a concurrency setting that is lower than the co-located system (i.e., 96 DB connections in Sys_A) is inferior for consolidated scenarios. In contrast, Figure 24(c) shows an ordering of performance graphs according to increasing software configuration concurrency. Concretely, this suggests that in dedicated deployments of n-tier applications, lower concurrency may be favorable. This finding corroborates our previous results that showed that in dedicated environments lower software resource allocations may significantly increase overall system performance by taking advantage of data locality effects (e.g., in caches) and eliminating context switching penalties (e.g., at CPUs) [102].

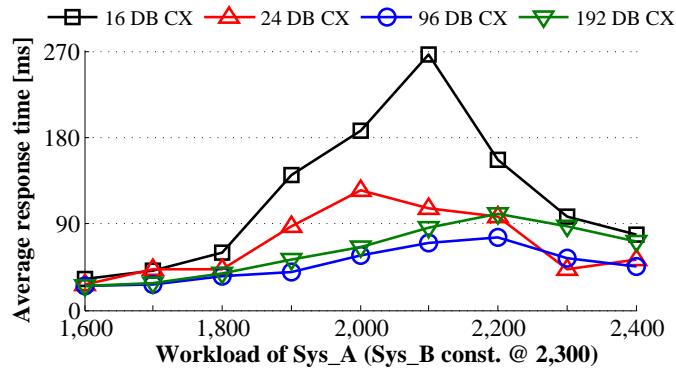
More generally, our results suggest that allocating additional software-resources in n-tier applications when deploying them in clouds may allow for higher resource utilization when sharing, without significant performance loss. Especially when compared to classic datacenter scenarios, software resource allocations should be increased to mitigate the impact of the second dimension response time amplification described in Section 4.3.2. Our data suggests that this practice is able to mitigate some of the negative implications of sharing at high utilization, making sharing at high utilization a more viable option. In other words, additional software resources help to increase performance stability in regions where cloud performance isolation mechanisms have less desirable performance properties.

4.4 Related Work

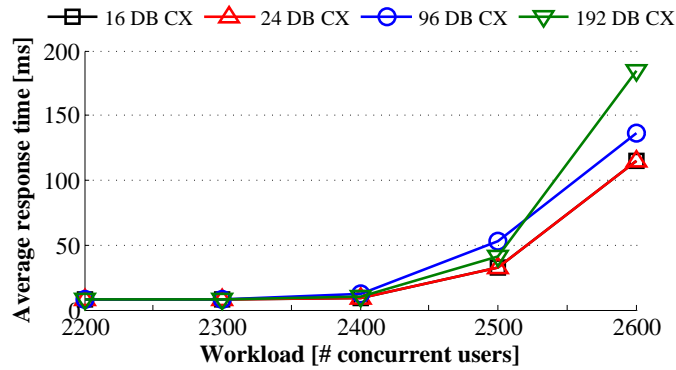
Many related research papers on application consolidation in cloud environments (e.g., [38, 69]) assume linear consolidation performance and apply optimization techniques to improve the location of various tasks. For example, a recent approach applies linear programming



(a) Sys_A shows very little variation for different software configurations suggesting that a concurrency setting that is at least equal to the co-located system (i.e., 16 DB CX in Sys_B) is dominant.



(b) Sys_B shows very high variation for different software configurations suggesting that a concurrency setting that is lower than the co-located system (i.e., 96 DB CX in Sys_A) is inferior.



(c) The ordering of the performance graphs according to increasing software configuration concurrency suggests that dedicated deployment of n-tier applications favors lower concurrency settings.

Figure 24: Comparison of three-tier system performance for various software resource allocations in both the consolidated (a), (b) and the dedicated (c) deployment scenarios show in Figure 17.

to improve consolidated application performance through dynamic VM migration [37]. Similarly, most papers that model and solve consolidation as a bin-packing optimization problem assume linear consolidation. Clearly, our experimental study of consolidation performance does not invalidate these good results, but our work helps to delimit the applicability of such results that assume linear consolidation performance.

A related line of research that also assumes linear consolidation uses real work application traces (e.g, [110]) or trace-driven simulation [108]. They typically use production application execution traces (instead of a benchmark application execution) to detect the major application resource requirements and then carefully consolidate the applications that have complementary resource requirements. It is our conjecture that trace-driven analyses that take into account deep knowledge of application resource requirements are more likely to find consolidation solutions that satisfy linear performance. This is due to their aim of reducing resource competition among applications, an effort that is closely aligned to the reduction of consolidation interference.

Most of the papers that assume linear consolidation performance use simulation or calculations. A new development is a recent paper on database server consolidation [33]. They use a non-linear optimization algorithm to find the consolidation strategy, and then evaluated their optimization algorithm through measurements of benchmarks and real database usage traces. Their measurements found a very close match between their optimized consolidation solution and the measured results, even for relatively high CPU utilization levels (30% average and the 95% percentile at near 60%). To the best of our knowledge, this work is the first experimental paper that claims low performance interference in their measurements of production clouds at realistic CPU levels, which represents a significant evolution from other measurement-based papers on performance interference. Consequently, this paper corroborates our research that aims at providing an extensive evaluation of consolidation performance in diverse scenarios.

There are several other related areas, but due to space constraints we can only enumerate a

sample of works. The first area is system management (proceedings of DSOM, IM, NOMS, LISA, and ICAC). Well known monitoring tools include: IBM Tivoli, HP OpenView, Stardust [96], Astrolabe [100], and SDIMS [107]. The second area is cloud benchmarking. In addition to RUBBoS [5], we are currently analyzing other cloud-benchmarks such as CloudStone [92], MRBench [52], and RUBiS [6].

4.5 Conclusion

This paper provides the first in-depth experimental study of consolidated n-tier application performance under high utilization in clouds. The main impact of a better understanding of the challenges and opportunities in performance management of applications in clouds is a more efficient utilization of cloud resources, which leads to higher profits. Currently, decision-makers hesitate to deploy mission-critical applications in clouds due to unpredictable performance under consolidation and therefore the true potential of clouds is often not fully leveraged. Our measurement-based analysis is a first step towards providing an accurate prediction of performance of consolidated benchmark applications in various scenarios, which will aid in increasing resource utilization. Consequently, cloud users, service providers, and researchers will eventually be able to run consolidated applications at higher utilizations than previously considered feasible with high confidence. This is a transformational research effort both for the systematic experimental evaluation of the hypothesis that applications consolidate with linear performance in clouds and for the quantitative understanding of the fundamental performance phenomenon of discontinuous properties in shared computer systems.

CHAPTER V

BOTTLENECK DETECTION USING STATISTICAL INTERVENTION ANALYSIS

The complexity of today's large-scale enterprise applications demands system administrators to monitor enormous amounts of metrics and to reconfigure their hardware as well as software at run-time without thorough understanding of monitoring results. The Elba project [7] is designed to achieve an automated iterative staging to mitigate the risk of violating Service Level Objectives (SLOs). As part of Elba, we undertake performance characterization of systems to detect bottlenecks in their configurations. This work introduces our concrete bottleneck detection approach based on statistical intervention analysis. We then show its robustness and accuracy in various configurations scenarios. We utilize a well known benchmark application, Rice University Bidding System (RUBiS), to evaluate the classifier with respect to successful identification of different bottlenecks.

5.1 Introduction

Pre-production configuration testing of complex n-tier enterprise application deployment, or staging, can be as demanding and complex as the production system itself. System analysts and administrators monitor and analyze a large number of application-specific metrics such as the number of active threads and the number of EJB entity bean instances, along with system-level metrics like CPU usage and disk I/O rate. Any of these resources may cause the system to violate performance service level objectives (SLO), usually specified as Service Level Agreement (SLA). Significantly lower cost and results at higher confidence levels may be produced by automated staging. The latter is an iterative process in the Elba project [7] whereby an application configuration is gradually refined.

The main contribution of this paper is an automated bottleneck detection scheme based on a statistical intervention model. This approach is distinct from our previous work that used machine learning [77]. We introduce a deterministic algorithm, which has proven to be very effective in the Elba environment. The process automatically examines and analyzes the entire metric data derived from the staging experiment trials. A limited set of interesting metrics is identified very fast without the need of an extensive training or configuration phase. This set is then ordered according to the degree of correlation with the high-level system performance. We also show that we are able to accurately determine potential bottlenecks in different scenarios. Moreover, the resulting output is easily interpretable due to the intuitive model structure.

The remainder of this paper is organized as follows. Section 5.2 describes our approach to bottleneck detection using intervention analysis. Section 5.3 presents the evaluation environment and our results utilizing RUBiS. Section 5.4 discusses related work, followed by the conclusion in Section 5.5.

5.2 Intervention Analysis

An effective staging phase assures system administrators that a hardware/software configuration is capable of handling workloads to be seen during production. Starting at an initial configuration, this phase augments resources allowing the configuration to better satisfy the SLOs. So far our bottleneck detection approaches consisted of a multi-step analysis. If a SLA was not met (SLO-satisfaction drops significantly) in a certain scenario, a three-step detection process began: staging the system with varying workloads and collecting performance data from system-level and application-specific metrics, training a machine learning classifier with the data, and finally querying the trained machine learning classifier to identify potential bottlenecks. Please refer to [77] for more details. While our three-step methodology proved to be successful, it mainly relies on machine learning algorithms to execute the final performance modeling and classification. This implies two typical shortcomings that

lie in the nature of the modeling scheme. Firstly, the machine learning classifiers require a training phase. This can be cost-intensive since certain accuracy and robustness levels might be defined a priori. Secondly machine learning classifiers produce a model that is not necessarily interpretable in a trivial manner. We discussed suitable interpretations in [77]. Nevertheless, this led to a residual degree of uncertainty in the interpretation of the analysis results.

In this article we propose a novel approach based on statistical techniques, which results in an improvement of our bottleneck detection process in a consistent manner. We introduce an intuitive statistical model, which eliminated the need of machine learning on the one hand. And on the other, we observe that our approach achieves the same high accuracy level at a lower cost (fewer staging trials). Therefore we greatly increase the efficiency of the detection process and enhance the clarity of the final results at the same time.

5.2.1 Assumptions

The following assumptions form the basis of our automated bottleneck methodology. They emphasize the general issues that need to be addressed by any satisfactory detection method and are reflected in previous Elba efforts.

- A single experiment trial is not sufficient to record a conclusive metric vector and thus several trials of varying workloads are required.
- Non-obvious interactions between resources make observation based bottleneck detection a hard problem. Nontrivial correlations have to be examined and the detection method needs to be able to produce a probabilistic result ranking.
- The number of recorded monitoring metrics is very high. It is critical to devise an approach that is able to sort through copious metric data automatically.
- The nature and appearance of metrics can vary significantly and they are typically categorized as either system-level or application-specific.

- High utilization of a resource implies high demand from an application while it may not necessarily be indicative of a bottleneck. A detection mechanism has to be capable of distinguishing bottlenecking behavior in terms of resource saturation.
- Especially trend changes in metric graphs are of high importance. In fact we found in our previous work that it was highly effective to examine first derivative approximations instead of the actually recorded values.

5.2.2 The Detection Model

These assumptions together with observations from empirical data analysis suggest a simple performance model. We formulate the latter in terms of statistical intervention analysis, which allows us to formalize the characteristic bottleneck behavior of the system accurately.

First we need to define an exogenous crossover point ($c \in W$). This specific number of concurrent user sessions can be seen as an intervention point that divides our workload span (W) into two disjunctive intervals.

$$I := \{w \in W \mid w < c\} \tag{1}$$

$$I' := \{w \in W \mid w \geq c\} \tag{2}$$

In this notation, I represents the set of workloads that result in high levels of SLO-satisfaction of the system, whereas the satisfaction levels drop significantly when exposed to workloads in I' (intervention effect).

For our purposes, we also need to adapt the standard transfer functional model formulation [23]. For any workload $w \in W$ an impact assessment model for the first difference of any metric value Y_w can be formulated in terms of Equation (3). Note that we use the first difference as approximation of the first derivative consistently with our previous findings [77].

$$\Delta Y_w := f(I_w) + N_w + \mu \quad (3)$$

$$I_w := \begin{cases} 1 & \text{if } w \geq c \\ 0 & \text{else} \end{cases} \quad (4)$$

In this formulation N_w is the noise component, and μ denotes the constant term. The effect of the intervention variable I_w on the metric trend is defined as $f(I_w)$. Following the standard notation, I_w is defined as an indicator function (Equation (4)). We can now sub-tract the noise component from both sides of Equation (1). Since we only have to deal with abrupt and permanent intervention effects we can assume linearity in the metric values. Based on this linearity assumption we introduce δ as the constant term of the intervention effect, which yields the following formulation.

$$\Delta Y_w - N_w = \delta I_w + \mu \quad (5)$$

In order to characterize the final model in a convenient manner, we define μ' in Equation (6) which leads to the final model formulation in Equation (7).

$$\mu' := \mu + \delta \quad (6)$$

$$\Delta \tilde{Y}_w := \Delta Y_w - N_w = \begin{cases} \mu & \text{if } w < c \\ \mu' & \text{else} \end{cases} \quad (7)$$

This notation emphasizes the importance of the potential change in the trend of the metric value Y_w as the system progresses from I to I' with increasing workload. Moreover, it allows us to establish causality between the model parameters of the low level metric and the high level system performance in an intuitive manner.

5.2.3 Determining an Intervention Point

Since the crossover point c between I and I' needs to be defined a priori, we define an iterative algorithm for our automated analysis scheme. The main idea is to assess the workload when the SLO-satisfaction S_w loses its stability and starts to deteriorate significantly upon further workload increase (i.e., we assume Property (8) and (9)). Although the model formulation requires an exact transition point, it is sufficient for our method to approximate c in a qualitative manner (refer to Table 10).

$$\forall i \in I : S_i \approx \text{const} \quad (8)$$

$$\forall i \in I' : S_i \ll \frac{1}{|I|} \sum_{j \in I} S_j \quad (9)$$

We start at the lowest workload in our dataset and iteratively increase the value by the smallest possible step. In every iteration we calculate a simple heuristic approximation of the ninety-five percent confidence interval of the SLO satisfaction values seen so far. We consider n_0 values which resulted from a workload smaller or equal to $w_0 \in W$ (the work-load currently examined).

$$90\% \leq \frac{1}{n_0} \sum_{0 \leq i \leq w_0} S_i - \frac{1.96}{\sqrt{n_0 - 1}} \sqrt{\sum_{0 \leq i \leq w_0} \left(S_i - \frac{1}{n_0} \sum_{0 \leq j \leq w_0} S_j \right)^2} \quad (10)$$

We continue to the next iteration as long as the lower bound of the confidence interval is not below ninety percent (Equation (10)). Thus we characterize the satisfaction level for the first interval in a binary fashion as suggested by our observations. Once the lower bound of the confidence interval drops below ninety percent we exit the algorithm. The exit point $w^* \in W$ is a heuristic approximation of the crossover point c . We can assume that the SLO-satisfaction has deteriorated significantly from its stable level for all workloads greater or equal to w^* , which yields the following formulation:

$$\hat{I} := \{w \in W \mid w < w^*\} \quad (11)$$

$$\hat{I}' := \{w \in W \mid w \geq w^*\} \quad (12)$$

5.2.4 Metric Selection Scheme

We can now turn to the process of selecting a set of potential bottleneck metrics and discarding all metrics that do not indicate a high resource saturation level. Given a known intervention (SLO begins to deteriorate) we identify all metrics that show evidence of a corresponding plateau (i.e., significant and permanent shift in average value) and a variability change in their first derivative (further evidence for a saturated resource). To identify the candidate set we perform a basic hypothesis-testing scheme adapted from [104]. We define a rule-based analysis process for testing the null Hypothesis (13) of constant mean μ and variance σ between the two intervals.

$$H_0 : \hat{\mu} \approx \hat{\mu}' \wedge \hat{\sigma} \approx \hat{\sigma}' \quad (13)$$

Empirical testing revealed that we have to account for the high variability of the metric data as well as adjust the analysis to specifically detect abrupt plateau shifts. Thus we deviate from the traditional intervention analysis methodology and devise a different testing scheme. We calculate representative quantiles for each interval and metric. The latter characterize the filtered behavior of the data in a more stable manner. We proceed to apply two selection rules in order to limit the group of candidate bottleneck metrics.

$$q_{0.5} > q'_{0.5} \wedge |q_{0.2} - q_{0.8}| > |q'_{0.1} - q'_{0.9}| \quad (14)$$

Rule (14) accounts for all limited metrics that will saturate at a level of hundred percent. We choose all metrics where the median has decreased as well as where the distance between ten- and ninety-quantile in the second interval is smaller than the distance between twenty-

and eighty-quantile in the first interval. If this rule is satisfied we have significant evidence to reject the null hypothesis and assign the metric to a set of potential bottlenecks.

$$q_{0.9} < q'_{0.1} \wedge q_{0.9} < q'_{0.5} \wedge q_{0.9} < q'_{0.9} \quad (15)$$

Rule (15) accounts for all metrics that are not limited and show an exponential behavior when the resource saturates. We select all metrics, where all three quantiles of the second interval have increased above the ninety quantile of the first one. Again we reject the H_0 if the rule applies. In this manner we have eliminated all metrics that do not show strong indications of bottlenecking behavior near the intervention point and narrowed our attention to potentially interesting resources. Note that the complete empirical derivation of the two decision rules is omitted due to space restrictions. Nevertheless it is based on standard statistical methods and our analysis experience.

5.2.5 Impact Assessment

Once we have identified the set of candidate bottlenecks we can perform a ranking to describe the magnitude of the change. The magnitude reveals the correlation with the intervention and specifically accounts for the exact time when the change in the metric occurred. Hence we design a normalizing ranking function R by calculating the quotient of the absolute mean values of the two intervals.

$$R := \left| \frac{\hat{\mu}}{\hat{\mu}'} \right| \quad (16)$$

This mechanism has two implications. Firstly, we assess how well the crossover point was chosen for each particular metric (temporal ranking). If the split is not exact, the resulting quotient will have a value closer to one. Furthermore, we have to rank how large the relative shift in plateau levels is for each particular metric. We expect bottlenecked metrics that were chosen with Rule (14) (limited metric) to display a very high-ranking value potentially approaching infinity. The slope of the metric values drops from a linear

increase to a stable value near zero. Metrics chosen by Rule (15) (unlimited metrics) will show a very low ranking value that is close to zero on the other hand. This means that a moderate positive slope changes to a very strong (exponential) growth. In the following evaluation we will subdivide the candidate set into set one and two. This will simplify the analysis for limited and unlimited metrics, respectively.

5.3 Experimental Evaluation

Rice University Bidding System, is a multi-tiered e-commerce application with 26 interaction types, such as browsing, bidding, buying, or selling items; registering users; and writing or reading comments. RUBiS provides two workload transition matrices describing two different user behaviors: a browsing transition consisting of read-only interactions and a bidding transition, including 15% write interactions. In our experiments the write ratio is extended adding additional variability as explained in [81]. We utilize the bidding transition as well as neighboring write ratios of 10% and 20% in our evaluation since these transitions are better representatives of an auction site workload [28] and provide a more accurate picture [81]. Our system reuses and extends a recent version of RUBiS from ObjectWeb [6]. Generally, experiments show that RUBiS is application-server tier intensive. In other words, it is characteristically constrained by performance in the EJB container tier as introduced in [28].

To execute the staging phase with RUBiS, we employ Apache 2.0.54 as an HTTP server, MySQL max-3.23.58 as a database server with type 4 Connector/J 3.0.11 as a JDBC driver, and JOnAS 4.4.6 / Tomcat 5.5.12 package as an EJB-Web container. Apache HTTP server is equipped with mod_jk so that it can be used as a front-end server to one or several Tomcat engines, and it can forward servlet requests to multiple Tomcat instances simultaneously via AJP 1.2 protocols. We increase the number of the maximum processes of Apache to avoid connection refusals from the server when numerous clients simultaneously request services. We also set the automated increment option on every primary key of the RUBiS

databases to prevent duplication errors when clients simultaneously attempt to insert data into a table with the same key. Finally, we adjust JOnAS to have an adequate heap memory size for preventing out-of-memory exceptions during staging.

For gathering system-level metrics, we wrote a shell script to execute Linux utilities, `sar` and `ps`, with monitoring parameters such as staging duration, frequency, and the location of monitored hosts. We also use Jimys Probing 0.1.0 for metrics generated from JOnAS / Tomcat server, `apachetop` 0.12.5 for Apache HTTP server, and `mysqladmin` for MySQL database server. We slightly modified `apachetop` to generate XML encoded monitoring results. The client workload generator is designed to simulate remote Web browsers that continuously send HTTP requests, receiving corresponding HTML files, and recording response time as a performance metric during staging. `Sysstat` 7.0.2 was used for system resource utilization tracking.

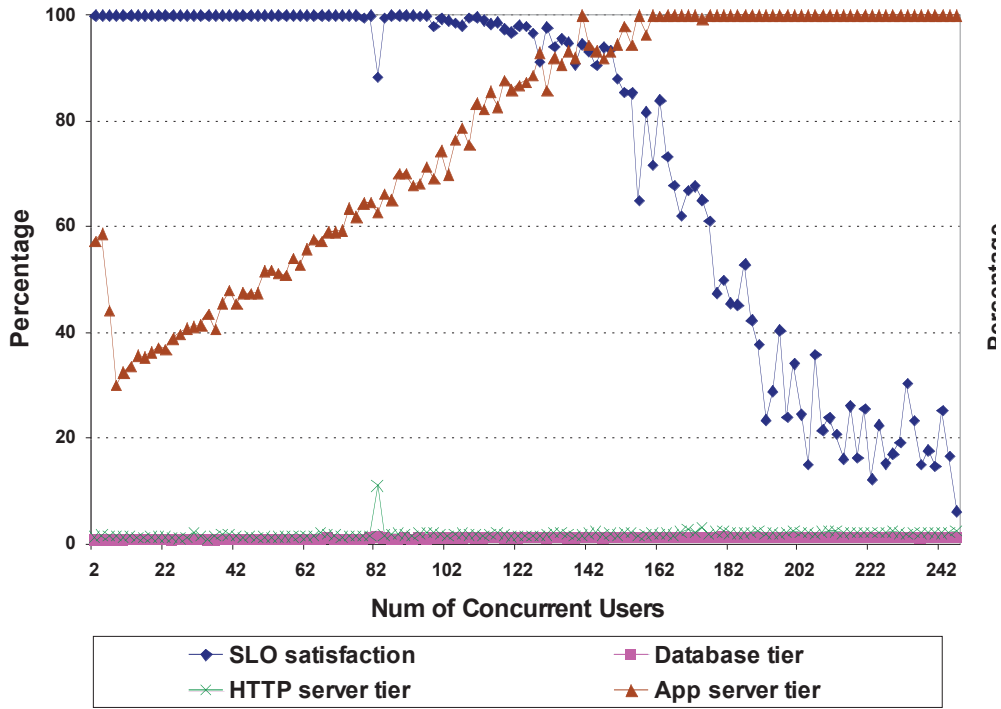
The experimental setup was deployed on two different clusters. Our initial data set (used in Section 5.3.1) was collected on the Georgia Tech Warp Cluster. This cluster is comprised of 56 Intel Blade Servers with Red Hat Enterprise Linux 4, with Linux kernel version 2.6.9-34-i386 as operating system. Each server is equipped with two Xeon 64-bit 3.06 GHz CPUs, 1 or 2 GB main memory, 1 Gbps network adapter, and a 5,400 RPM disk with 8 MB cache. The second cluster used for the data generation was the Emulab [3], which provides more than 200 servers of different types. Emulab also allows the physical separation of experiments by simulating a local network topology for each experiment. The results detailed in Section 5.3.2 incorporate two types of servers. Primarily we employed a high end system with one Xeon 3.0Ghz 64 bit CPU, 2 GB main memory, six 1 Gbps network adapters, and a 10,000 RPM disk. In order to change the bottleneck pattern we also used low end machines with a Pentium P3 600 MHz processor, 256 MB main memory, five 100 Mbps network adapters, and 7,200 RPM disk. Both server types ran with Red Hat Enterprise Linux 4, with Linux kernel version 2.6.9-34-i386.

5.3.1 Bottleneck Detection in the 1/1/1 Configuration

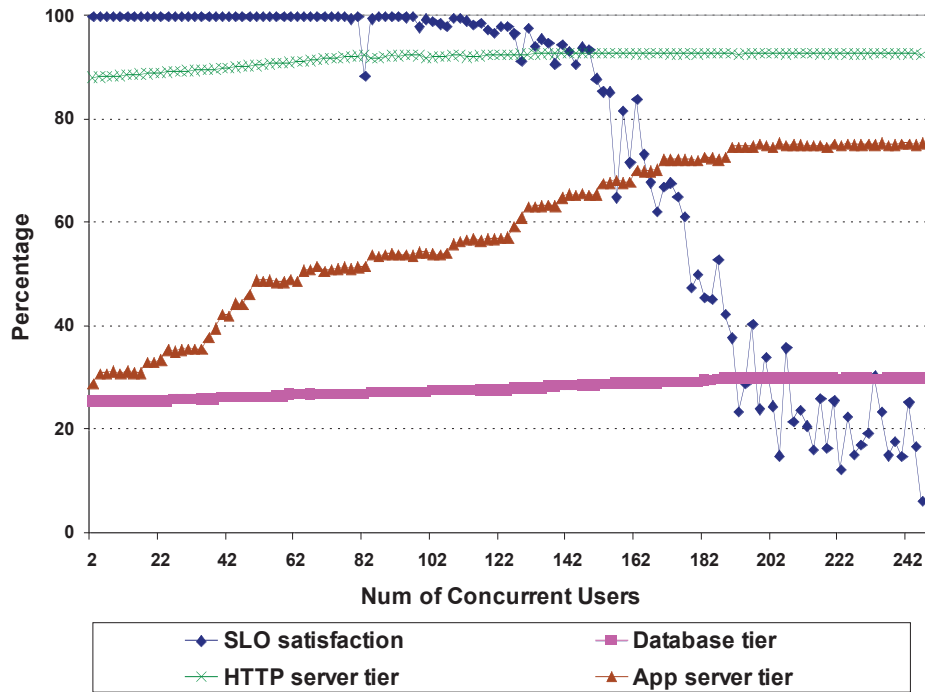
In this section we present a walk-through of our bottleneck detection process for a 1/1/1 configuration (no server replication in the tiers). The RUBiS benchmark was set to use the bidding transition matrices and the workload was incremented in steps of two.

The graphs in Figure 25 show two different representative metrics in all tiers. The SLO satisfaction is graphed against the CPU usage in Figure 25(a) and against the memory usage in Figure 25(b). The depicted satisfaction level needs to be calculated by for each trial. This is conveniently resolved by the policy specific SLO-evaluator that is generated by Mulini. Individual satisfaction levels are determined for each of the SLO components. Interested readers can refer to [7, 51] for more details. For simplicity reasons we solemnly use the response time of the system from the SLO-evaluator in the presented analysis. It is clearly visible that the satisfaction level meets our assumptions. It remains very stable up to around one hundred and fifty users, and decreases rapidly once this point is crossed. The CPU usage of the application server increases linearly and saturates at 100% when the SLO-satisfaction drops down to around 85%. Furthermore, it is evident that the variability of CPU usage strongly decreases at the same time, signifying the maximal saturation level. The trends of other CPU utilizations remain linear and stable on the contrary. In Figure 25(b) we can see that the memory of the application server and database server is underutilized. Both show a fairly stable linear trend. Although the memory usage of the HTTP server is somewhat high its trend is almost flat. The variability of all other metrics stays fairly constant throughout the whole experiment. Following the argument in [51] we can regard the application server CPU a typical representative of a single bottlenecked resource that will show a strongly non-stationary behavior in its delta values (first difference normalized by the step-width). All other delta series will retain a stable behavior throughout the entire work-load span.

We can now turn to the actual detection process. Table 7 contains the output of the algorithm used to determine the intervention as described in Section 5.2.3. We performed the analysis on a dataset consisting of one hundred and seventy-five staging trials. The number



(a) SLO satisfaction versus CPU usage metric.



(b) SLO satisfaction against memory usage metric.

Figure 25: SLO-satisfaction and usage metric graphs for 1/1/1 RUBiS experiment configuration.

Table 7: Heuristic approximation of the intervention point.

AVG [%]	ST-DEV [%]	CI-LB [%]	WL [#]
98.39	2.37	93.75	148
98.31	2.46	93.49	150
98.17	2.72	92.84	152
98.09	2.81	92.58	154
97.79	3.83	90.28	156
97.63	4.05	89.70	158
97.40	4.53	88.52	160
97.26	4.66	88.12	162

Table 8: Results of the bottleneck detection process.

Metric	$q_{0.1}$	$q_{0.5}$	$q_{0.9}$	$q'_{0.2}$	$q'_{0.5}$	$q'_{0.8}$	R
APP_CPU	-1.01	0.05	0.74	-0.62	-0.02	0.58	70.05
DB_KBCached	-3.80	4.63	12.39	-3.56	3.26	11.46	11.72
APP_KBBuffers	7.40	53.00	170.60	0.00	0.00	39.00	3.33
DB_CPU	-0.03	0.00	0.04	-0.03	0.00	0.03	2.81
APP_Memory	-0.15	0.08	1.04	-0.14	0.03	0.38	2.26
WWW_CPU	-0.07	0.00	0.06	-0.05	0.00	0.06	1.08

of concurrent users ranged from two to three hundred and fifty. The lower bound of the confidence interval drops below ninety percent for a workload of one hundred and fifty-eight. According to Section 5.2.3 this defines our heuristic approximation of the crossover point c .

The final outcome of our heuristic testing scheme (Section 5.2.4) and the impact assessment (Section 5.2.5) for a limited dataset (first one-hundred and twenty-five trials) are summarized in Table 8. The two delta value intervals I and I' are [4;156] and [158;250] respectively. While Rule (14) returns six hits, Rule (15) results in an empty set of candidate bottlenecks. All other metrics are discarded automatically. The values in the last column reveal the application tier CPU as most likely bottleneck. Thus our model has correctly detected the bottleneck in this scenario. For further understanding of the table it is important to note that we map all related values to its resource for the final interpretation. Therefore the R value of the APP_CPU can result from more than one metric (e.g., the overall usage or the system usage), for instance.

We now proceed to demonstrate the robustness of our method when subjected to various

Table 9: Ranking function value against interval width.

I	I'	R^*	Pred Acc
[4;156]	[158;206]	6.94	1
[4;156]	[158;256]	68.32	1
[4;156]	[158;350]	13.92	1
[58;156]	[158;206]	3.83	1
[58;156]	[158;256]	7.66	1
[58;156]	[158;350]	15.01	1
[108;156]	[158;206]	4.33	1
[108;156]	[158;256]	8.67	1
[108;156]	[158;350]	17.11	1

Table 10: Ranking function value against different crossover points.

I	I'	R^*	Pred Acc
[16;114]	[116;214]	-	0
[26;124]	[126;224]	-	0
[36;134]	[136;234]	5.09	1
[46;144]	[146;244]	6.86	1
[56;154]	[156;254]	7.93	1
[66;164]	[166;264]	∞	1
[76;174]	[176;274]	59.43	1
[86;184]	[186;284]	0.93	0

configuration settings. We show that our approach is highly robust with regard to variations in the width of the intervals (Table 9), the position of the crossover point (Table 10), and the step-width between the different workloads (Table 11). Table 9 shows the value of the ranking function depending on the length of the two input intervals. Our model predicted the bottleneck correctly each time. We see that the width of the second interval influences the magnitude of the R value strongly.

We now examine the impact of different choices of crossover point values, which are summarized in Table 10. Within certain intuitive limits the model predicts correctly. This is of special importance since the determination of the intervention point is the result of the algorithm in Section 5.2.3. We see that its heuristic character is justified in the nature of the data.

Finally we turn to Table 11 and the analysis of the robustness of the choice of step-width.

Table 11: Ranking function value against the step width.

I	I'	Step	# Trials	R^*	Pred Acc
[4;156]	[158;350]	2	174	13.92	1
[4;156]	[160;348]	4	87	12.58	1
[4;156]	[164;348]	8	44	188.60	1
[4;148]	[164;340]	16	22	112.17	1
[4;132]	[164;324]	32	11	∞	1

The table contains the ranking function values for different step-widths. At first it looks surprising that the ranking values increase almost monotonically as the step-width increases. Nevertheless, this is due to the stochastic nature of our data and the fact that with increased step width the two intervals are separated farther apart. By increasing the observable change in the bottleneck metric the results become clearer. This proves the robustness of our method and reveals its effectiveness when exposed to data of higher step-width.

5.3.2 Performance Comparison of the Analysis

In this section we present the results of our experiments across a wide range of configurations to show how the method evaluates a changing bottleneck pattern. This data was collected on Emulab [3] with different write ratios (WR0.1 and WR0.2) and server configurations (H and L).

Table 12 lists the top candidate bottleneck metrics and their R value in the last two columns. We also applied a simple heuristic filtering mechanism to discard uninteresting ranking values. The latter eliminates the problematic behavior of some utilization values for instance, which was detailed in our previous work. We automatically discard values if the utilization does not cross a certain threshold (90% in our case) [51]. The table shows that our methodology is able to detect the shifting bottleneck as we progress to higher replication levels of the application server tier.

In order to make the performance limitation appear faster we employed a lower write ratio and lower end DBs in the last two data sets. At a replication level of eight application servers the bottleneck has shifted to the database tier. Our algorithm identifies the DB

Table 12: Results of the bottleneck detection process.

Config	WR	W	# Trials	c	S1/S2-Sz	Result Metric	R^*
H/2H/H	20%	100–600	51	440	11/2	APP_CPU	73.02
H/4H/H	20%	540–1,040	51	844	4/1	APP_CPU	16.61
H/6H/H	20%	1040–1,448	51	1,264	7/0	APP_CPU	13.10
H/8H/L	10%	1300–1,820	51	1,490	4/0	DB_Memory	10.22
H/8H/2L	10%	1400–1,920	51	1,640	5/1	APP_CPU	7.82

memory as a potentially saturated resource. Now we examine the effect of replicating the bottlenecked DB. This again results in a shift of the bottleneck towards the application tier and is successfully detected by our algorithm. It is also evident that we are able to perform our detection process accurately with a significantly lower number of trials than other approaches.

5.4 Related Work

The area of performance modeling in multi-tier enterprise systems has been subjected to substantial research efforts in the recent time. Many of the well-documented approaches use machine learning or queuing theory.

Cohen et al. apply a tree-augmented Naive Bayesian network to discover correlations between system-level metrics and performance states, such as SLO satisfaction and SLO failure [31]. Powers et al. also use machine learning techniques to analyze performance [80]. However, rather than detecting bottlenecks in the current system, they predict whether the system will be able to withstand load in the following hour. Similarly, we have performed a comparative study of machine learning classifiers to investigate performance patterns [7]. Our goal was to compare the performance of several well-known machine learning algorithms as classifiers in terms of bottleneck detection, and finally to identify the classifier that best detects bottlenecks in multi-tier applications. Several other studies are based on dynamic queuing models combined with predictive and reactive provisioning [97]. Their contribution allows an enterprise system to increase capacity in bottleneck tiers during flash crowds in production.

Elba, in addition to being oriented towards avoiding in-production performance shortfalls, emphasizes fine-grained reconfiguration. By identifying specific limitations such as low-level system metrics (CPU, memory, etc.) and higher-level application parameters (pool size, cache size, etc.) configurations are tuned to the particular performance problem at hand. Another fundamental difference of our work is that in addition to correlating metrics to performance states, we focus on the detection of actual performance-limiting bottlenecks. We employ a unique procedure to analyze the change in trends of metrics. Finally, our set of metrics for bottleneck detection includes over two hundred application-level metrics as well as system-level metrics.

5.5 Conclusion

Our detection scheme based on intervention analysis has proven to be very effective with our experimental data. The method is able to characterize the potential change in metric graph trends automatically and assess its correlation with SLO violations. Our statistical modeling approach eliminates the previously necessary data filtering (e.g., correlation analysis) and model calibrating phases (e.g. classifier training). The results are clear and intuitive in the interpretation. We showed that our new method yields these general as well as practical advantages in our evaluation. Potential bottlenecks are identified accurately in different scenarios. As future work this method could be extended with a maximization / minimization scheme for the ranking function. This would allow a more thorough root-cause analysis in the case of multiple bottlenecks. We also plan to employ our results as input for a regression model that will be able to predict actual SLO satisfaction levels.

CHAPTER VI

EXPERIMENTAL EVALUATION OF N-TIER SYSTEMS: OBSERVATION AND ANALYSIS OF MULTI-BOTTLENECKS

In many areas such as e-commerce, mission-critical n-tier applications have grown increasingly complex. They are characterized by non-stationary workloads (e.g., peak load several times the sustained load) and complex dependencies among the component servers. We have studied n-tier applications through a large number of experiments using the RUBiS and RUBBoS benchmarks. We apply statistical methods such as kernel density estimation, adaptive filtering, and change detection through multiple-model hypothesis tests to analyze more than 200GB of recorded data. Beyond the usual single-bottlenecks, we have observed more intricate bottleneck phenomena. For instance, in several configurations all system components show average resource utilization significantly below saturation, but overall throughput is limited despite addition of more resources. More concretely, our analysis shows experimental evidence of multi-bottleneck cases with low average resource utilization where several resources saturate alternatively, indicating a clear lack of independence in their utilization. Our data corroborates the increasing awareness of the need for more sophisticated analytical performance models to describe n-tier applications that do not rely on independent resource utilization assumptions. We also present a preliminary taxonomy of multi-bottlenecks found in our experimentally observed data.

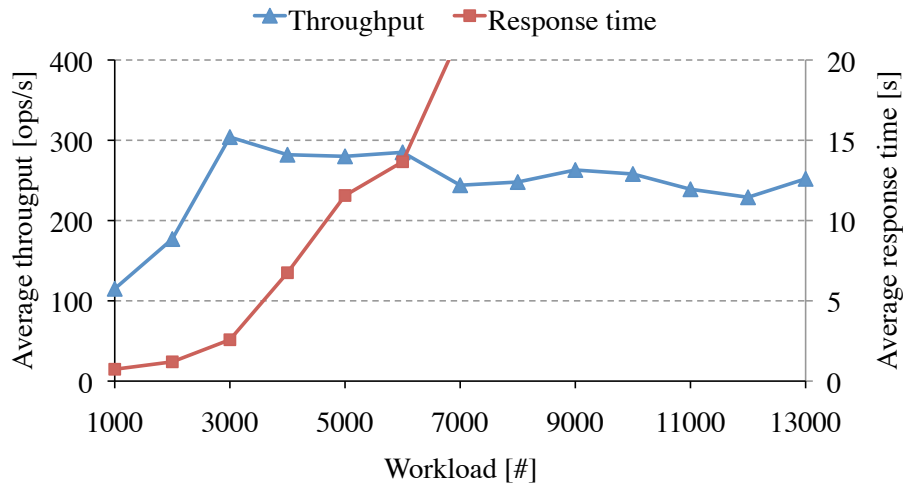
6.1 Introduction

In modern datacenters, enterprise-class n-tier systems with web servers, application servers, and database servers are growing in economic importance, infrastructure footprint, and application complexity. Traditional performance analysis methods are challenged by this

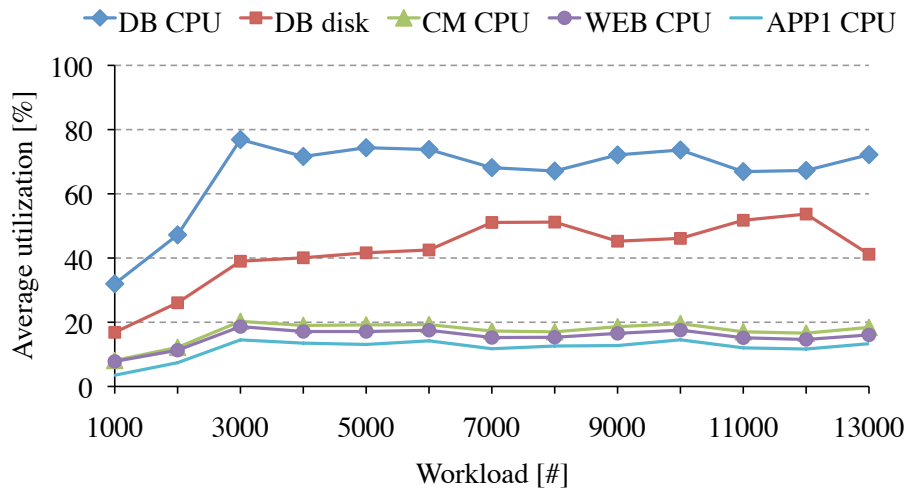
growth due to bottleneck phenomena that so far have been considered rare and unusual. In this paper, we show experimentally such “unusual” phenomena and how to detect and analyze them. Our data from system configurations reveal bottleneck cases with different kinds of partially saturated resources, which can be grouped according to their resource usage dependence and saturation frequency into *oscillatory* bottlenecks, *concurrent* bottlenecks, and *simultaneous* bottlenecks. In order to distinguish these cases from single-bottlenecks, which have traditionally been recognized as predominant saturation phenomenon, we use the umbrella-term of *multi-bottlenecks*.

Multi-bottlenecks have been previously studied in system theory [19, 26, 61] and system analysis [83, 84]. However, classical computer performance analysis [46, 58], which started with stable workloads on mainframes, is traditionally restricted to detecting single-bottlenecks. Typical assumptions of analytical models (e.g., mean value analysis in queuing theory) are the independence of arriving tasks and service times. Partially motivated by the growth of n-tier systems, some recent performance models have moved beyond these assumptions. Consequently, the importance of bursty workload conditions [25, 70] and non-stationary workload in general [93] have been recognized. Nevertheless, average utilization values remain the method of choice in top-down n-tier system analysis [14, 28, 81, 109]. When mentioned [25, 109], phenomena such as multi-bottlenecks in n-tier applications have been described as “challenging case”.

In this paper we analyze experimental results with multi-bottlenecks. We show that these bottlenecks may happen while system components show average resource utilization significantly below saturation. Figure 26 exemplifies this case by contrasting the throughput and response time of a saturated RUBBoS experiment with the corresponding candidate-bottleneck resources. While there is an obvious correlation between system throughput and database CPU utilization, the question why this correlated resource does not seem to saturate remains unanswered. Because none of the resources show full resource utilization levels, standard approaches to bottleneck detection in n-tier systems (e.g., [60]) are not able to



(a) Average throughput and response time.



(b) Resource utilization for candidate-bottleneck resources.

Figure 26: 1/2/1/1L RUBBoS experiment with read/write workload.

diagnose them. However, instead of attributing the obvious performance limitation to some hidden (i.e., unmonitored) resource, we show—using our methodology—that the observed throughput limitation is due to an oscillatory bottleneck in the database where CPU and disk saturate alternatively (see Section 6.6).

The main contribution of this paper is threefold. First, our experimental evaluation of n-tier application benchmarks (i.e., RUBiS and RUBBoS) documents the presence of presumed unusual multi-bottlenecks. Second, we introduce a simple classification of multi-bottlenecks that enables the detection of the observed phenomena. Third, using statistical methods such as kernel density estimation, adaptive filtering, and change detection through multiple-model hypothesis tests, we show how to detect multi-bottlenecks with an efficient top-down approach, even if average resource utilization is significantly below saturation.

The remainder of this paper is structured as follows. In Section 6.2 we introduce the simple classification of multi-bottlenecks that is used in our analysis. In Section 6.3 we outline the necessary methods for statistical interpretation of the measurement data. Section 6.4 presents the experimental setup and infrastructure. Section 6.5 shows a scenario with seven concurrent bottlenecks. In Section 6.6 we present two different oscillatory bottleneck cases. Related work is summarized in Section 6.7, and Section 6.8 concludes the paper.

6.2 *Simple Classification of Multi-bottlenecks*

The common understanding of a *system bottleneck* (or bottleneck for short) can intuitively be derived from its literal meaning as the key limiting factor for achieving higher system throughput. Hence, an improvement to the throughput of the bottleneck resource results in the highest possible system throughput improvement [46, 103]. Classical queuing theory defines the set of bottlenecks in a queuing system as follows. B is the set of all resources i that reach full utilization U_i when N , the number of jobs in the system, tends to infinity under stable class-mix conditions [19].

$$B = \left\{ i \mid \lim_{N \rightarrow \infty} U_i(N) = 1 \right\} \quad (17)$$

Due to their simplicity and intuitiveness, definitions similar to Equation (17) have usually provided the foundation for reasoning about bottleneck behavior in computer system performance analysis. But despite their popularity, such formulations are based on assumptions that do not necessarily hold in practice. In other words, we show experimentally that Equation (17) does not allow correct identification of bottlenecks in the case of empirical n-tier application monitoring data. Our data suggest that resources in n-tier applications cannot generally be assumed to exhibit independent utilization. In fact, bottlenecks may be comprised of more than one physical resource. Similarly, the dimension of time has to be taken into account in n-tier systems, which may be subject to very strong workload fluctuations. Therefore, the assumption of stable class-mix conditions has to be relaxed as well. Finally, an infinite number of users is not technically feasible, hence, the assumption of full resource utilization has to be adapted to actual monitoring conditions.

Since our data show that the aforementioned assumptions may be too rigid, the need arises to classify the observed phenomena in an alternate way. But before doing so through a formal set of definitions in the later part of this section, we first establish the key differences between our taxonomy and common approaches to n-tier bottleneck detection. Because of queuing theory, the term bottleneck is often used synonymously for single-bottleneck. In a single-bottleneck cases, the saturated resource typically exhibits a near-linearly load-dependent average resource utilization that saturates at a value of one past a certain workload. Consequently, such bottlenecks are straight-forward to detect. However, the characteristics of bottlenecks may change significantly if more than one bottleneck resource is present in the system, which is the case for many real n-tier applications with heterogeneous workloads. Therefore, we distinguish the phenomena discussed in this paper from single-bottlenecks by using the umbrella-term *multi-bottlenecks*.

Because system resources may be causally dependent in their individual usage patterns,

multi-bottlenecks introduce the classification dimension of *resource usage dependence*. Additionally, greater care has to be taken in classifying bottleneck resources according to their *resource saturation frequency*. We distinguish between resources that saturate for the entire observation period (i.e., fully saturated) and resources that saturate for only certain parts of the period (i.e., partially saturated). Note that previous efforts in this area have typically omitted the notions of dependence and saturation frequency in their analysis (e.g., [60]).

Figure 27 summarizes the classification that forms the basis of our multi-bottleneck detection. The x-axis and y-axis represent resource saturation frequency and resource usage dependence, respectively. During our extensive empirical evaluation, we have only observed three of the four possible combinatoric cases, which suggests that if resources exhibit full saturation, their resource utilization is not dependent on any other saturated resource. Under this assumption, multiple fully saturated resources, imply resource usage independence, which is classified as *simultaneous* bottleneck case. A well-known example of systems with resources that are resource usage independent are embarrassingly parallel applications. *Concurrent* bottlenecks happen if multiple resources saturate independently of each other, but only for a fraction of the observation period. As the load increases, a concurrent saturation pattern may transform into simultaneous saturation. The most notable characteristic of usage independent resources is that each resource's saturation may be evaluated separately. On the contrary, the detection of *oscillatory* bottlenecks is more challenging because multiple resources form a combined bottleneck, which can only be analyzed and resolved in union. Oscillatory bottlenecks consist of partially saturated resources and are caused by resource usage dependencies (e.g., load-balancing policies) that are an inherent characteristic of complex n-tier systems. Such dependencies result in a usage alternation between the saturated resources (i.e., there is no overlapped saturation), which is observable as interleaved saturation patterns. The latter may cause particularly low average saturation for every single resource, and observable saturation frequency can become difficult

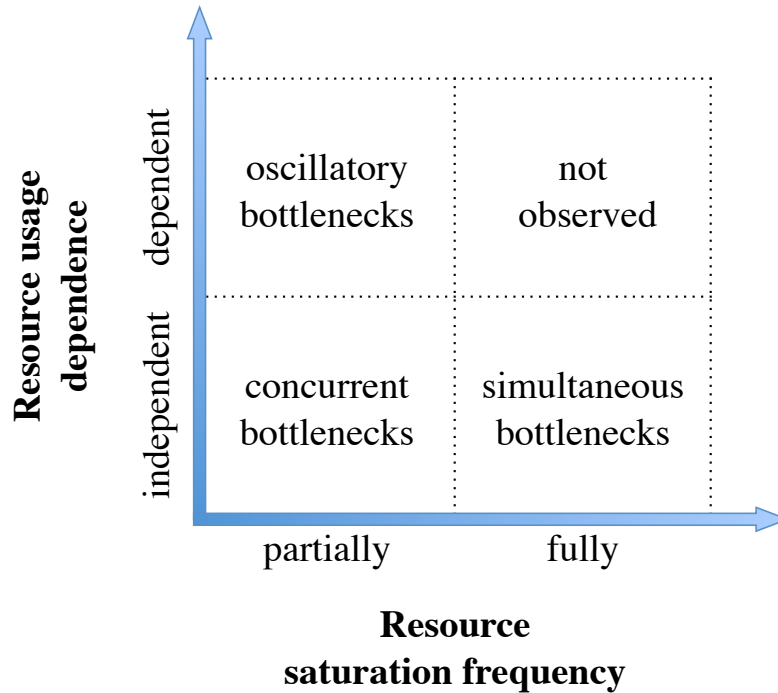


Figure 27: Simple multi-bottleneck classification.

to interpret. Therefore, it is inevitable to take resource usage dependence into account in multi-bottleneck detection. We find that once resources are grouped according to their usage dependence, it is possible to compute their combined frequency and derive a representative measure for the bottleneck magnitude.

Given these findings, the following definitions are sufficient for a simple multi-bottleneck classification. The resulting generic schema may be parameterized in a concrete setting, such as presented in this paper, and utilized to detect bottlenecks in n-tier application monitoring data. In the following, we augment the definitions with our concrete parameter choices, leaving sensitivity analysis and parameterization comparisons as an interesting topic for future research.

Critical saturation. *System resource $i \in I$, where I is a set of system resources, is critically saturated for an observation interval t of length $\lambda > 0$ iff its utilization exceeded a saturation threshold $\alpha \in [0, 1]$ for this interval, whereby α is referred to as the critical saturation threshold.*

We found that parameterizing tuple (λ, α) with 1 second and 95%, respectively, leverages workload fluctuation, technical feasibility, and standard statistical significance well in our data. Set I consists of all monitored resources, which are all CPUs, disks, and network links in our testbed.

Resource saturation frequency. *The resource saturation frequency f_R^i (or frequency for short) of resource $i \in I$ is the number of intervals with critical saturation divided by the total number of intervals in observation period Λ .*

We adopted the system observation period Λ equal to the experiment runtime of 8 minutes. Therefore, the resource saturation frequency f_R^i indicates how often resource demand exceeded the capacity of resource i during runtime.

Fully saturated resource. *A fully saturated resource $i \in I$ has maximal saturation frequency; i.e., $f_R^i \approx 1$.*

We found that $f_R^i \geq 0.95$ constitutes a convenient detection rule for full saturation of resource i .

Partially saturated resource. *A partially saturated resource $i \in I$ is not fully saturated, but has a non-negligible saturation frequency; i.e., $f_R^i \in (0, 1)$.*

We found that $0.01 < f_R^i < 0.95$ constitutes a convenient detection rule for partial saturation of resource i .

Bottleneck resource. *A bottleneck resource $b \in I$ is either a fully saturated resource or a partially saturated resource.*

Resource usage dependence. *A bottleneck resource $b_j \in I$ is resource usage dependent (or dependent for short), if there exists another bottleneck resource $b_k \in I$ such that their binary critical saturation states $B_j^t \in \{0, 1\}$ and $B_k^t \in \{0, 1\}$ have a mutually exclusive relationship for any given interval t ; i.e., $\text{Prob}(B_j^t = 1 \wedge B_k^t = 1) \leq \varepsilon$ with $0 < \varepsilon \ll 1$ for all t in Λ .*

We found that this definition implies an intuitive rule in a top-down approach to bottleneck detection. Because of the mutual exclusion property, dependent bottleneck resources do not exhibit overlap of critical saturation intervals among each other. Even at very high workloads, such overlap has a statistically insignificant probability; e.g., $\text{Prob}(\text{overlap}) \leq 5\%$.

Resource usage independence. *A bottleneck resource $b_j \in I$ is resource usage independent (or independent for short), if there exists no other bottleneck resource $b_k \in I$ such that their binary critical saturation states $B_j^t \in \{0, 1\}$ and $B_k^t \in \{0, 1\}$ have a mutually exclusive relationship for any given interval t ; i.e., $\text{Prob}(B_j^t = 1 \wedge B_k^t = 1) > \varepsilon$ with $0 < \varepsilon \ll 1$ for all t in Λ .*

In contrast to dependent bottleneck resources, independent bottleneck resources show a clearly increasing overlap of critical saturation intervals with growing workload; e.g., $\text{Prob}(\text{overlap}) > 5\%$.

Single-bottleneck. *A single-bottleneck $\beta \in I$ is either a fully saturated bottleneck resource or partially saturated bottleneck resource iff there exists only one bottleneck resource in the system.*

Oscillatory bottleneck. *An oscillatory bottleneck $\beta \subseteq I$ is a set of partially saturated, resource usage dependent bottleneck resources iff there exist more than one bottleneck*

resource in the system.

In this paper, we focus on the maximal set of resources that form an oscillatory bottleneck. The study of subsets of resources in such oscillations is a subject of future research.

Concurrent bottleneck. *A concurrent bottleneck $\beta \in I$ is a partially saturated, resource usage independent bottleneck resource iff there exist more than one bottleneck resource in the system.*

Simultaneous bottleneck. *A simultaneous bottleneck $\beta \in I$ is a fully saturated, resource usage independent bottleneck resource iff there exist more than one bottleneck resource in the system.*

Multi-bottleneck. *A multi-bottleneck is either an oscillatory bottleneck, a concurrent bottleneck, or a simultaneous bottleneck.*

Bottleneck. *A bottleneck is either a single-bottleneck or a multi-bottleneck.*

Bottleneck saturation frequency. *The bottleneck saturation frequency f_B^β (or frequency for short) of bottleneck β is the number of intervals where at least one of the bottleneck resources in β is critically saturated divided by the total number of intervals in observation period Λ .*

The bottleneck saturation frequency indicates how often resource demand exceeded the capacity of any of the resources that are analyzed in union due to their strong usage dependence. Hence, this measure allows the assessment of how often a system under examination suffered of a particular performance limiting phenomenon.

Primary bottleneck. *The primary system bottleneck is the bottleneck with the highest*

bottleneck saturation frequency.

Note that there can be more than one primary bottleneck and that any single-bottleneck or simultaneous bottleneck is also a primary bottleneck by definition.

6.3 Statistical Data Interpretation

In the following we provide a brief overview of the most important statistical concepts that formed the basis of our data analysis. Since real systems are typically subject to variable request characteristics, it is necessary to analyze the distributions of resource utilization values to infer actual saturation characteristics. However, histograms are often poor estimates of unknown density functions [11], therefore, we chose estimation through kernel densities [41], instead. Given a finite sample X_1, \dots, X_n from a univariate distribution, the unknown density function g can be estimated from the observed data using kernel regression. Although literature offers various kernel functions, the symmetric Gaussian density is a popular choice in density estimation. Given a value x , the kernel function c , and a smoothness parameter h (i.e., “bandwidth”) the density estimator \bar{g}_c takes the following form.

$$\bar{g}_c = \frac{1}{n} \sum_{t=1}^n \frac{k}{h} \left(\frac{x - X_t}{h} \right) \quad (18)$$

There are different approaches to calculating bandwidth h . We choose a common strategy (i.e., AMISE estimation [41]), which simplifies to estimation with the sample standard deviation estimate $\bar{\sigma}$.

$$\bar{h}_0 = \frac{\bar{\sigma} \sqrt[3]{4}}{\sqrt[3]{n}} \quad (19)$$

An example of a resulting three-dimensional density graph is shown in Figure 29(a).

In the analysis of multi-bottlenecks, it is further necessary to assess the dependence relationship of resources in excess of their relative probabilities. In order to reduce the

complexity of this analysis in an efficient top-down approach, it is possible to segment the monitoring data into piecewise constant functions. This method provides intuitive aggregation and reduces the data size significantly compared to bottom-up analysis. Such a segmenting problem is also known as adaptive filtering problem that detects abrupt changes in streams of noisy data and has been previously studied in signal processing. Therefore, we perform an automated multiple-model hypothesis test on a set of signal estimation models, which are based on different assumptions each, to divide the measured time series into segments according to piecewise constant mean and variance models [40]. Given the time index t , the noise measure e , the parameter vector θ , and the noise variance γ , each signal y can be expressed as follows.

$$y(t) = \theta(t) + e(t) \quad (20)$$

$$Ee(t)^2 = \gamma(t) \quad (21)$$

In our case $y(t)$ in (20) is identical to X_t in (18). The set of final change times is determined algorithmically [40]. This approach is directly interpretable as characterization of each stable interval of the piecewise constant resource utilization functions. We characterize each utilization interval with respect to being statistically distinguishable from critical saturation (see Section 6.2). Such a classification is performed for each bottleneck resource metric, and the results can be summarized in a simple graph for the entire system (e.g., Figure 30).

6.4 Experimental Setup

Among n-tier application benchmarks, RUBBoS and RUBiS have been used in numerous research efforts due to their real production system significance. In our experiments, the run consist of an 8-minute ramp-up, a 12-minute run period, and a 30-second ramp-down. Performance measurements (e.g., CPU or network utilization) are taken during the run using Linux account logging utilities (i.e., Sysstat) with one-second intervals.

RUBBoS [5] is an n-tier e-commerce system modeled on bulletin board news sites similar to Slashdot. The benchmark can be implemented as 3-tier (web server, application server, and database server) or 4-tier (with the addition of cluster middleware such as C-JDBC) systems. The benchmark places high load on the database tier. The workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. The benchmark includes two kinds of workloads: browse-only and read/write interaction mixes.

RUBiS [6] is an n-tier system benchmark modeled on online auction sites such as eBay. The benchmark can be implemented as a 3-tier, 4-tier, or 5-tier system. We have chosen a configuration consisting of web server, web container, EJB container, cluster middleware, and database server. Web and EJB containers are deployed together on the same physical nodes. The benchmark usually places a high load on application servers. The workload consists of 26 interactions such as register user, sell item, and place bid. RUBiS includes two kinds of workloads: browse-only and read/write interaction mixes.

The experiments used in this paper were run in the Emulab testbed [3] with various types of servers. Table 13(b) contains a summary of the hardware used in our experiments. Normal and low-cost nodes were connected over 1,000 Mbps and 100 Mbps links, respectively. The experiments were carried out by allocating a dedicated physical node to each server. In the initial setting all components were normal nodes. As an alternative, database servers were also hosted on low-cost machines. We use a four-digit notation #W/#A/#C/#D to denote the number of web servers, application servers, cluster middleware nodes, and database servers. The server node type is either *normal* or *low-cost* (“L”). If a specification is omitted, it can be assumed that the default node type (i.e., normal) has been used. A sample topology of an experiment with one web server, two application servers, one cluster middleware node, and two low-cost database servers (i.e., 1/2/1/2L) is shown in Table 13(c).





The presented dataset is part of an ongoing effort, for which we have run a very high number of experiments over a wide range of configurations and workloads. A typical RUBiS

Table 13: Details of the experimental setup on the Emulab cluster.

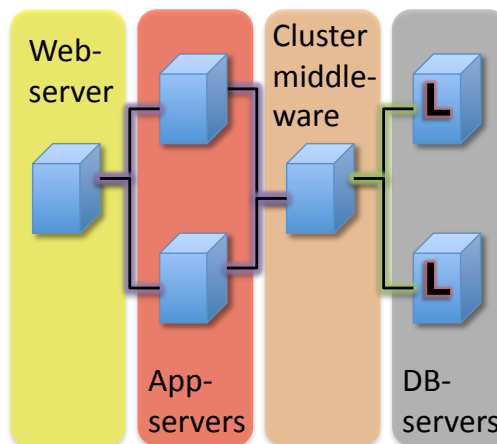
(a) Software setup

Classification	Software
Web server	■ Apache 2.0.54
Application server	■ Ap. Tomcat 5.5.17 ■ JOnAS 4.6.6
Cluster middleware	■ C-JDBC 2.0.2
Database server	■ MySQL 5.0.51a
Operating system	Redhat FC4 Kernel 2.6.12
System monitor	Systat 7.0.2

(b) Hardware node setup

Type	Components		
Normal 	Processor	Xeon 3GHz	64-bit
	Memory	2GB	
	Network	6 x 1Gbps	
	Disk	2 x 146GB	10,000rpm
Low-cost 	Processor	PIII 600Mhz	32-bit
	Memory	256MB	
	Network	5 x 100Mbps	
	Disk	13GB	7,200rpm

(c) Sample topology (1/2/1/2L)



or RUBBoS experimentation cycle requires thousands of lines of code that need to be managed for each experiment. The experimental data output are system metric data points (i.e., network, disk, and CPU utilization) in addition to higher-level monitoring data (e.g., response times and throughput). Although the scripts contain a high degree of similarity, the differences among them are subtle and important due to the dependencies among the varying parameters. Maintaining these scripts by hand is a notoriously expensive and error-prone process.

To enable experimentation at this scale, we employed an experimental infrastructure created for the Elba project [7] to automate system configuration management, particularly in the context of n-tier system staging. The Elba approach [81] divides each automated staging iteration into steps such as converting policies into resource assignments [86], automated code generation [50], benchmark execution, and analysis of results.

6.5 Concurrent Bottlenecks

In this section we show a RUBiS benchmark experiment that exhibits concurrent saturation of application server CPUs and the database network link. More concretely, these data are obtained with a 1/6/1/1L RUBiS configuration using the browse-only interaction mix. Traffic shaping (implemented by the packet scheduler modules in the Linux NET3 kernel) has been used to throttle the network bandwidth between the application servers and the database server. Packets are added to a scheduler queue tree and shaped corresponding to a 2Mbps bandwidth limit using the drop-tail queuing discipline. The system workload ranges between 500 and 1,400 users in steps of 100.

Figure 28 summarizes the overall system performance, which is observable in the clients. The throughput increases near-linearly up to the performance knee at around 800 concurrent users where the rate saturates at around 140 interactions per second. Similarly, the average response time exceeds six seconds past a workload of 800 users. Evidently, there is at least one bottleneck in the system that needs to be identified through a detailed analysis.

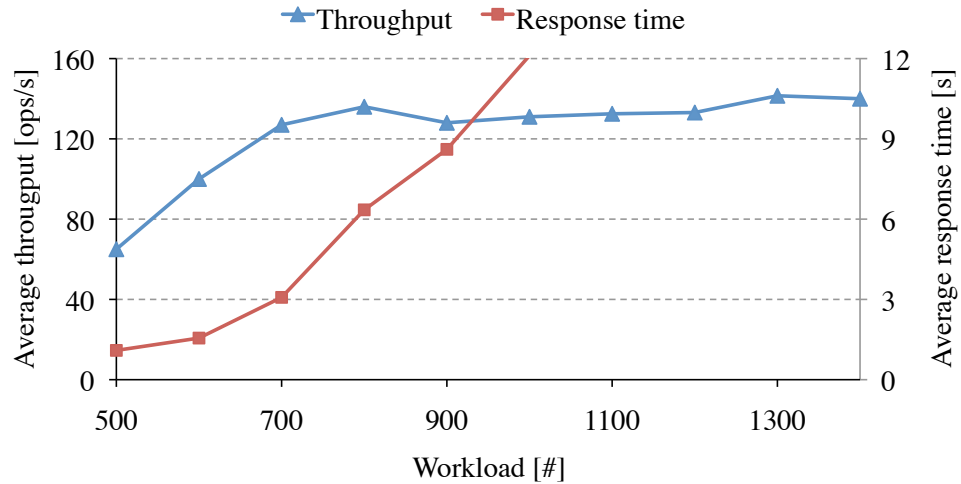
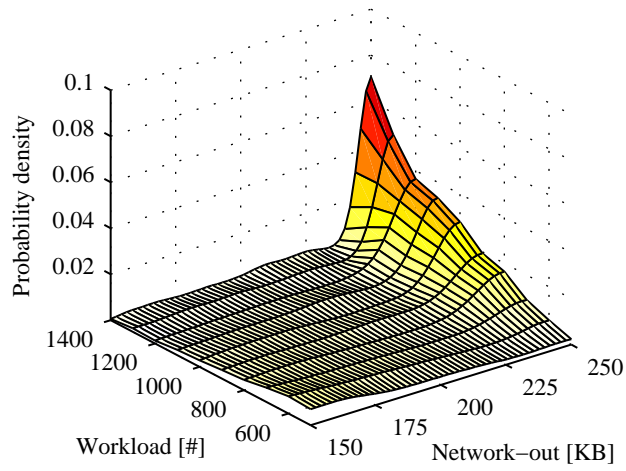
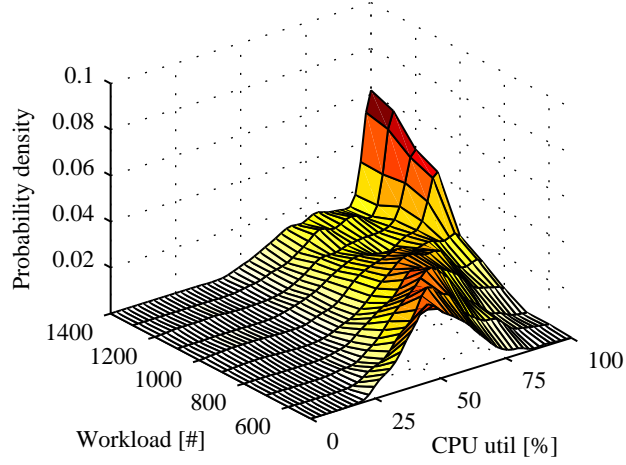


Figure 28: Average throughput and response time of a 1/6/1/1L RUBiS experiment with browse-only workload and throttled database network bandwidth (2 Mbps).

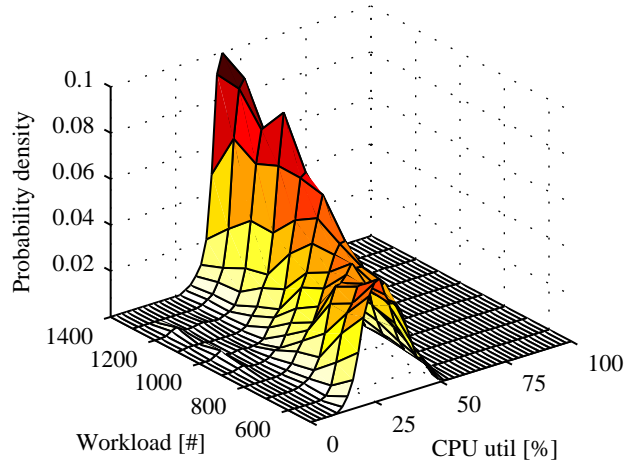
Figure 29 shows the summary of the bottleneck analysis for this scenario. The bottlenecks in the system are best illustrated by studying their resource utilization as a function of workload. The three density graphs in Figures 29(a), 29(b), and 29(c) show the increasing demand peak (z-axis) on the resource utilization (x-axis) as the workload grew from 500 to 1,400 users (y-axis). More specifically, the three density graphs contrast the changes in the resource consumption profiles of bottleneck resources (i.e., application server CPUs and database network) and non-saturated resources (i.e., database CPU). Note that the application server CPU graph is representative of all application servers. While the first graph (Figure 29(a)) shows the continuous saturation of the network bandwidth at 250 KBps (consistent with the link shaping parameter of 2 Mbps), the application server CPU utilization (Figure 29(b)) transitions from a normal to a multimodal saturated curve shape with a dominating mode at the upper capacity limit. This multimodality, which causes average utilization values to be misleadingly low, is the result of an unstable resource consumption profile due to non-stationary request sequences. For higher workloads the application server CPUs vary between being critically utilized and being underutilized. Overall, the seven saturated system resources limit the number of tasks going into the database server. Therefore, its



(a) DB network outgoing rate density.



(b) App1 CPU utilization density.



(c) DB CPU utilization density.

Figure 29: Detailed bottleneck analysis of a 1/6/1/1L RUBiS experiment with browse-only workload and throttled database network bandwidth (2 Mbps).

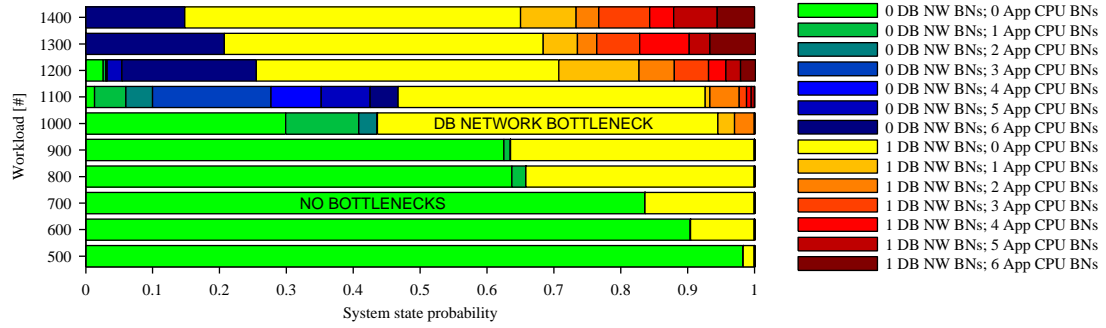


Figure 30: Probability distribution (i.e., frequency of resource saturation states) among the fourteen different bottleneck resource saturation states, in 1/6/1/1L RUBiS experiment with browse-only workload and throttled database network bandwidth (2 Mbps).

CPU remains unsaturated, and its utilization density retains its normal shape despite growing intensity throughout for the entire workload span (Figure 29(c)).

In order to precisely characterize whether the uncovered bottleneck resources are concurrent or oscillatory bottlenecks, we estimated the saturation frequencies of each resource and visualized them in Figure 30. Although there are fourteen different combinations, the concurrent character of the seven bottlenecks is apparent. There is a clear linear decrease in probability of non-overlapping bottleneck states, and a corresponding linear increase in overlapping bottleneck states as the workload grew to 1,400 users. For workloads higher than 1,000, the probability of states without critical saturation of at least one resource is statistically insignificant. Both, the frequency of the network bottleneck and the CPU bottlenecks are solely load-dependent, which implies saturation-independence among the resources. Because none of the resources is fully saturated, we can conclude that the system suffers of seven concurrent bottlenecks in the six application server CPUs and the database network link.

6.6 Oscillatory Bottlenecks

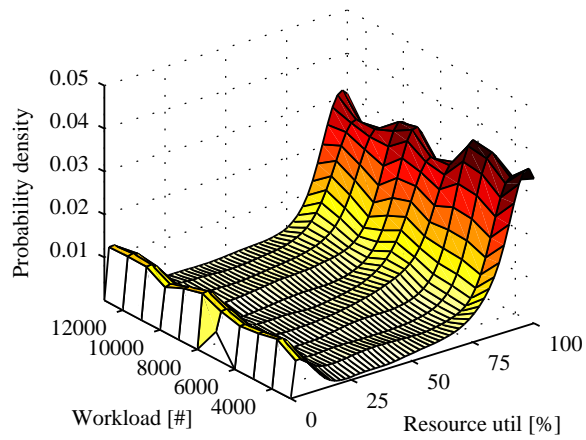
A common assumption in computer system performance analysis is that low average resource utilization implies absence of saturation. However, in the following we show that a standard RUBBoS read/write mix workload may cause request sequences in the database that result

in alternating partial resource saturation with particularly low average resource utilization. While Section 6.6.1 details a scenario with an oscillatory bottleneck within a single node, Section 6.6.2 shows an example of an oscillatory bottleneck with eight resources distributed in the database tier.

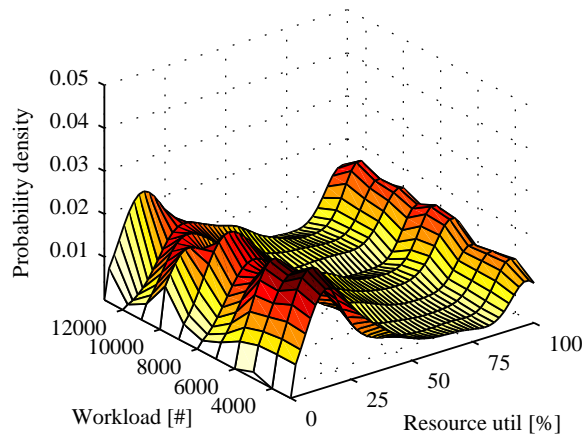
6.6.1 Within-node Dependences

As previously shown in Figure 26(a), the 1/2/1/1L RUBBoS scenario under read/write workload exhibits a clear overload pattern past workloads of 3,000 user. Nevertheless, the average utilization values for candidate bottleneck resources (Figure 26(b)) remain far from saturation. Although the comparison of these two figures reveals a strong correlation between system throughput and database CPU utilization, the CPU utilization seems to saturate at an average of less than 80 percent.

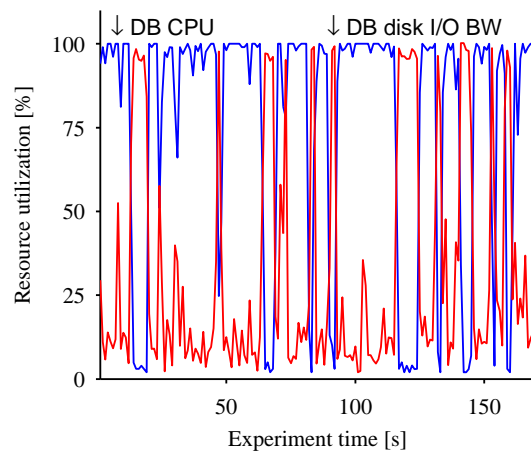
In order to diagnose the system bottlenecks, we turn to Figure 31. The analysis of Figures 31(a) and 31(b) yields the explanation of the previously observed low utilization values. In fact, the two examined densities (database CPU and disk) are both bimodal with two distinct concentration sectors for the entire workload span. Both utilization metrics seem to vary between low and high values during the experiment time. A sample examination of the two time series for a short interval of runtime for 12,000 users (Figure 31(c)) seems to support this impression. However, it is important to note that such an examination is not scalable and requires human intuition to derive insights in the actual system behavior. Therefore, we automatically summarize the metric data in Figure 32, which reveals two bottleneck resources with four saturation states for the entire workload span. Clearly, overlapping saturation is extremely rare. The ratio of time fractions for the two non-overlapped bottleneck state seems to be relatively stable, which further strengthens the assumption of high dependence between the two bottleneck resources. Consequently, the reason for the system saturation lays in an oscillatory bottleneck with interleaved saturation of database CPU and disk. A thorough log analysis reveals that if many long queries are



(a) DB CPU utilization density.



(b) DB disk utilization density.



(c) Sample time series of database resource utilization for 12,000 users.

Figure 31: Bottleneck analysis of a 1/2/1/1L RUBBoS experiment with read/write workload.

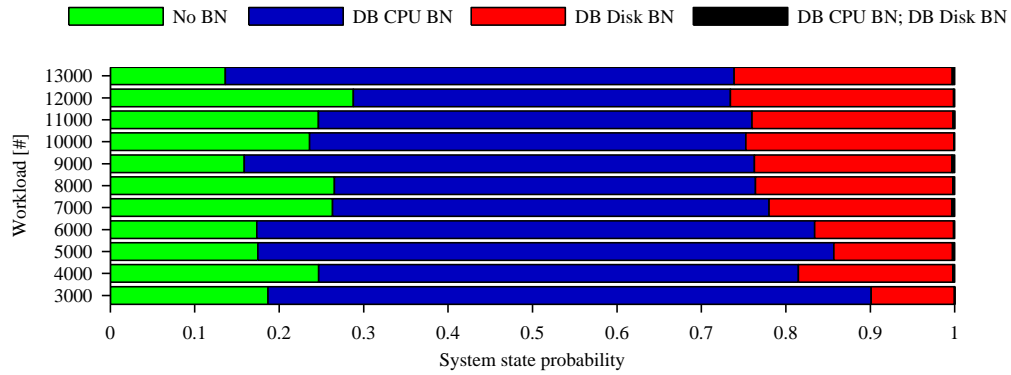


Figure 32: Probability distribution (i.e., frequency of resource saturation states) among the four different bottleneck resource saturation states in 1/2/1/1L RUBBoS experiment with read/write workload.

bottlenecked at the database disk, the CPU remains idle since it is waiting for I/O, and no new request are admitted to the node. This causes the observed exclusive saturation pattern.

6.6.2 Between-Node Dependences

The performance metrics of a 1/1/1/8L RUBBoS deployment under standard read/write workload for 1,000 to 13,000 concurrent user sessions shown in Figure 33 reveal system saturation past a workload of 4,000 users. The analysis, necessary to infer the underlying bottlenecks, is summarized in Figures 34 and 35. Figure 34(a) shows that the CPU utilization in the database does not reach critical levels during the experiment time. Consequently, this resource can be disregarded as bottleneck candidate. The inspection of the density graph for the disk utilization in the first database (representative of all eight databases) reveals slightly elevated density values at the high tail of the right-skewed density (see Figure 34(b)). Unlike the previous examples, these values do not seem to explain the overall performance deterioration at first sight. The elevation levels are constant, and their overall probability remains significant but small throughout the entire experiment. In other words, the resource shows a saturation behavior, which is very infrequent during the runtime. This could suggest a strongly oscillating bottleneck with a large number of bottleneck resources. Such an interpretation is further supported by Figure 34(c), which shows the density for the maximal

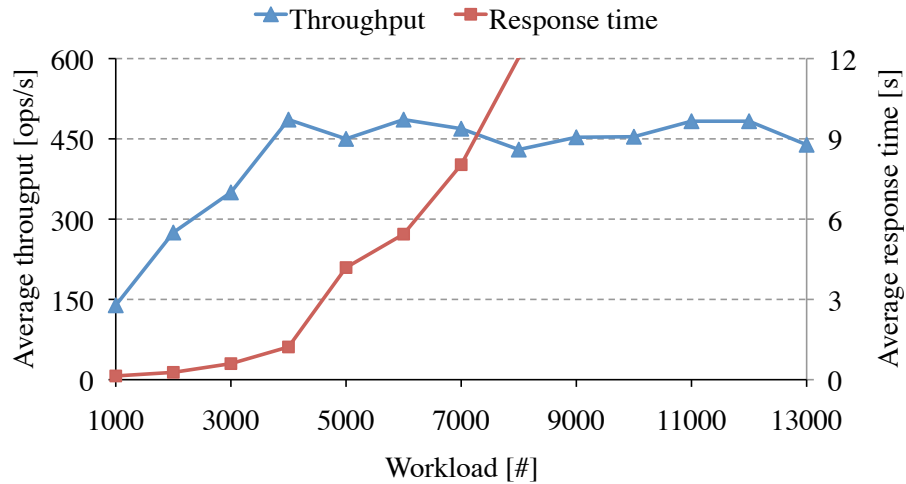
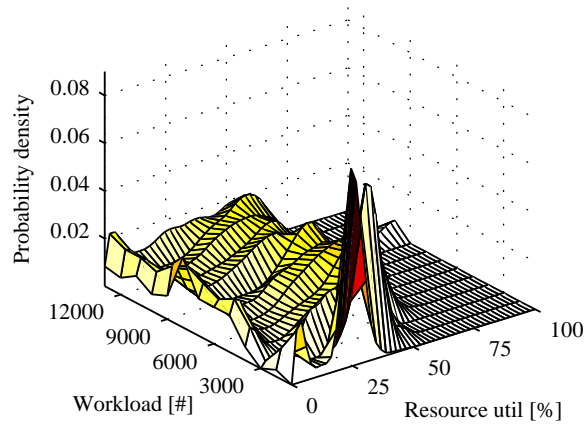


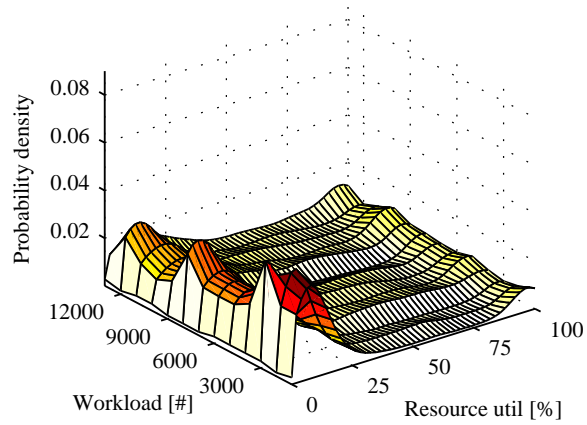
Figure 33: Average throughput and response time of a 1/1/1/8L RUBBoS experiment with read/write workload.

disk utilization among all eight databases. Past a workload of 3,000 users, the maximal value is always higher than 50 percent, and a dominant peak has appeared at the high percentiles. This means that the system exhibits a high utilization in at least one of the database disks at all times for higher workloads.

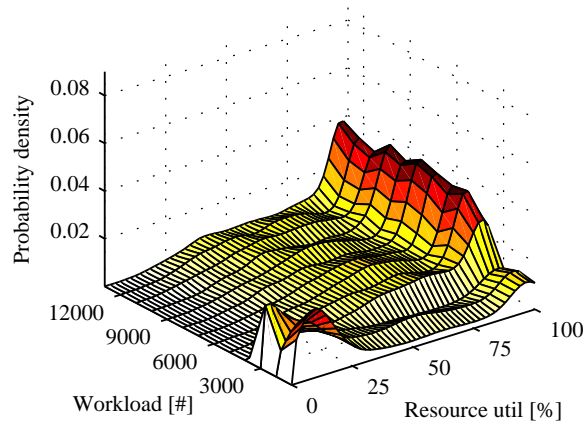
Nonetheless, the question remains whether the bottleneck resources saturate dependently. While a sample manual evaluation of the resource utilization time series (see Figure 35(a)) does not yield clear results due to the magnitude of the variability, the automated partitioning used to generate Figure 35(b) reveals the oscillatory character of this bottleneck. The probability plots are strongly dominated by two system states (i.e., single database disk bottleneck or no bottleneck at all). There is virtually no overlap between the saturation of the eight resources. The rarely observed overlap can be attributed to the noise that is introduced into the data by each resource and by the stochastic aggregation method. Therefore, we can conclude the the performance limitation in this scenario is caused by an oscillatory bottleneck with eight saturation-dependent, partially saturated database disks. A detailed log analysis shows that because of the workload distributing capabilities of C-JDBC and the relative infrequency of overly long queries, the bottleneck is strongly distributed among



(a) DB1 CPU utilization density.

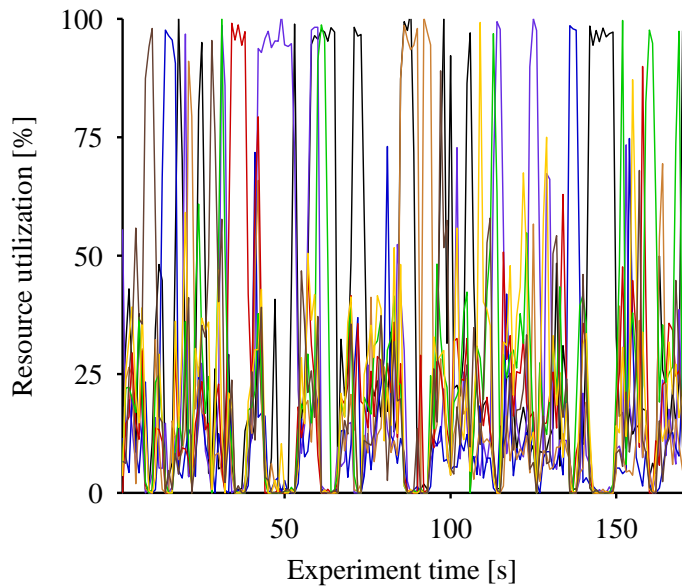


(b) DB1 disk utilization density.

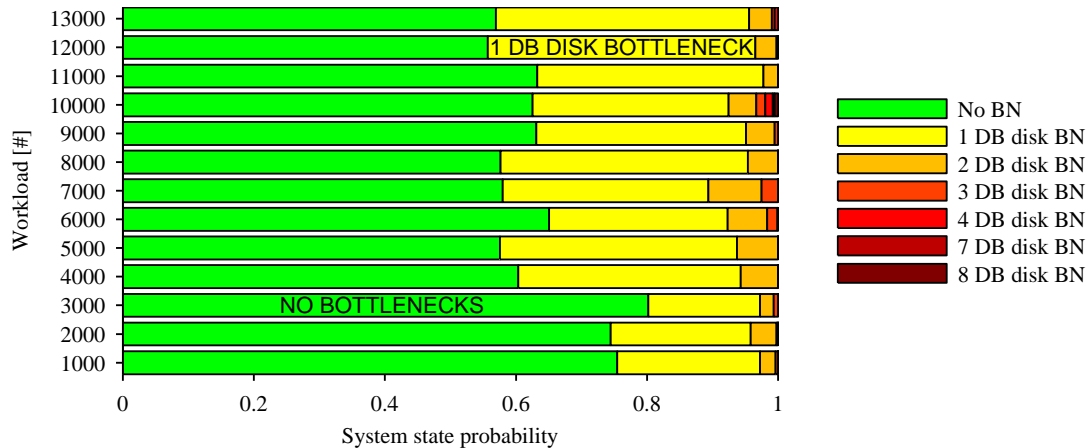


(c) Maximal DB utilization density among all eighth DBs.

Figure 34: Density analysis of 1/1/1/8L RUBBoS experiment with read/write workload.



(a) Sample time series of disk utilization in all DBs with 12,000 users.



(b) Probability distribution (i.e., frequency of resource saturation states) among the seven different bottleneck resource saturation states.

Figure 35: Detailed bottleneck analysis of a 1/1/1/8L RUBBoS experiment with read/write workload.

the eight resources. The interleaved saturation is caused by the default C-JDBC replication policy, which waits for the completion of all concurrent write-requests before committing. Writes are always immediately sent to all databases (i.e., multi-master replication), thus the entire system is bottlenecked if one database node saturates.

6.7 *Related Work*

Traditional performance analysis in computer systems presumes models based on expert knowledge, employs standard statistical methods, and parameterizes them based on a certain experimentation design [46, 58]. Queuing models have been common practice in many research efforts dealing with performance prediction [94, 98]. Although these approaches have been applied very successfully, they suffer from their rigid assumptions when handling all evolution of large applications. The availability of extensive instrumentation data profiles [94] or constant mean inter-arrival times of request [98] do not hold in general since actual parameters vary widely in real applications. The characteristic load non-stationarity in n-tier systems has been exploited for performance prediction by Stewart et al. [93], who also explain how their anomaly detection can be used to invoke a bottleneck detection process such as the one presented in this paper.

Statistically induced models have been recently used to remove human intervention from the loop [10, 31, 45], and extensive experiments have been conducted to compare different bottleneck detection methodologies [65]. Nevertheless, all these algorithmic approaches explicitly correlate high-level application performance with low-level system behavior. In this paper, we solely use performance degradation patterns as a trigger for our analysis, similarly to real system administrators [22]. In practice, many computer manuals possess a performance tuning section, which typically relies on specialized “rules of thumb”. Additionally, commercial tools (e.g., HP Open View or IBM Tivoli) offer the possibility of inspecting an abundant variety of metric data graphically without clear aggregation and analysis frameworks. Some discussion on a two-dimensional bottleneck characterization

in RUBiS and RUBBoS has been previously provided by Amza et al. [14]. However, this approach solely relies on manual human diagnosis and only targets stable bottleneck characterization with small-scale experimentation.

In contrast to our work, previous bottleneck detection research in computer systems has built upon an extremely detailed understanding of the systems (e.g., invasively instrumented central system [21]) or an analysis confined to a small resource subset (e.g., network traffic or software configurations [83]). Methods for bottleneck detection and analysis have also been also discussed in literature on simulation in areas such as industrial production [84]. The latter work emphasizes a technique for dealing with shifting bottleneck behavior based on resource utilization. The notion of ordering bottlenecks by severity with the help of resource demand distributions has been introduced by Luthi [61]. He provides proof that approximating distributions with histograms yields a higher precision than conventional methods. Unlike our observation-based work, both these approaches remain very generic without inference of domain specific phenomena.

6.8 Conclusion

For mission-critical applications such as e-commerce, n-tier systems have grown in complexity with non-stationary workloads and inter-task dependencies created by requests that are passed between the various servers. These complicating factors create *multi-bottlenecks* such as oscillatory bottlenecks (where inter-task dependencies cause the bottleneck to migrate among several resources) and concurrent bottlenecks (where multiple bottlenecks arise among several resources). Multi-bottlenecks are non-trivial to analyze, since they may escape typical assumptions made in classic performance analysis such as stable workloads and independence among tasks.

In this paper, we describe an experimental study of multi-bottlenecks using a large dataset of n-tier application benchmark data. We used techniques and tools developed for automated system management (e.g., code generation for experimental scripts) to collect

more than 200GB of measurement data on the n-tier application benchmarks RUBiS and RUBBoS. Using statistical techniques such as kernel density estimation, we show that multi-bottleneck phenomena arise naturally in sufficiently complex n-tier systems. For instance, in 72.2% of our RUBBoS experiments with low-cost database nodes (e.g., Section 6.6), we were able to identify oscillatory bottlenecks, and in 83.7% of our RUBiS experiments, we found either concurrent or simultaneous bottlenecks (e.g., Section 6.5). Furthermore, it is non-trivial to reveal multi-bottlenecks since they often happen when no single-bottleneck is visible (i.e., average resource utilization well below saturation for all resources).

CHAPTER VII

CLOUDXPLOR: A TOOL FOR CONFIGURATION PLANNING IN CLOUDS BASED ON EMPIRICAL DATA

Configuration planning for modern information systems is a highly challenging task due to the implications of various factors such as the cloud paradigm, multi-bottleneck workloads, and Green IT efforts. Nonetheless, there is currently little or no support to help decision makers find sustainable configurations that are systematically designed according to economic principles (e.g., profit maximization). This paper explicitly addresses this shortcoming and presents a novel approach to configuration planning in clouds based on empirical data. The main contribution of this paper is our unique approach to configuration planning based on an iterative and interactive data refinement process. More concretely, our methodology correlates economic goals with sound technical data to derive intuitive domain insights. We have implemented our methodology as the *CloudXplor* tool to provide a proof of concept and exemplify a concrete use case. CloudXplor, which can be modularly embedded in generic resource management frameworks, illustrates the benefits of empirical configuration planning. In general, this paper is a working example on how to navigate large quantities of technical data to provide a solid foundation for economical decisions.

7.1 Introduction

Although modern businesses are under constant market pressure to reduce the cost of their computing infrastructures, making sustainable configuration planning decisions is becoming an increasingly challenging task. Under the premise of lean and cost efficient information systems, new Information Technology (IT) trends and paradigms have led to a rapid growth of management complexity. In fact, recent research shows that there are

several developments that are of particular relevance to the area of configuration planning.

Despite massive amounts of empirical data, generated through systematic experimentation, have become readily available [62], traditional approaches to configuration planning still largely rely on analysis through analytical modeling. There is a gap between the technical potential for large-scale data generation and methodologies that are suitable for efficient data evaluation and interpretation. In parallel, enterprise-class n-tier systems with web servers, application servers, and database servers are ever-growing in economic importance, infrastructure footprint, and application complexity. Classic performance analysis methods are challenged by this growth due to bottlenecks that so far have been considered rare and unusual. Moreover, non-stationary workloads and dependencies among tasks, which are commonly encountered in modern enterprise systems, may violate popular modeling assumptions such as single bottleneck queuing networks [66]. Unlike analytical approaches, methods based on actual empirical data do not rely on such rigid assumptions and are not susceptible to the aforementioned oversimplifications.

While classic design approaches to datacenters dictate investment in hardware capable of sustaining peak workloads, service oriented approaches suggest purchasing a base infrastructure and renting the rest. This trend not only calls for decision systems with flexible cost model support but also places particular emphasis on the optimization of operational expenditures [42]. Similarly, a key concern in green datacenter management are the rising costs of operation. New approaches are required to tame the energy costs of enterprise information systems through adaptive provisioning. However, this is particularly difficult for large distributed systems with highly volatile workload processes [43]. New tools are necessary to provide decision makers with comprehensive understanding of the relationship between their computing performance landscape and their financial infrastructure constraints.

This paper addresses these developments and presents a novel approach to reliable configuration planning for clouds based on empirical data. Our approach is founded on an interactive and iterative data refinement process that enables configuration planners to

follow intuitive data aggregation steps, leading from raw data to high-level configuration planning decisions. We further introduce *CloudXplor*, which prototypically implements our configuration planning approach within a web framework accessible from any generic web browser. Through this implementation, we were able to evaluate our configuration planning functionality on a large experimental dataset that has been previously collected.

The main contribution of this paper is our unique approach to configuration planning based on data refinement. More concretely, our methodology correlates economic goals with sound technical data to interactively derive intuitive domain insights at different aggregation levels. We have implemented our methodology as *CloudXplor* to provide a proof of concept and exemplify a concrete use case. *CloudXplor*, which can be modularly embedded in generic resource management frameworks, illustrates the benefits of empirical configuration planning. In general, this paper is a working example on how to navigate large quantities of technical data to form a solid foundation of economical decisions.

The data in this paper are part of an extensive experimental dataset, which was collected using software tools for automated system management. These data may be used to predict and manage n-tier system performance and utilization, and the results in this paper suggest the need for more studies on how to effectively take advantage of such large datasets.

With the goal of identifying non-rare phenomena with potentially wide applicability, our data analysis focused on representative benchmarking configurations very similar to their default settings. Unlike traditional system tuning work, we did not attempt to tune specific software products to find “the best a product can do” settings. Nevertheless, such tuning work is an interesting area for future work, and of particular importance for applied research in industry.

The remainder of this paper is structured as follows. In Section 7.2 we provide a brief overview of background on Service Level Agreements, experimental infrastructure, and multi-bottleneck phenomena. In Section 7.3 we outline our approach to configuration planning through empirical data refinement. Section 7.4 introduces the actual implementation of

CloudXplor. In Section 7.5 we present a configuration planning case study based on actual empirical data. Related work is summarized in Section 7.6 before Section 7.7 concludes the paper.

7.2 Background

This section provides background that is of particular importance for our approach. We briefly present our view on Service Level Agreements (SLAs), experimental infrastructure for data generation, and multi-bottleneck phenomena. Readers familiar with these aspects of our work may also directly skip to the configuration planning approach in Section 7.3.

7.2.1 Service Level Agreements

CloudXplor is designed to support the process of configuration planning for IT infrastructures with an explicit focus on economic aspects. This perspective on the process demands the explicit definition of all relevant economic aspects. In order to provide an intuitive understanding, we define a *infrastructure cost* model and a *provider revenue* model that together reflect the cost and the value of the provided service. Modeling these two layers separately enables a cost-benefit analysis. Figure 36 shows the structure of the model. The profit is defined as the provider revenue of the system minus the infrastructure cost for providing the service. While it is usually easy to define a cost model for the operation of the infrastructure, the definition of a realistic provider revenue model is more complex. In the context of cloud computing, SLAs have become state-of-the-art. They define the level of service a provider guarantees to his customers. The SLA document usually contains the provider's revenue model, determining the earning of the provider for SLA compliance as well as the penalties in case of failure. In the presented scenario the provider's revenue is the sum of all earnings minus the sum of all penalties. The definition of reasonable SLAs is a non-trivial task because these agreements need to reflect economic value as well as customer service requirements. For instance, a mission-critical service should have higher earnings and penalties to set the right incentives for the provider to comply with the agreement.

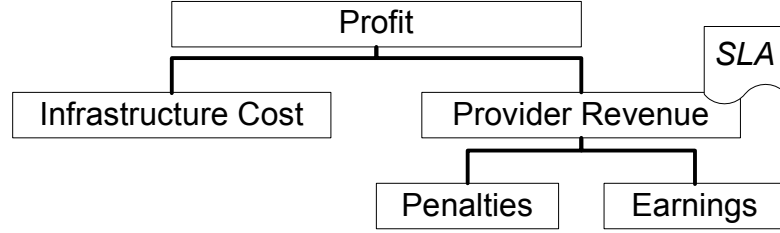


Figure 36: Profit model schema.

Furthermore, SLAs have to describe the common terms of business such as performance measures, metrics for the evaluation of the latter, legal and accounting issues, as well as exact contract periods. In the technical context of CloudXplor, the metrics for evaluating the performance characteristics of the system are of particular importance.

$$rev(rt_i) = \begin{cases} v & \text{if } 0 \leq rt_i \leq t_1 \\ v - c_1 & \text{if } t_1 < rt_i \leq t_2 \\ \vdots & \\ v - c_n & \text{if } t_n < rt_i \leq tp \\ p & \text{otherwise} \end{cases} \quad (22)$$

$$\text{provider revenue} = \sum_i rev(rt_i) \quad (23)$$

Several methods exist for the performance evaluation of system monitoring data. A common method is the definition of Service Level Objectives (SLOs) that set a maximum response time for each request. The SLO applied in the our case study (Section 7.5) is more complex and is formally defined in Equation (22). The response time rt_i of each request i is evaluated according to the following formulation. For each successfully processed request $rt_i < t_n$, the provider receives a certain earning v . With increasing response time a late charge is applied in discrete steps. If the response time exceeds t_j , then a late charge of c_j is deducted from the earning. Solely if the response time drops below a predefined threshold tp , the SLO is violated, leading to a penalty of p . The corresponding SLA in Equation (23), abstractly defines the provider revenue as the sum of all earnings and penalties for all

request. The resulting profit is defined as the revenue minus the infrastructure cost. This type of SLA supports a reliable operation of systems because the provider is motivated to provide the service at a high quality. Whenever the provider is not able to meet highest performance standards, he is incentivized to continue to provide the service as his cash flow continues to be positive. Agreements with hard thresholds might lead to a situation in which the provider reduces the priority of services that he cannot satisfy because he has to pay a penalty anyways.

7.2.2 Experimental Infrastructure

The empirical dataset that is used in this work is part of an ongoing effort for generation and analysis of performance data from cloud information systems. We have already run a very high number of experiments over a wide range of configurations and workloads in various environments, ranging from large public clouds to small private systems. A typical experimentation cycle (i.e., code generation, system deployment, benchmarking, data analysis, and reconfiguration) requires thousands of lines of code that need to be managed for each experiment. The experimental data output are system metric data points (i.e., network, disk, and CPU utilization) in addition to higher-level metrics (e.g., response times and throughput). Although the management and execution scripts contain a high degree of similarity, the differences among them are subtle and important due to the dependencies among the varying parameters. Maintaining these scripts by hand is a notoriously expensive and error-prone process.

7.2.3 Single-bottlenecks vs. Multi-bottlenecks

This subsection provides a brief overview of bottleneck phenomena and their particular implications for configuration planning. Interested readers should refer to dedicated sources [65, 66] for a more comprehensive overview.

The abstract definition of a *system bottleneck* (or bottleneck for short) corresponds to its literal meaning as the key limiting factor for achieving higher system throughput.

Consequently, this intuitive understanding has usually been consulted for analysis of bottleneck behavior in computer system performance analysis. Despite the convenience of this approach, these formulations are based on assumptions that do not necessarily hold in practice. For instance, the term bottleneck is often used synonymously with the term single-bottleneck. In a single-bottleneck case, the saturated resource typically exhibits linearly load-dependent average resource utilization that reaches 100 percent for large system loads. However, if there is more than one bottleneck resource in the system, bottleneck behavior typically changes significantly. This is the case for many real n-tier applications with heterogeneous workloads. Therefore, we explicitly distinguish between single-bottlenecks and the umbrella-term *multi-bottlenecks*.

Because system resources may be causally dependent in their usage patterns, multi-bottlenecks necessitate classification according to *resource usage dependence*. Additionally, multi-bottlenecks necessitate classification according to their *resource saturation frequency*. Resources may saturate for the entire observation period (i.e., fully saturated) or less frequently (i.e., partially saturated). Note that previous efforts in this area have typically omitted the notions of dependence and saturation frequency in their analysis (e.g., [60]). Figure 37 summarizes the classification that forms the basis of the multi-bottleneck definition as described above. It distinguishes between *simultaneous*, *concurrent*, and *oscillatory* bottlenecks. In comparison to other bottlenecks, resolving oscillatory bottlenecks is a very challenging task. Multiple resources form a combined bottleneck, which may only be addressed by considering the saturated resources in union. As a result, the addition of resources in saturated complex n-tier systems does not necessarily improve performance. In fact, determining regions of multi-bottlenecks through modeling may be an intractable problem. Consequently, multi-bottlenecks require measurement-based experimental approaches that do not oversimplify system performance in their assumptions.

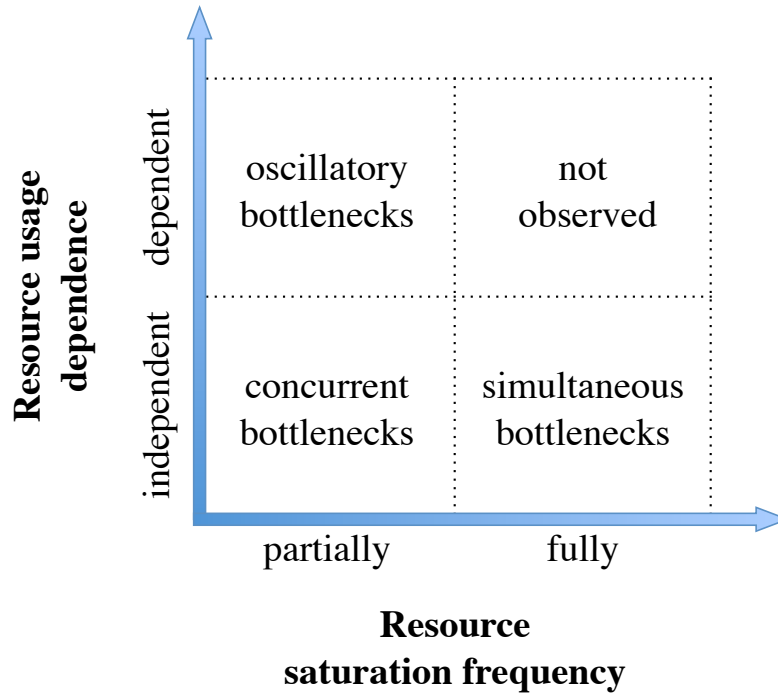


Figure 37: Simple multi-bottleneck classification.

7.3 Empirical Configuration Planning

Data refinement of experimentally derived data is a non-trivial and error prone task when done manually and without proper tooling support. In this section we introduce the interactive and iterative data refinement process that forms the basis of the configuration planning capability of the CloudXplor tool.

In general, the data refinement process for configuration planning, as depicted in Figure 38, can be divided into four distinct data analysis modules¹. Each of these modules can be characterized by a different degree of information density. The data transformation process itself is a direct result of the sequential application of various aggregation and filtering functions that are necessary to navigate, understand, and interpret the complex data space. In order to deal with the complexity and richness of the underlying data space, the transition process is multi-directional by design. Choices are made iteratively and interactively.

¹All included graphs reappear in the latter part of this paper.

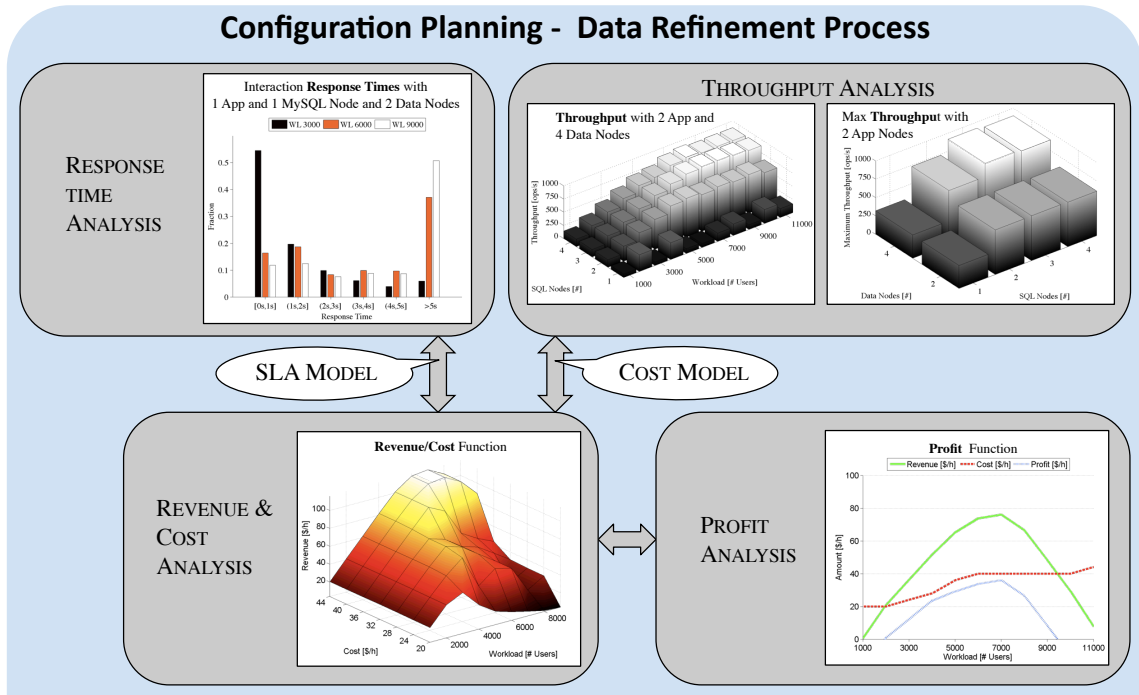


Figure 38: Schematic data refinement process in the CloudXplor Tool.

The **response time analysis** module (top left in Figure 38) allows analyzing the response time in the system. However, since the analysis of averaged values may easily lead to oversimplification, the data is aggregated in histograms that offer more detailed insights in the response time distributions. By fixing specific configurations and workloads, the user may zoom into the data and inspect response time distributions according to an a priori specified interval function. The latter is typically chosen according to an SLA function of interest. In the example graph (see Figure 43 for details), the interval function is specified as a mapping of six response time intervals, which are later mapped to specific revenue and penalty amounts as described in Section 7.2.1.

The **throughput analysis** module (top right in Figure 38) can be used to perform a throughput analysis of the system. In this stage, the data may be analyzed separately from other performance metrics, such as response time. In order to maximize the amount of displayed data, a three-dimensional graphical representation is used to allow the choice of two variable dimensions (e.g., number of SQL nodes, number of application servers, or

workload). In cases where solely configuration parameters are chosen as axes, the throughput has to be aggregated in the z-dimension (e.g., maximal throughput).

The **revenue and cost analysis** module (bottom left in Figure 38) combines the datasets from the response time and throughput analysis models and aggregates the data into a three-dimensional revenue function. This process requires the inclusion of two additional models. The response time data has to be correlated with the SLA function. This yields the revenue as illustrated on the z-axis (compare Figure 44). Additionally, the sizing information from the throughput analysis is correlated with a cost model, which yields the configuration cost as illustrated on the y-axis (compare Figure 44). In the simplest case, the cost model is a linear mapping between the hardware cost and the number of nodes in the system. In general, the revenue and cost are subject to the changing workload conditions (x-axis).

The **profit analysis** module (bottom right in Figure 38) can be used to assess the optimal workload size for the system in terms of profit. In the transition between the revenue and cost module and the profit module, the three-dimensional relationship between system load, system cost, and revenue are aggregated to investigate the dataset from an economical perspective. Economic reasoning dictates that the actual size of the chosen infrastructure is directly (and solely) implied by a profit maximization scheme. More concretely, as long as the profit is being maximized, the decision maker does not care whether he is running a large or small infrastructure. Given a certain workload, the maximal profit can be found by calculating the cost of each infrastructure and subtracting this value from the corresponding revenues. Once the infrastructure cost dimension is collapsed, the two-dimensional output can be directly used to determine the workload that yields the optimal profit for the application and system under test. The final output are revenue, cost, and profit functions. Each point on the workload span (x-axis) corresponds to an optimal configuration that is unambiguously mapped through the data aggregation in the last transition (i.e., profit maximization).

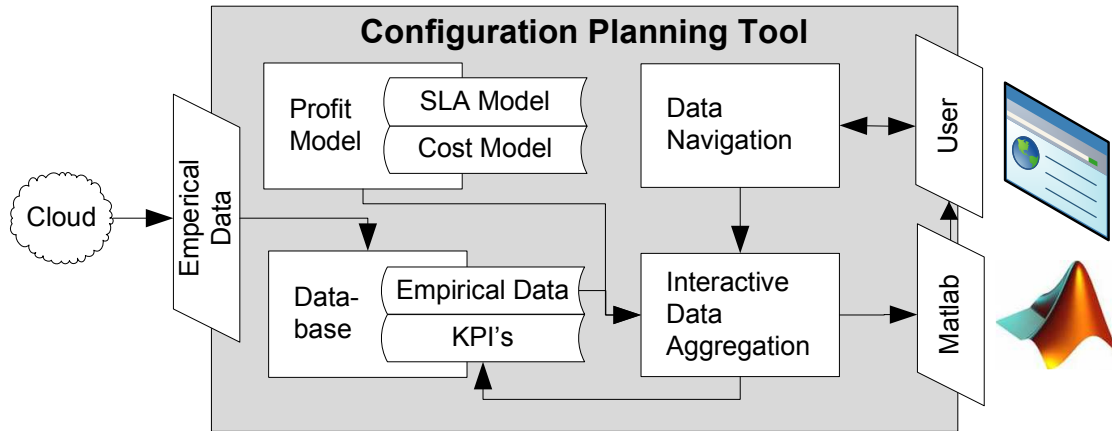


Figure 39: Schema of CloudXplor Tool.

7.4 Tool Implementation

Our configuration planning methodology has been prototypically implemented in the CloudXplor tool. This application has been developed in Microsoft Visual Studio 2008 as a light weighted web application. To enable fast and complex data processing, a MySQL server is integrated into the tool for data storage as well as an interface to remotely call Matlab (i.e., an language for technical computing) programs. Figure 39 depicts the main program structure and the tool environment. CloudXplor consists of four main units and three interfaces.

The first interface imports the empirical data, generated on an arbitrary cloud with an arbitrary benchmark. The second interface allows our tool to utilize the functionality of Matlab programs, which is our method of choice for complex calculation tasks and graphical rendering. The third interface exposes the functionality of CloudXplor to the end user. Furthermore, the tool functionality and analysis results can also be directly exported through this interface as a service. This allows the integration of CloudXplor into generic resource management frameworks, which enriches the usability of the tool in more involved settings.

The program logic of CloudXplor is divided into four units. The foundation of the data refinement process is provided by the Profit Model Unit and the Database Unit. The latter

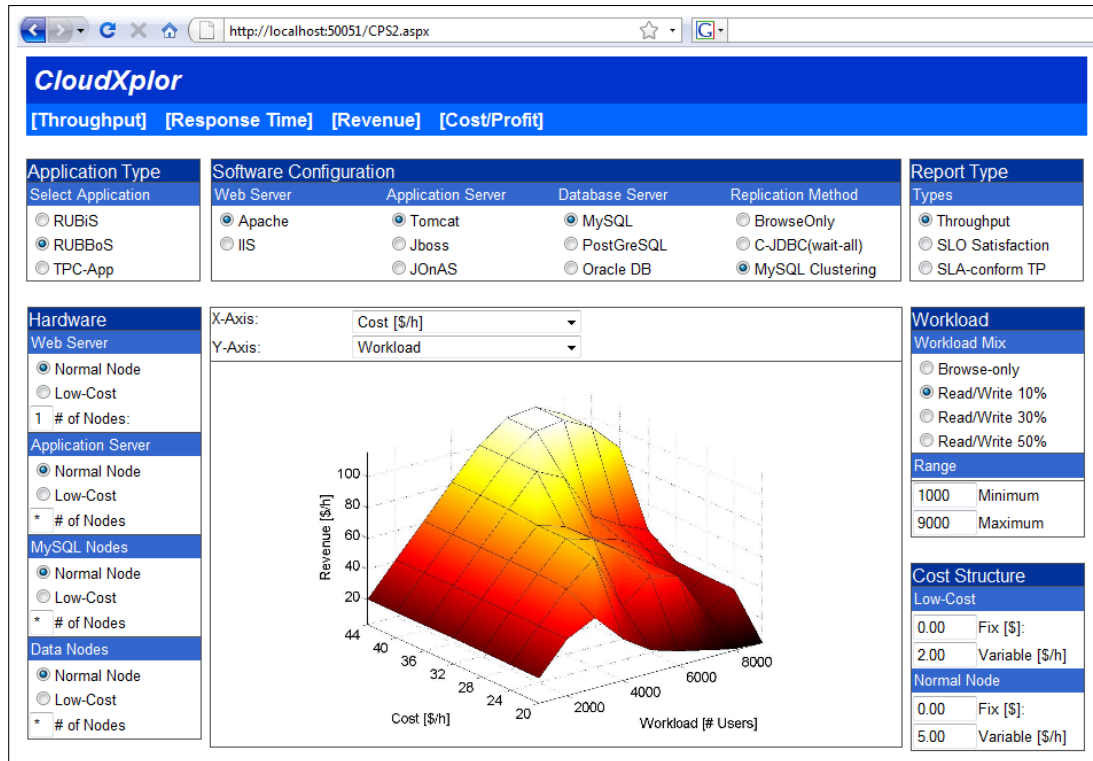


Figure 40: Screenshot of current implantation of the CloudXplor Tool.

contains the empirical data to enable fast access during the analysis. The former (i.e., the Profit Model) derives economic key figures based on the performance behavior of the system by utilizing the SLA model and the cost model. The Data Navigation Unit provides the logic for the data refinement process verifying the validity of user commands. These input parameters are sent to the Interactive Data Aggregation Unit that aggregates and correlates empirical and economic data. The results of this process are sent to the Matlab interface for further processing as well as for graphical rendering with external Matlab modules. The final analysis results are transferred to the user interface. All useful Key Performance Indicators (KPIs), generated during the refinement process, are stored in the CloudXplor database. These KPIs can be used for the comparison of different setups or accessed by other tools for further high level processing.

Figure 40 shows a CloudXplor screenshot that was taken during the execution of the data refinement process that is subject of Section 7.3. The key element of the data refinement

process is the central graph. It provides an intuitive view on system performance metrics as well as on economic data. The user can interactively change parameters to evaluate the impact of configuration changes in real-time. A variety of options is provided in the controls aligned around the graph, such as hardware and software configuration as well as economic and workload parameters. While the option fields require distinct input, the numeric input boxes can either be set to a certain value or a wild card. In the latter case, CloudXplor analyzes the scenario for each valid value of this field. The results are presented in the graph, whereby the user can assign each metric to each axis. In case multiple input parameters remain flexible, the user can specify a concrete report type. CloudXplor then uses the best setting for each flexible parameter for the graph. This allows an easy comparison of different configurations, which implies more intuitive assessment of economic impact of this parameter range.

7.5 Case Study

In this section we present a case study that exemplifies the use of CloudXplor on an actual empirical data set. We first introduce the dataset in terms of benchmark application, software, and hardware setup. In the second part of the section, we detail the output of the case study performed with our tool.


7.5.1 Setup

Among n-tier application benchmarks, the Rice University Bidding System (RUBBoS) has been used in numerous research efforts due to its real production system significance. In our experiments, each experiment run consist of an 8-minute ramp-up, a 12-minute run period, and a 30-second ramp-down. Performance measurements (e.g., response time or CPU utilization) are taken during the run using the benchmark's client generator module or Linux account logging utilities (i.e., Sysstat) with one-second intervals. Readers familiar with this benchmark can directly refer to Table 14, which outlines the concrete choices of software components used in our experiments.

Table 14: Software setup.

Function	Software
Web server	Apache 2.0.54
Application server	Apache Tomcat 5.5.17
Database server	MySQL-cluster 6.2.15
Operating system	GNU/Linux Redhat FC4 Kernel 2.6.12
System monitor	Systat 7.0.2

Table 15: Hardware setup.

Type	Components		
Normal 	Processor	Xeon 3GHz	64-bit
	Memory	2GB	
	Network	6 x 1Gbps	
	Disk	2 x 146GB	10,000rpm

RUBBoS [5] is an n-tier e-commerce system modeled on bulletin board news sites similar to Slashdot. The benchmark can be implemented as 3-tier (web server, application server, and database server) or 4-tier (with the addition of cluster middleware such as C-JDBC) systems. The benchmark places high load on the database tier. The workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. The benchmark includes two kinds of workloads: browse-only and read/write interaction mixes. Typically, the performance of benchmark application systems depends on a number of configurable settings (including software and hardware). To facilitate the interpretation of experimental results, we chose configurations close to default values. Deviations from standard hardware or software settings are spelled out when used.

The data in this section are generated from a set of experiment that were run in the Emulab testbed [3], which provides various types of servers. Table 15 contains a summary of the hardware used in this paper. Normal nodes were connected by a 1Gbps network. The experiments were carried out by allocating a dedicated physical node to each server.

7.5.2 Results

Due to the space constraints of this article, we can solely include a few sample graphs, which comprise a tiny subset of the actual navigable space of our data. More concretely, the data that is shown in this section is limited to read/write workload with ten percent write interaction frequency. Furthermore, the workload spans 1,000 to 11,000 users in steps of 1,000. The user numbers reflect generated client threads, that interact with the system based on a Markov transition probability matrix, where each state has a exponential think time with seven second mean.

Figure 41 shows the throughput of the system with one web server, two application servers, and four data node servers under read/write workload. As the workload (x-axis) increases, the system bottlenecks at a workload of 2,000 users for the configuration with a single SQL node (y-axis). This bottleneck can be resolved by adding more SQL nodes. However, once there are three SQL nodes in the system and a workload of 7,000 users is reached, the system throughput can no longer be increased through the addition of SQL nodes. This analysis yields the assumption that the system bottleneck has shifted elsewhere or has potentially become a multi-bottleneck between multiple server types.

A different type of analysis seems to offer a potential explanation for the observed system behavior. Figure 42 shows the maximal system throughput that is achievable with one web server and two application servers. In contrast to the previous graph, there are two different kinds of bottlenecks that are successfully resolved in this dataset. First, there is a SQL node bottleneck between one and two SQL nodes. Second, there is a data node bottleneck that is resolved between the configuration with two SQL nodes and two data nodes and the configuration with two SQL nodes and four data nodes. After that, the addition of another SQL nodes again increases performance to a maximal system throughput around 900 interactions per second. This analysis suggests, that the addition of further data nodes might increase the overall system throughput even further. Note that it is due to the implementation specifics of the MySQL clustering mechanism, that the number of data

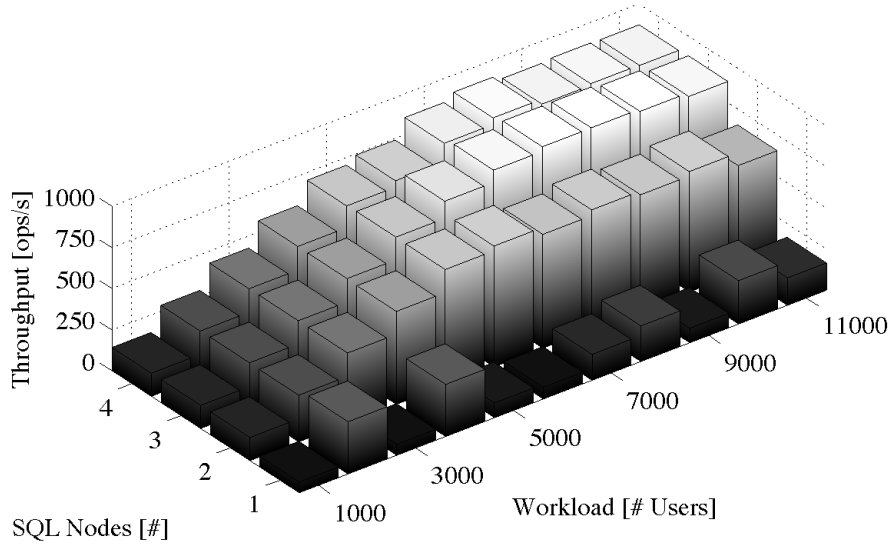


Figure 41: Throughput of a RUBBoS system with 1 web, 2 application, and 4 data nodes servers under read/write workload.

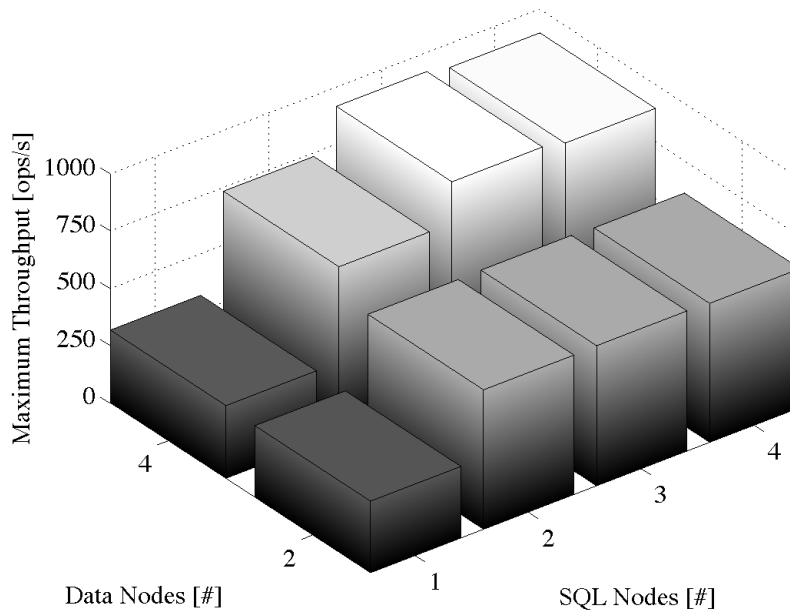


Figure 42: Maximal throughput of a RUBBoS system with 1 web and 2 application servers under read/write workload.

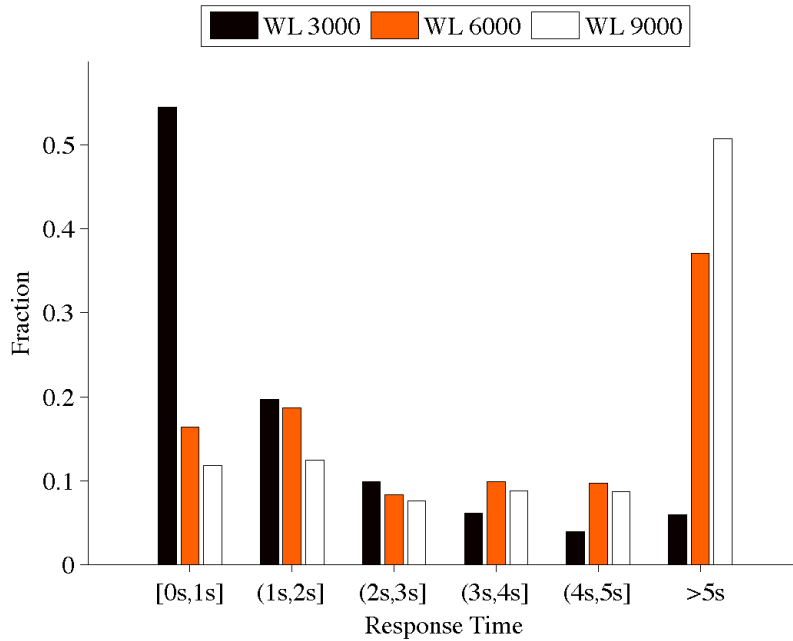


Figure 43: Response time distributions of a RUBBoS system with 1 web, 1 application, 1 MySQL, and 2 data node servers under read/write workload.

nodes may only be increased in powers of two.

In parallel to the analysis of the throughput, an analysis of the response time may be performed. Figure 43 shows three sample response time distributions. The underlying configurations for these graphs are one web server, one application server, one MySQL server, and two data node servers. The depicted workloads are between 3,000 and 9,000 users in steps of 3,000. The histogram intervals have been chosen according to the SLA model, which is assumed in this case study. This model is designed according to the formulation in Section 7.2.1 and summarized in Table 16. In this way the graph can be easily interpreted in its economical context. At a workload of 3,000 users, the system is largely able to meet SLA demands. Consequently, over 50 percent of all user interactions result in a full revenue payoff. However, as the workload increases, the (relatively small) system gets overloaded very quickly. The response time distributions become highly right-skewed with high percentages of penalized interactions. In the case of a workload of 9,000 users, over

Table 16: 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

half of all interactions are penalized as unsuccessful (i.e., response time is greater than five seconds).

After exploring throughput and response time separately, the data can be combined in a unified analysis under economical aspects. Following the data refinement process depicted in Figure 39, the response time data need to be transformed with the SLA model (Table 16), and the throughput data need to be transformed with a cost model. For simplicity, we assume that the usage cost for each server node are uniform at a price of four dollars per computing hour. Note that CloudXplor is also able to implement any arbitrary cost model through a direct mapping of each configuration to a fixed and variable cost component.

The transformation results are shown in Figure 44. Another transformation that was applied to the data is a direct result of economical reasoning. Although technically possible due to the real-system character of this investigation, decreasing revenue under constant workload and increasing resource costs has been removed through maximization. In other words, the graph has been transformed to be monotonically increasing along the y-axis (i.e., configuration cost). The analysis of the three-dimensional graph reveals two highly significant insights. First, the revenue grows evenly across all configurations for low workloads (i.e., constant slope plane for workload between 1,000 and 5,000 users). Second, the ridge of the graph runs diagonally between 2,000 users in the cheapest configuration and 7,000 users in the high-end version. Past a workload of 7,000 users, all configuration variations result in decreasing revenue due to frequent SLA violations and corresponding penalties. This means that the system under test is not able to further increase profitability

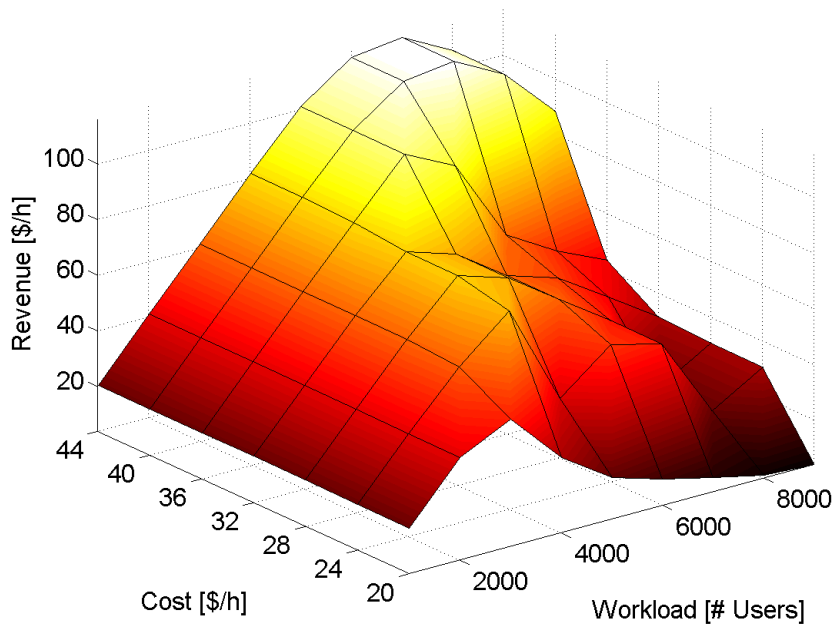


Figure 44: Revenue and cost analysis of a RUBBoS system under read/write workload.

by sustaining more than 7,000 concurrent users.

From an economic perspective, it is not significant how a particular revenue is generated as long as the profit of the enterprise is maximized. Therefore, it is beneficial to reduce the complexity of the three-dimensional data into a single two-dimensional representation. This can be done by optimizing the profit along the cost axis. The result is shown in Figure 45. This figure immediately reveals the economic impact of any arbitrary workload situation within the examined workload span. Concretely, decision makers are able to assess the profitability of their system directly from usage statistics. On the other hand, this aggregation can also be automatically resolved to correlate each workload situation with its profit maximizing cloud configuration.

The profit optimal configurations of the examined system are shown in Table 17. We use a four-digit notation #W/#A/#C/#D to denote the number of web servers, application servers, SQL nodes, and data nodes. The table shows that the data analysis process revealed a unique configuration plan that can be used to provision the system under the premise of

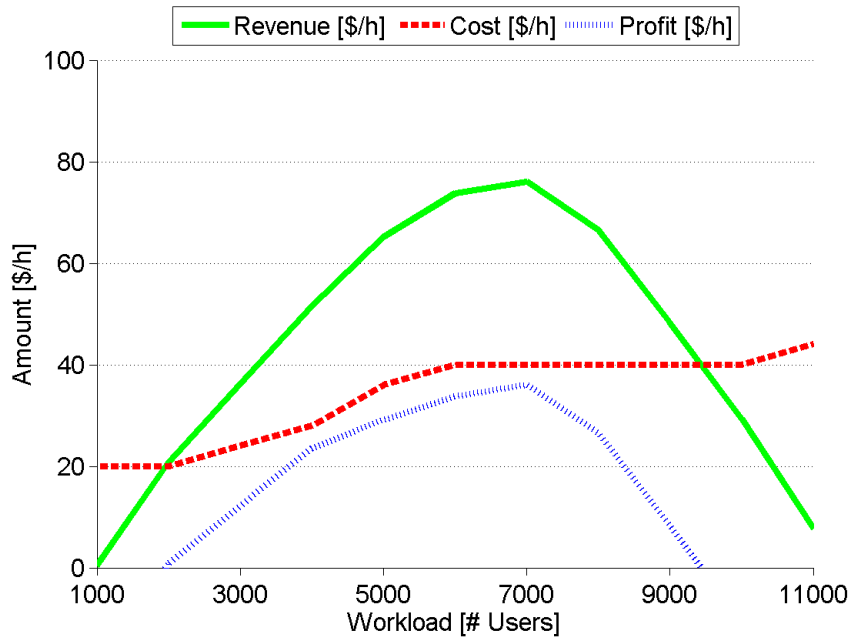


Figure 45: Profit and cost analysis of a RUBBoS system under read/write workload.

Table 17: Profit optimal configurations.

Workload [# Users]	Opt. Configuration
1,000	1/1/1/2
2,000	1/1/1/2
3,000	1/1/2/2
4,000	1/2/2/2
5,000	1/2/2/4
6,000	1/2/3/4
7,000	1/2/3/4
8,000	1/2/3/4
9,000	1/2/3/4
10,000	1/2/3/4
11,000	1/2/4/4

optimal profitability.

7.6 Related Work

Traditionally, performance analysis in IT systems postulates models based on expert knowledge and uses standard statistics to parameterize them based on some experimental dataset [46, 58]. Queuing models have been widely applied in many successful performance prediction methodologies [94, 98]. Nonetheless, these approaches often suffer from generality limitations due to their rigid assumptions when handling all evolution of large applications. For example, assuming the availability of extensive instrumentation data profiles [94] or constant mean inter-arrival times of request [98] do not hold in general because many real parameters may be subjected to high variability. Moreover, non-stationarity of workload is very common in n-tier systems. This characteristic has been exploited for performance prediction by Stewart et al. [93]. The authors also explain in their work how their anomaly detection can be used to invoke a bottleneck detection process. More recently, statistically induced models have been suggested to automate the model generation process and remove the dependence on expert knowledge [10, 31, 45]. In fact, extensive experiments have been conducted to compare different bottleneck detection methodologies [65].

In parallel to the technical complexity, economic considerations have recently become a key issue in IT system operation. This new awareness on sustainable operation modes can be seen in the emerging trend of Green IT [90]. Although this term is primarily ecologically motivated, it is also closely related to efficiency in the domain of datacenter operation. For IT operation, environmental goals are largely congruent with economic objectives because the major Green IT scope is to increasing efficiency in this case [88]. One option is the enhancement of isolated units such as power supplies [101]. Another more holistic approach is utilization optimization. For instance, virtualization and consolidation efforts are well-established concepts from this area [88]. However, these concepts may not always be applicable. Large information systems with volatile workload processes,

for example, might have too complex performance patterns, which requires a thorough an in-depth understanding of the system behavior in order to satisfy QoS [43].

The development of advanced Service Level Management (SLM) concepts, defining the terms of businesses between providers, has recently become a very active research topic [99]. Defining feasible SLAs is non-trivial for the providers because they need to supply the guaranteed QoS while operating efficiently. With the emerging trend of cloud computing, the importance of SLM has even grown further. The benefits of cloud computing in enterprises have been previously assessed in theory. Dias de Assuncao et al. investigated the potential of combining cloud and owned infrastructure resources to achieve higher performance [34]. Hedwig et al. developed a model to determine the optimal size of an enterprise system that satisfies peak demands with remote resources [42]. Both works suggest the inclusion of cloud resource into efficient production systems. Ragusa et al. developed a prototype of such a system able to automatically scale by including remote resources. Despite these efforts, Buyya et al. examined the current state-of-the-art in cloud markets concluding that today's implementations do not fulfill the requirements of modern enterprise systems [24]. Moreover, Risch and Altmann empirically verified this argument empirically verified this argument by showing that cloud computing is seldom used by enterprises [12]. One significant reason is that cloud providers usually solely guarantee best effort. More concretely, agreeing to specific SLAs bears significant economic risks due to the complexity of the underlying infrastructures.

Resource Management Systems (RMSs) are a popular concept to control decentralized environments. Nonetheless, Krauter et al. developed a taxonomy for today's RMSs showing that economic aspects are rarely sufficiently integrated [57]. CloudXplor closes this gap by correlating SLAs to technical properties and deriving their economic implications.

7.7 Conclusion

Recent research has established configuration planning of modern IT systems as particularly difficult for a number of reasons. Among others, the popularity of cloud computing and trends such as green resource management challenge current practices. In this paper we have presented a support tool to help decision makers find sustainable configurations that are systematically designed according to economic principles. Our data based approach is novel because it is founded on a unique methodology to combine economic goals with technical data. We have provided a proof of concept by implementing our methodology as the web application. The latter is modular and can be regarded as a working example on derivation of economical insights from technical data through a systematic refinement process.

Our current and future work include augmenting the CloudXplor tool (i.e., our configuration planning methodology) with a workload analysis module that is able to provide support for loading traces and time series analysis. Furthermore, we intend to extend the tool's comparison functionality to generate intuitive results with joint comparison of various hardware infrastructures.

CHAPTER VIII

AUTOMATED CONTROL FOR ELASTIC N-TIER WORKLOADS BASED ON EMPIRICAL MODELING

Elastic n-tier applications have non-stationary workloads that require adaptive control of resources allocated to them. This presents not only an opportunity in pay-as-you-use clouds, but also a challenge to dynamically allocate virtual machines appropriately. Previous approaches based on control theory, queuing networks, and machine learning work well for some situations, but each model has its own limitations due to inaccuracies in performance prediction. In this paper we propose a multi-model controller, which integrates adaptation decisions from several models, choosing the best. The focus of our work is an *empirical model*, based on detailed measurement data from previous application runs. The main advantage of the empirical model is that it returns high quality performance predictions based on measured data. For new application scenarios, we use other models or heuristics as a starting point, and all performance data are continuously incorporated into the empirical model's knowledge base. Using a prototype implementation of the multi-model controller, a cloud testbed, and an n-tier benchmark (RUBBoS), we evaluated and validated the advantages of the empirical model. For example, measured data show that it is more effective to add two nodes as a group, one for each tier, when two tiers approach saturation simultaneously.

8.1 Introduction

Efficiency is a non-trivial challenge when managing the performance of web-facing applications, such as e-commerce websites and social networks, because their workloads are characterized by sudden and periodic variations. In fact, these non-stationary workloads have peak loads several times the sustained load and are particularly difficult to forecast

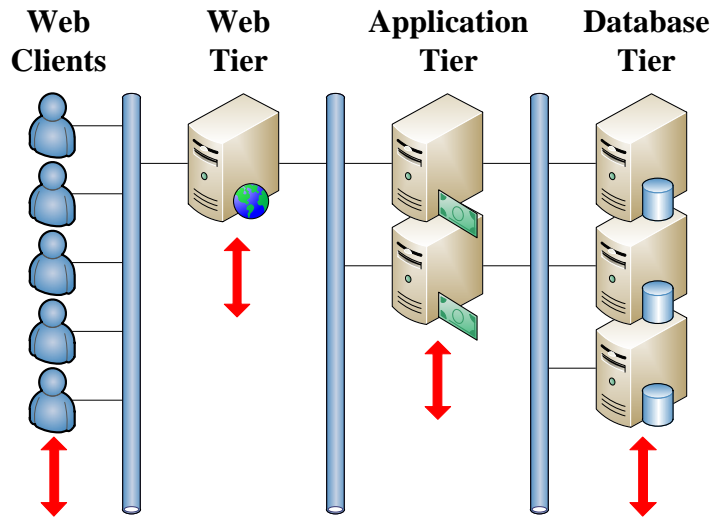


Figure 46: Elastic four-tier application.

accurately [2, 16, 36]. If these *elastic* applications are deployed in a dedicated datacenter, the service provider will face a challenging trade-off. On the one hand, satisfying QoS objectives such as short response time at peak loads will keep a large number of nodes at low utilization levels most of the time. On the other hand, provisioning mainly for the sustained load by keeping a relatively high utilization of a small number of nodes will lead to saturation and loss of business during peak loads.

Cloud environments have been touted as good solution for elastic applications because independent peak loads allow sharing of the same nodes using techniques such as consolidation; however, the relative immaturity of cloud technology renders many important technical challenges not (yet) addressed. For instance, while achieving an economical sharing of cloud resources at sustained loads *and* satisfying QoS objectives at peak loads is clearly of great interest for both cloud providers and cloud users, the necessary management automation is still an open (commercial and academic) research question. In fact, to meet this challenge, automation of adaptive resource provisioning for elastic applications is one of the major topics of interest for the autonomic computing community [59, 75, 76, 97, 106].

This paper focuses on elastic n-tier applications that are deployed on Infrastructure as a Service (IaaS) cloud environments such as Amazon EC2. A three-tiered application

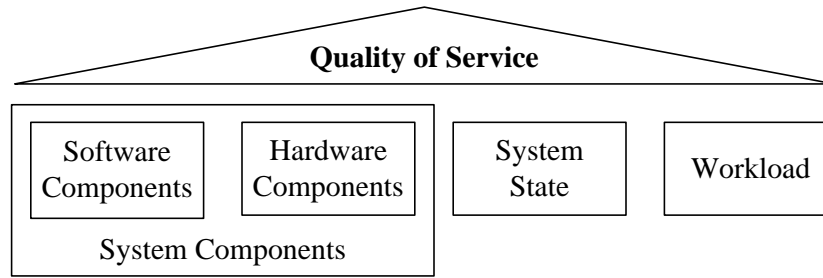


Figure 47: Major quality of service determinants.

example with web clients, web tier, application tier, and database tier is shown in Figure 46. Such n-tier applications are common because of their modular structure and potential for performance scalability by adding more servers. However, due to the dependencies among the servers, it is non-trivial to scale n-tier applications in a cloud environment [59,66,67,97]. Moreover, Figure 47 illustrates other factors that may significantly affect the system QoS, including hardware and software components (e.g., type of storage), system state (e.g., data size and distribution), and workload (e.g., workload level and mix). We call these factors QoS *determinants*. Due to the dependencies among servers, potentially inaccessible QoS determinants, and uncontrollable environment changes, continuous performance modeling of elastic n-tier applications is a complex non-linear analysis problem with multiple levels. Our approach extends automated control (e.g., [59,97]) through detailed measurement storage that captures the current dependencies among components and layers empirically in a direct mapping of QoS to workload and tunable system knobs. This facilitates automated and incremental modeling of complex performance characteristics at high accuracy and ensures continuous model evolution over time.

The conceptual contribution of this paper is a novel Integrative Adaptation Engine (IAE) architecture with a multi-model controller (Figure 48). The controller monitors an elastic n-tier application as workloads and QoS determinants vary, and decides on an appropriate allocation of Virtual Machine (VM) instances per tier, which we refer to as *configuration*. We call our controller *multi-model* because it integrates several different models to determine

“the best” adaptation action; e.g., in terms of lowest adaptation costs [49]. The major innovation of the multi-model controller is its *empirical model* that is designed as an interface to the Operational Data Store (ODS) in Figure 48. Based on queries that are triggered by the controller logic, the ODS searches through a knowledge base of previously measured operational performance data for similar configurations and workloads. If a recent match is found, the quality of the controller’s adaptation decisions should be very high (at least as high as with other a priori instantiated models) since the system performance should match the previously measured results. An important feature of the empirical model is its ability to gradually increase its prediction precision and accuracy by accumulating new measurement data over time. The main advantage of the multi-model controller is its inherent extensibility with regard to other performance prediction models (ranging from simple heuristics to sophisticated layered queuing networks) through a unified meta-model interface.

The technical contribution of this paper is an experimental evaluation of the empirical model approach, using a prototype implementation of the IAE. The evaluation demonstrates the advantages of the empirical model running an elastic n-tier application workload. The IAE prototype delivers high quality adaptation decisions based on three important elements: (1) multi-model control, (2) workload forecast, and (3) continuous learning. First, the multi-model controller maintains QoS through adaptation decisions from two models: the empirical model and an additive horizontal scale model [59] that allocates nodes one-by-one according to CPU utilization. If a match is found in the ODS, the empirical model decision is adopted; otherwise, the additive horizontal scale model is used. Second, a workload forecast model [44] predicts short-term future workload through Fourier Transformations to mitigate seasonal workload fluctuations and adaptation lead-times. Third, a learning module captures the running application’s performance data and stores them into the ODS continuously to augment the model’s knowledge base. The empirical model’s knowledge base can initially be seeded with a set of measurements from automated experiments [63,67].

Our experimental evaluation shows that the empirical model can deliver more effective adaptation actions, compared to models such as the additive horizontal scale model. For example, our measurements show that the intuitive strategy of adding/subtracting nodes one-by-one becomes sub-optimal when two mutually-dependent tiers are approaching saturation simultaneously. Adding a node to one tier requires the immediate addition of a node to the dependent tier to prevent an overall performance degradation effect. Consequently, it may be more cost effective to add those nodes as a group adaptation, instead of doing it one-by-one.

This work builds on the foundations and lessons learned from a number of our previous papers. Prior to this work, we have explored performance analysis and control from the perspectives of control theory [106], queuing theory [48, 49], automated learning [105], and empirical analysis [43, 44]. In parallel, we have experimentally analyzed cloud system scalability with a particular focus on n-tier applications [67]. Based on our findings, we have identified *multi-bottleneck* phenomena outside of classical modeling assumptions [66], which led to the development of an empirical configuration planning tool [63].

The rest of this paper is organized as follows. Section 8.2 introduces our Integrative Adaptation Engine architecture. In Section 8.3, we describe the elastic application testbed, provide IAE implementation details, and discuss our experimental results. Section 8.4 further contextualizes our work and provides an overview of related approaches. We conclude the paper in Section 8.5 with a brief summary and discussion of our findings.

8.2 *Adaptation Architecture*

The three main components of the IAE architecture are the elastic application system, the multi-model controller, and the Operational Data Store (ODS). In this context, the elastic application system comprises both the computing cloud (hardware-side) and the elastic n-tier application (software-side). Figure 48 provides a high-level overview of the topology and communication between these components.

The elastic n-tier application is hosted on VM instances that are obtained from the cloud

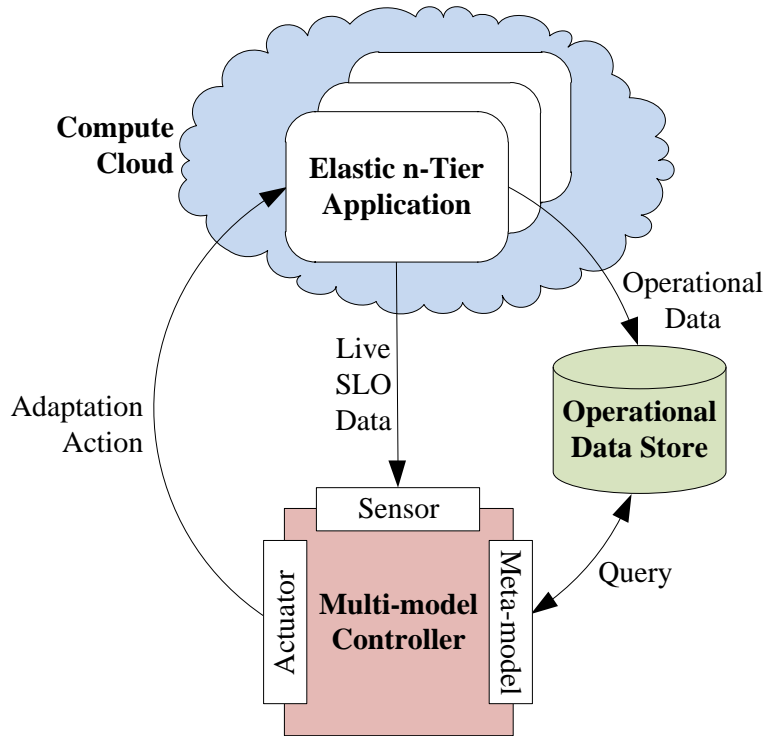


Figure 48: Integrative Adaptation Engine architecture.

infrastructure provider. VM instances are available on a commodity pricing basis (i.e., pay-as-you-use) with fine-grained rental periods. The guest application can be assumed to be a web-facing application with multiple tiers, which serve a web-interaction workload generated by an elastic number of independent clients. During operation, an n-tier application system exposes various hooks that allow the recording of rich monitoring data. Typically, the quality of the provided service (i.e., the user experience) is characterized in the form of Service Level Objectives (SLOs), which are combined into Service Level Agreements (SLAs). In our architecture the percentage of satisfied SLOs is directly reported to the adaptation controller’s sensor as a live data stream.

The persistent storage module for all available operational data such as resource utilizations, number of VM instances per tier, and interaction-specific throughput is the Operational Data Store (ODS). The ODS enables the IAE to make adaptation decisions based on previously observed application system behavior, and is the second of the three main IAE

components. The third is the multi-model controller, which interacts with the outside world through three interfaces: the sensor, the meta-model, and the actuator. In particular, the sensor interface is responsible for triggering the adaptation decision workflow in response to a live SLO input. During the decision-making process the adaptation controller uses the meta-model interface to determine the “state of the outside world”. The meta-model provides access to all previously recorded operational data in the ODS (see the following sections). If the controller determines that the elastic application system can be optimized, an appropriate adaptation action is triggered through the actuator interface. Typically, adaptation actions have to be communicated to both the cloud provider’s API as well as the elastic n-tier application’s API.

From a more general perspective, the main data-flow in Figure 48 constitutes an infinite cycle. This design facilitates the continuous extension of the state-space that is known to the empirical model. Conceptually, this design provides the potential for high modeling accuracy resulting from dynamic data evolution over time, which corresponds to continuous model evolution.

8.2.1 Meta-model

The meta-model is the the multi-model’s interface that enables the adaptation controller to access previously observed operational data in the ODS. As shown in Figure 49, the meta-model is extensible and can implement a number of sub-models. For this paper, we have restricted the meta-model to three sub-models: the Empirical Model (EM), the Workload Forecast Model (WFM), and the Horizontal Scale Model (HSM). The analysis of more complex hybrid-model solutions are beyond the scope of this paper and are left as future work.

8.2.2 Horizontal Scale Model

The purpose of the HSM is to query resource utilization values in order to determine which tier is over-utilized (or under-utilized). As previously mentioned, this type of scale decision

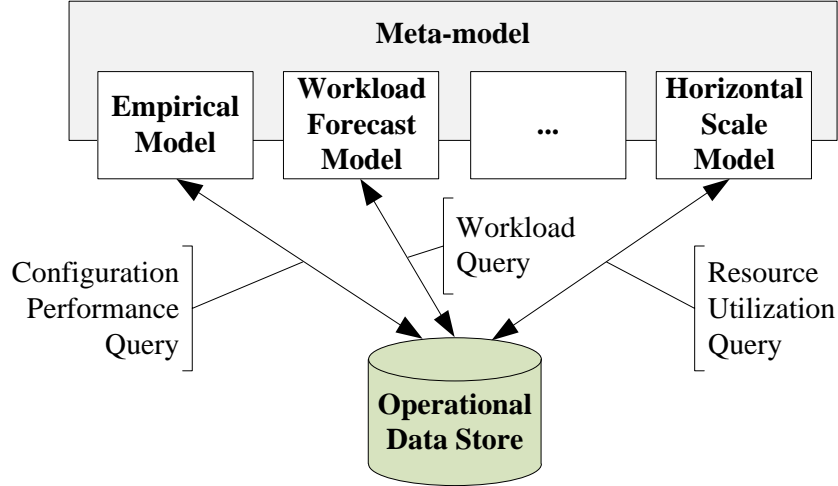


Figure 49: Extensible meta-model that implements the EM, the WFM, and the HSM with ODS access.

analysis was employed by Lim et al. in their Horizontal Scale Controller (HSC) [59]. Following Lim et al., the HSM classifies the tier with the highest (or lowest) CPU-utilization average in the past measurement interval as over-utilized (or under-utilized). In our approach, the HSM serves as an alternative decision method to the EM. As detailed in Section 8.2.5, this alternative decision method is necessary in case the queried state-space is not (sufficiently) known to the EM. In other words, the HSM compensates the main drawback of the EM, which is the dependence on previously recorded data. In combination the two models facilitate both efficient adaptation to previously encountered scenarios and adaptation to previously unseen scenarios.

8.2.3 Empirical Model

The EM queries the ODS for performance achieved by distinct configurations based on throughput vectors. Correspondingly, the performance data in the ODS are organized in a throughput vector space, which is multidimensional, application-specific, and cloud-specific. Each vector \vec{x} in the ODS summarizes a set of interaction-type throughput rates for a short time interval. Each interaction corresponds to a distinct dimension; e.g., $\vec{x} = (x_1, x_2, \dots, x_n)'$ where x_i is the throughput of interaction-type i during one specific measurement interval.

This data organization enables the EM to query the recorded data based on an anticipated throughput rate vector \vec{y} that can be derived from the currently experienced workload level and a short-term workload prediction (see Section 8.2.4). The goal of a query to the ODS is to retrieve “suitable” configurations, which are capable of sustaining the anticipated throughput \vec{y} . In this context, the suitability of a configuration can be defined as an insignificant (i.e., short) vector space distance between the predicted throughput \vec{y} and the previously recorded throughput \vec{x} .

The vector space distance for our ODS is formulated on the basis of the well-known Euclidean norm. We transform the classic Euclidean distance in order to include additional domain-specific constraints. Correspondingly, we define our own distance function (24) as a summation of function f .

$$dist(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\| = \sqrt{\sum_{i=1}^n f(x_i, y_i)} \quad (24)$$

Function f is defined in Equation (25) and uses the Euclidean metric if the recorded throughput is less than or equal to the query throughput. The Euclidean metric value is multiplied by a weighting parameter $\alpha \in [0, 1]$ in all other cases.

$$f(x_i, y_i) = \begin{cases} (x_i - y_i)^2 & \text{if } x_i \leq y_i \\ \alpha * (x_i - y_i)^2 & \text{otherwise} \end{cases} \quad (25)$$

The intuition behind this definition is to retrieve configurations that are capable of sustaining a throughput that is greater than or equal to the query throughput. Therefore, the distance should mainly increase through throughput deviations that are negative, which violates the desired performance. In the most intuitive case, weighting parameter α is equal to zero. However, it may also be desirable to extend the state-space exploration phase of the IAE. In these cases, a higher weight results in a more frequent use of alternative sub-models, which leads to a higher state-space coverage.

The EM queries to the ODS can be designed according to the following pseudo-SQL

based on the scalar stored function **DIST**(@x_vector, @y_vector), which implements the aforementioned distance metric.

```
SELECT
    MAX(time_stamp) AS last_used, AVG(sla) AS sla,
    config AS configuration_setting, cost,
    COUNT(id) AS support
FROM Operational_Data
WHERE
    DIST(@x_vector, @y_vector) < @threshold
GROUP BY
    configuration_setting
ORDER BY
    cost, last_used DESC;
```

The scalar @threshold is substituted by an appropriate distance threshold. The choice of this threshold determines the domain-specific selectivity of the EM with regard to judging deviations from previously recorded throughput values. Higher selectivity increases the chances of relying on alternative sub-models (e.g., the HSM) in the meta-model. Further selectivity constraints can be formulated on SLA-satisfaction and support in the result set. While the latter ensures that there is sufficient evidence of the suitability for a given configuration, the sla column ensures that the desired QoS was actually met. The retrieved tuples from the ODS are grouped by distinct configurations (e.g., cardinality per tier) and are ordered by the configuration cost. In horizontal scaling scenarios with uniform VM types, the cost is proportional to the total number of VM instances. The secondary ordering attribute last_used corresponds to the most recently observed configuration. This facilitates the continuous evolution of the performance model if changes occur to the underlying performance determinants. In other words, the model gives preference to the

most recently recorded data, which can be regarded as an intuitive form of data aging in a learning module.

8.2.4 Workload Forecast Model

Although workload is one of the critical QoS determinants (Figure 47), current automated adaptation systems (e.g., [59]) are often purely reactive. This may cause inefficient adaptation if the workload level is highly volatile [97] or even oscillating system size due to periodic workload fluctuations. This risk is compounded by adaptation lead-times between the request of additional resources and their availability. Therefore, it is necessary, under the aforementioned conditions, to provide the multi-model controller with a predictive understanding of the application's workload process. We approach this challenge by designing a WFM, which allows the controller to pro-actively adapt the system before actually facing a specific workload scenario.

Typical elastic applications are characterized by a large number of users that send independent requests, and by a workload that may seasonally vary by more than one order of magnitude in a single day [36]. Therefore, we have previously designed a forecast mechanism based on Fourier Transformation (FT) for this specific class of workloads [44]. More concretely, the WFM queries the ODS for historic workload data whereby the query directly aggregates the high-resolution data into a second-level time series. The main seasonal effects are determined based on FT. As FT decomposes the process into a set of trigonometric functions, each of the main factors of influence can be extrapolated. The near future behavior of the process is predicted by overlaying all significant extrapolations and determining the relative change. The predicted workload-level change is then multiplied with the current interaction-specific throughput to estimate the short-term workload development. Please refer to the cited reference for a comprehensive description of this previously published result.

8.2.5 Adaptation Workflow

The decision-making process inside the multi-model controller can be represented as a workflow graph with multiple sequential and parallel decision blocks. The more sub-models that are included in the meta-model, the more rule-based decision logic has to be added to the workflow. In the case of our initial prototype, the decision logic has to manage the dataflow from the EM, the WFM, and the HSM. The corresponding workflow graph in Figure 50 has two main execution paths. While the path on the left leads to a potential scale-down adaptation, the right path leads to a potential scale-up adaptation. The triggering of these adaptations is based on *sensitive SLA*, which sets higher QoS requirements than *normal SLA*. The definition of a sensitive metric allows initiating a scale-up adaptation before the normal SLA (i.e., the actual QoS metric) is violated, and it similarly allows initiating a scale-down adaptation only if the sensitive SLA is fully satisfied. In contrast to our approach, previous control approaches (e.g., [59]) typically only monitor QoS indirectly by assuming strong correlation between QoS and a representative resource utilization metric such as CPU utilization.

The two main execution paths in Figure 50 are analogous for both scale-up and scale-down. First, the adaptation workflow proceeds only if either of the two aforementioned sensitive SLA requirements is met. Second, a short-term workload prediction is obtained. Third, the ODS is queried in order to obtain an empirically founded adaptation action. Fourth, if the ODS query resulted in a NULL-result, the HSM is queried to make an adaptation decision exclusively based on CPU utilization. Finally, the previously determined adaptation action is initiated through the actuator interface.

8.3 Experimental Evaluation

8.3.1 Elastic Application Testbed

RUBBoS [5] is an n-tier e-commerce application modeled on bulletin board news sites similar to Slashdot. We have used the benchmark implementation with four tiers (i.e., client

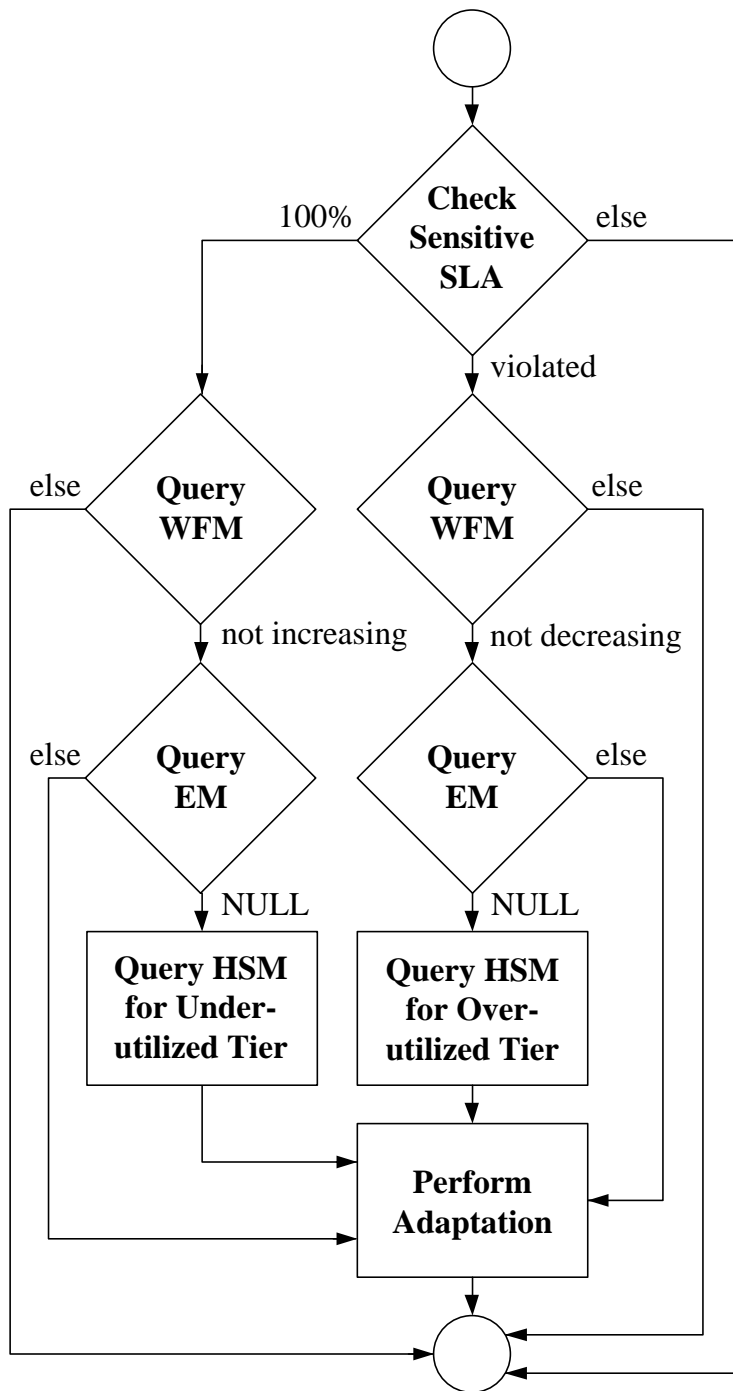


Figure 50: Decision-making workflow inside the multi-model controller.

emulator, web server, application server, and database server). In general, this application places high load on the backend. The generated workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. We ported the Java implementation of RUBBoS to the Microsoft .NET environment. We modified the benchmark to run six native RUBBoS interactions, and we implemented a seventh interaction that we call the Business Logic Request (BLR). The BLR has the properties of a typical CPU-bound application-server-intensive interaction. More concretely, complex (CPU-intensive) operations are performed in the application tier based on simple database queries with relatively small result sets. In essence, the resulting benchmark presents itself as a well-balanced system that distributes the load evenly among all three tiers¹, which is the main target environment of this paper. The .NET code was deployed in enterprise class Microsoft environment. The frontend and the application server tier were deployed on Microsoft Internet Information Services (IIS) servers. As a backend, Microsoft SQL Server was used with the schema and large dataset provided by the RUBBoS benchmark. We modified the RUBBoS benchmark workload generator to use an open model with a freely manageable number of concurrent user interactions (i.e., no client thread sleep time) [36]. The interactions are chosen based on an a priori specified probability vector. All benchmark application components were deployed on Microsoft Windows Server 2003 (SP2).

We used a custom local cluster as our cloud infrastructure. We implemented scripts that allow us to obtain and release virtual resource similar to a real IaaS cloud environment. The test cluster was built using hardware virtualized Xen 3.4.3 VM instances. The underlying operating system was Red Hat Enterprise Linux 5 (2.6.18-194.11.1.el5xen kernel) on 12 hardware nodes with 8GB of memory and an Intel Core2 QuadCPU (Q9650) with 3.00GHz each. All VMs were created with one VCPU and 2GB of memory into a pool of idle resources and ramped up upon request. In our experiment setup, we have abstracted away

¹We use a symbolic three-digit notation #Web/#App/#DB to denote the configuration; e.g., the smallest application size will be referenced as 1/1/1.

data consistency and rebalancing issues; however, state of the art solutions such as the previously mentioned Data Rebalance Controller [59] could seamlessly be included into our modular architecture.

8.3.2 IAE Implementation Details

In order to provide maximal compatibility to the elastic application testbed, the IAE prototype was implemented based on a similar software stack as the testbed. Most of the prototype code was written in the Microsoft .NET environment while the mathematical logic of the WFM sub-model uses Matlab 2010. The multi-model controller code was deployed as a service on an IIS server, and the ODS was built in a Microsoft SQL Server backend. The learning module functionality in the ODS was designed as an append-only database relation that captures application performance data in interaction-specific one-second throughput vectors. The scalar `@threshold` parameter was instantiated with 50 interactions per second. The EM query result set (see Section 8.2.3) is further processed by a second query that filters all configurations based on `support` of at least 30 distinct observations, `sla` of at least 90% on average, and by returning the cheapest configuration (i.e., addition of “**TOP 1**” to the query). Finally, the abstract `config` specification in the ODS relation was instantiated as three-integer notation that represents tier cardinality (i.e., the configuration), and the `cost` attribute was mapped as sum of all tier cardinalities.

Due to the space constraints of this paper, we set the distance function weighting parameter $\alpha = 0$ and left the analysis of convergence properties of the EM as future work. The response time thresholds for the sensitive SLA and the normal SLA were instantiated to 0.5 and 1 second, respectively, based on the characteristics of the RUBBoS benchmark. SLA-satisfaction was calculated as percentage of requests that were processed within the response time threshold during a time interval of 30 seconds (intercepted from frontend logs). Similarly, the HSM averaging interval was set to 30 seconds.

8.3.3 Experimental Results

The following experimental results are divided into three parts. First, we introduce results that were generated with a synthetic (i.e., monotonically increasing step-function) workload in order to initialize the ODS in the IAE architecture. Second, we exemplify the characteristics of the EM scaling process based on the previously collected operational data. Third, we conclude the experimental evaluation by zooming out and exposing the IAE prototype to a one-week production system workload trace that illustrates the macro-characteristics of our approach.

8.3.4 Synthetic Workload Results

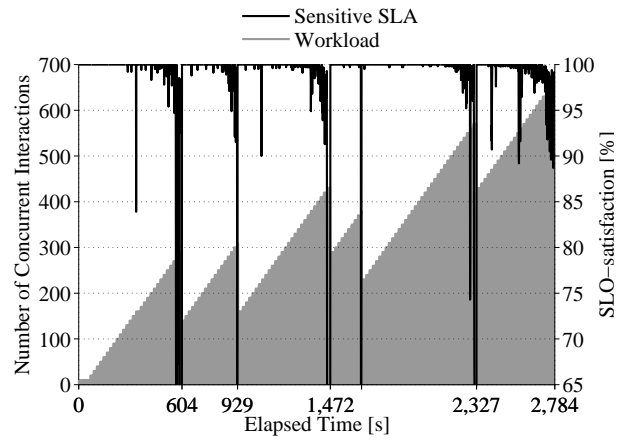
Here, we analyze results that were obtained with a monotonically increasing synthetic workload and an empty ODS. Consequently, the adaptation decisions are based on the input of the HSM. This approach serves two important purposes. First, we illustrate a state of the art control-based approach (i.e., HSC [59]) in our test environment. Second, we explore a practical strategy to fill the ODS with data and “initialize” our EM for the analysis that is shown in the following sections.

Both Figures 51 and 52 and Figures 53 and 54 are organized identically, and they depict the scale-up characteristic of the elastic n-tier application when exposed to an increasing interaction load. The difference between these two figures is that while the seven interactions are distributed uniformly in the former, the probability for Business Logic Requests (BLRs) was doubled in the latter. As discussed in the following, this change leads to a significantly higher load on the application tier, which ultimately causes different scaling characteristics. The workload traces are illustrated through the shaded area in Figures 51(a), 51(b), 53(a), and 53(b). The workload is generated according to a heuristic loop: Increment the number of concurrent interactions every 30 seconds by ten until the normal SLA-satisfaction in the last 30 seconds drops below 90 percent on average; then scale-up, drop the current workload level by 150 concurrent interactions, and restart the loop. Sensitive and normal

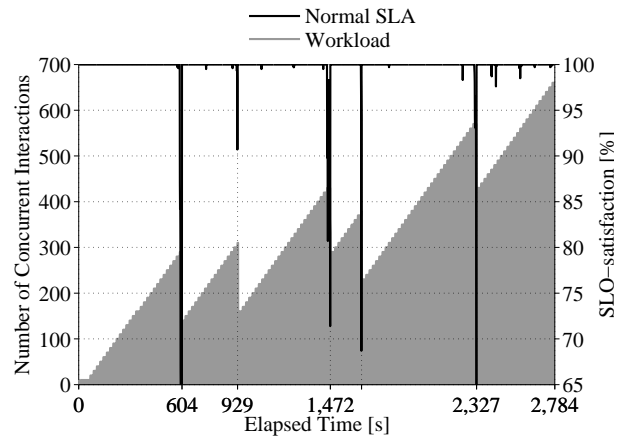
SLA-satisfaction illustrate the sensor metric and the control-target, respectively. While Figures 51(c) and 53(c) show the number of VMs in each tier of the application during the scale up process, Figures 52(a)–52(c) and 54(a)–54(c) show the corresponding average CPU utilizations for each tier. These CPU utilizations are the sole decision metric for the HSM.

The analysis of Figures 51 and 52 confirms that the application load is balanced if all seven requests are distributed equally. Figure 51(c) shows a balanced scale-up pattern with nearly equal numbers of VMs per tier. As the workload scales from 10 to about 700 concurrent interactions, the elastic application scales from 1/1/1 to 3/3/2. However, the scale-up performance (i.e., maximal throughput per scaling step) is not monotonically increasing, which may seem surprising at first. In fact, the fourth configuration step (i.e., the addition of the third application server) results in a maximal achievable throughput level (380 ops/sec with 2/3/1) that is lower than the maximal throughput level in the previous scaling step (410 ops/sec with 2/2/1). The explanation for this phenomenon lays in the balanced performance among all tiers. Concretely, the increase in capacity of the application tier conversely increases the load in the database tier because the application servers process requests faster. This increased “pressure” on the database tier results in overall system saturation at an absolutely lower workload compared to the previous configuration. In fact, the CPU utilization analysis (Figures 52(a)–54(c)) reveal that the observed bottleneck cannot be explained with full CPU utilization and is most likely due to an I/O bottleneck between the two tiers. As illustrated in Figure 51(c), the overall system performance can be significantly increased (580 ops/sec with 2/3/2) once both tiers have been scaled up, and the communication is handled by a more balanced number of servers.

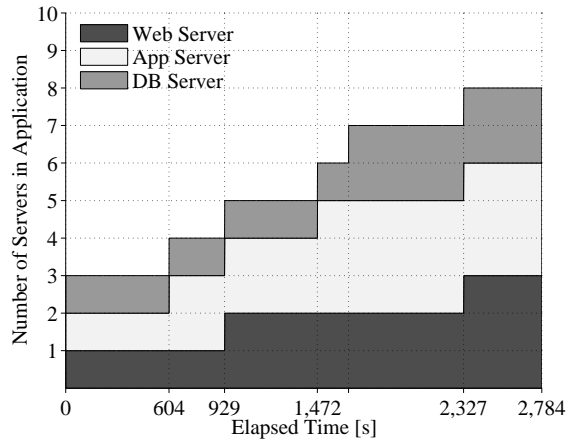
Since the probability of application tier intensive requests (i.e., BLR) has been significantly increased (from 14% to 25%) in the second scenario (Figures 53 and 54), the same synthetic workload level results in a higher number of VMs in the application tier (Figure 53(c)). As the workload scales from 10 to approximately 500 concurrent interactions, the elastic application scales from 1/1/1 to 2/5/1. The analysis of the application



(a) Sensitive SLA-satisfaction (0.5s) vs. workload.

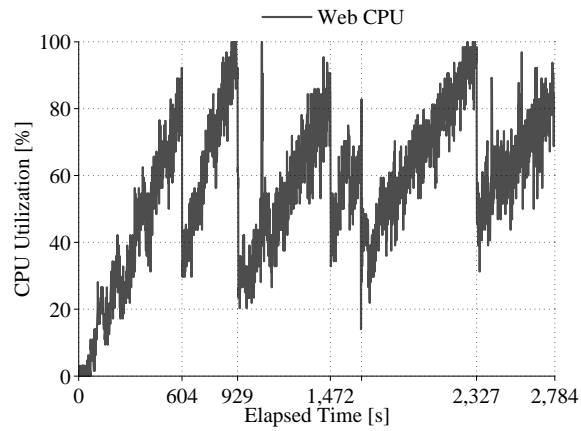


(b) Normal SLA-satisfaction (1s) vs. workload.

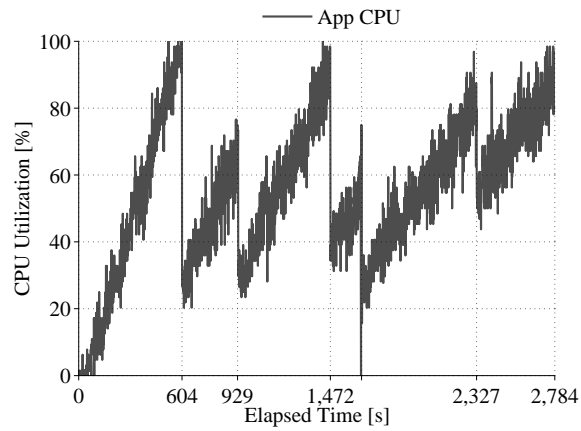


(c) Replication level of servers in application.

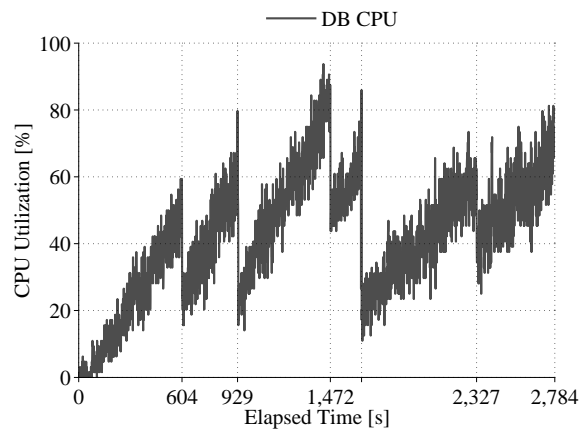
Figure 51: ODS initialization based on a synthetic workload with a uniform interaction distribution.



(a) Web server CPU utilization trace.

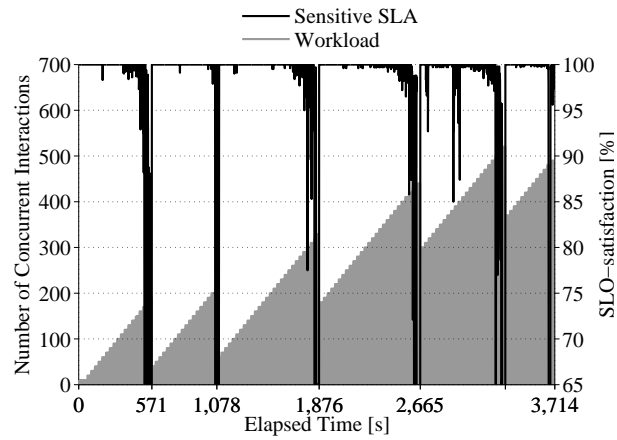


(b) Application server CPU utilization trace.

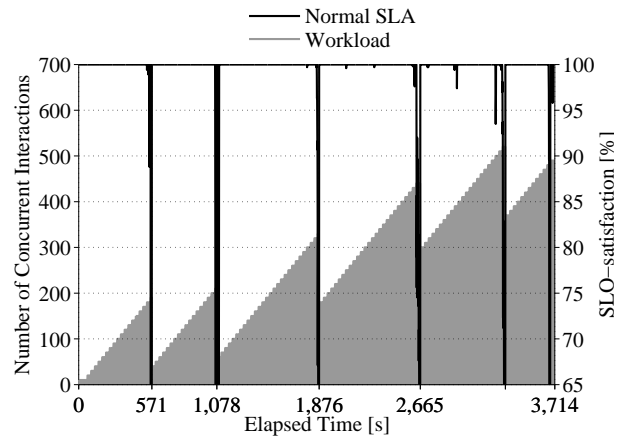


(c) Database server CPU utilization trace.

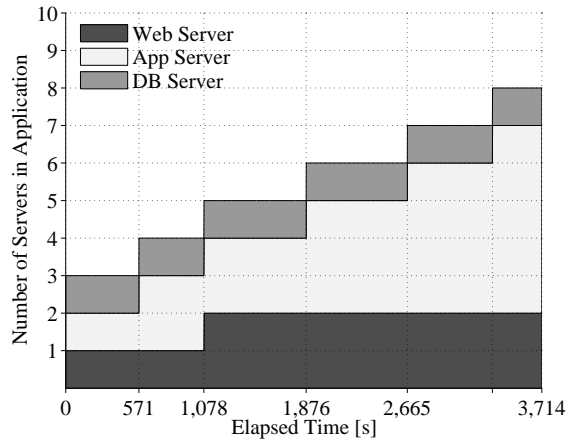
Figure 52: ODS initialization based on a synthetic workload with a uniform interaction distribution.



(a) Sensitive SLA-satisfaction (0.5s) vs. workload.

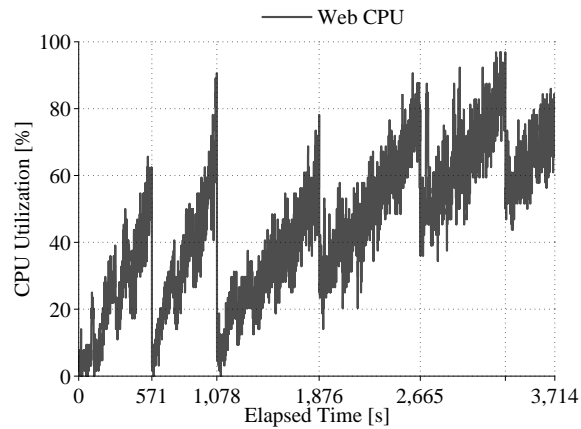


(b) Normal SLA-satisfaction (1s) vs. workload.

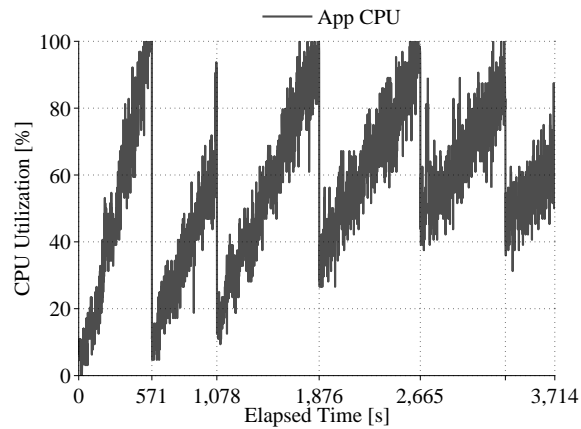


(c) Replication level of servers in application.

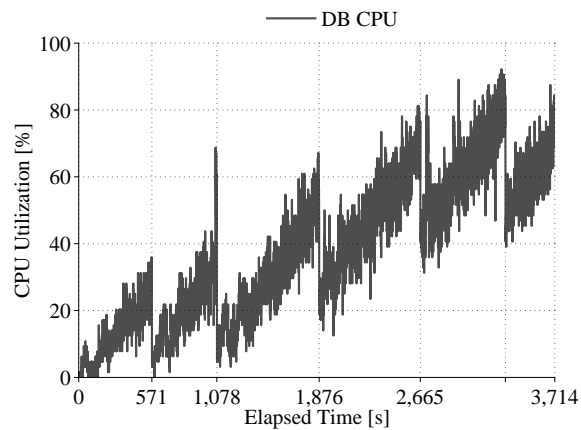
Figure 53: ODS initialization based on a synthetic workload with increased application server load.



(a) Web server CPU utilization trace.



(b) Application server CPU utilization trace.



(c) Database server CPU utilization trace.

Figure 54: ODS initialization based on a synthetic workload with increased application server load.

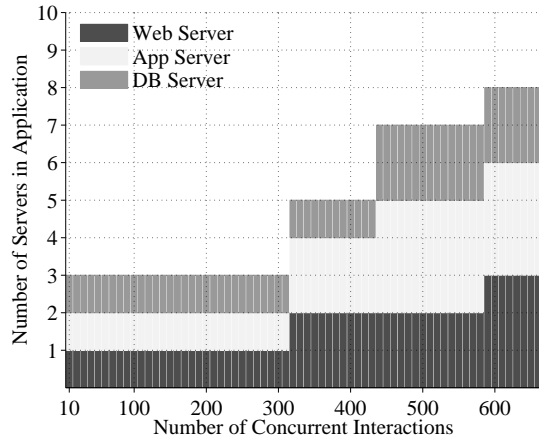


Figure 55: EM-based scaling with uniform interaction distribution (ODS initialized according to Figures 51 and 52).

tier CPU utilization in Figure 54(b) corresponds to the CPU-bound implementation of the BLR. Consequently, the HSM operates well based on the average application tier CPU utilization, which results in an approximately monotonic throughput increase per scale-up step (Figures 53(b) and 53(a)). Solely in the last scaling step (from 2/4/1 to 2/5/1), the application performance is limited by an I/O bottleneck that corresponds to the previously explained symmetric scaling phenomenon. Another important observation is the consistent relationship between sensitive and normal SLA-satisfaction. Naturally, the sensitive SLA-violations are a superset of the normal SLA-violations. This confirms the suitability of the sensitive SLA as a sensor metric for the pro-active control of the normal SLA, which is the true measure of QoS in our target environment.

8.3.5 Empirical Scaling Results

The results in this section illustrate the scaling decisions of the EM in contrast to the scaling decisions of the HSM in the previous section. Figure 55 shows that the EM is capable of making more complex scaling decisions than the HSM by scaling multiple tiers at the same time. For example, the EM automatically mitigates the aforementioned non-linear scaling characteristic and determines that it is more efficient to scale directly from 2/2/1 to

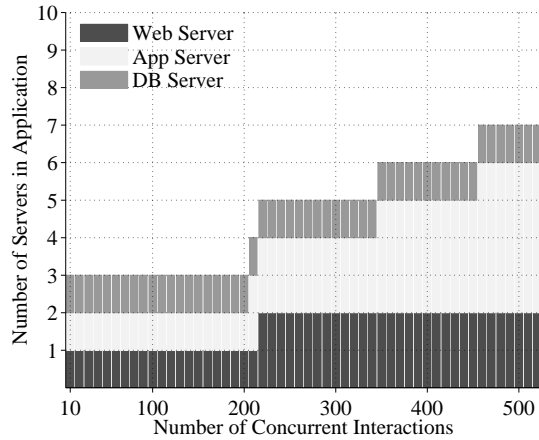


Figure 56: EM-based scaling with increased app. server load (ODS initialized according to Figures 53 and 54).

2/3/2 as the workload grows greater than 420 concurrent interactions. Another important difference between the EM and the HSM is shown in Figure 56. Despite the potentially available eighth VM, the EM does not scale beyond 2/4/1 because the previously observed 2/5/1 performance was not economical due to the I/O bottleneck.

The analysis of Figures 55 and 56 illustrates that the EM is not directly dependent on resource utilization metrics because it is able to directly map workload (x-axis) to economical configurations (y-axis). This property of the EM abstracts away the need to statically define an appropriate global resource utilization threshold such as 20% CPU utilization in Lim et al.'s paper [59]. Therefore, the EM approach enables operation at potentially higher resource utilization levels (see Figure 54(b)) as long as the SLA-satisfaction remains within desired boundaries (see Figure 53(b)).

In Figure 57 we analyze the EM with regard to a shifting workload mix. The x-axis shows the recorded ratio of BLRs and the y-axis shows the corresponding configuration at a workload greater or equal to 400. For this analysis the data in the ODS included all of the experiment runs that we have conducted to this point. From left to right, this figure illustrates the changing provisioning requirements as the bottleneck shifts from the DB tier to the application tier. Again, the EM is able to seamlessly adjust to these changing

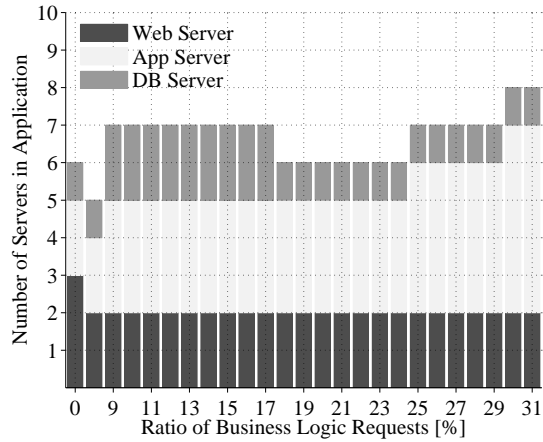


Figure 57: EM-based scaling with uniform workload and changing ratio of Business Logic Requests.

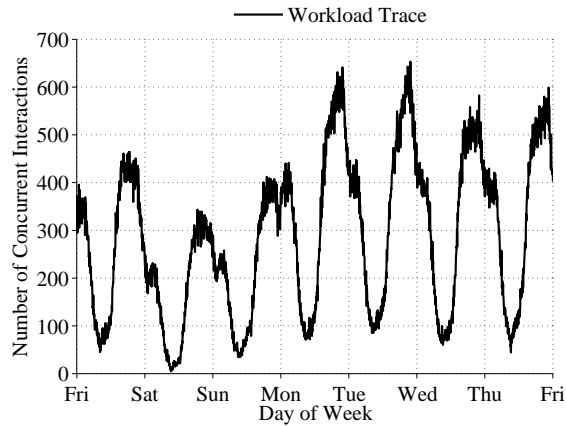


Figure 58: Assimilated Wikipedia.com workload.

conditions, irrespective of whether the change happens gradually or periodically.

8.3.6 Real Workload Results

In this section, we analyze our adaptation engine prototype when exposed to a constant mix workload (balanced interactions), which is derived from absolute user numbers that were recorded for the Wikipedia.com website in May 2008 (Figure 58). For the purpose of our analysis, the real hourly trace was transformed to span 0–700 users, spline interpolation was used to generate 30 second granularity [44], and a Gaussian noise process was injected

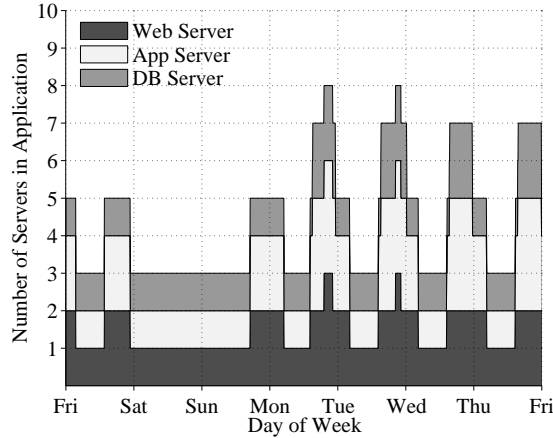


Figure 59: IAE application scaling based on the workload in Figure 58 and balanced interactions.

into the signal [36]. The analysis of Figure 59 confirms that the prototype is able to control the elastic application autonomously. It is noteworthy that low seasonal peaks such as on Sundays do not result in system scale-up beyond 1/1/1. Furthermore, most adaptation actions involve the addition of at least two VM resources in order to generate a balanced system cardinality, which corresponds to this particular workload mix scenario.

8.4 Related Work

To the best of our knowledge, there are few papers on the integration of multiple automated control models. Urgaonkar et al.’s work [97] is an example, where they combine reactive control and queuing-based prediction. Despite the small number of previous papers (some discussed below), there is little doubt that combining multiple models to overcome their individual limitations can be beneficial. More generally, although we have not found automated control mechanisms for cloud management that use empirical models directly, our work on multi-model controller and ODS can be seen as an integration (with extensions) of three component technologies: control systems, queuing models, and automated learning. We briefly outline some relevant work from each area in this section.

Another paper that includes a multi-model controller uses an integral control technique

called proportional thresholding for automated elastic storage in clouds [59]. Their scaling of the storage service is managed through a meta-controller combining a control system-based data rebalance controller and a horizontal scale controller. Their paper makes some simplifying assumptions, e.g., storage service QoS depends only on CPU utilization as the sole control metric. They also set the target CPU utilization of 20%, which may be reasonable for storage service, but is relatively low for cloud node utilization. Other control system-based approaches differ from our work because they assume the availability of continuously scaling resources in non-distributed environments [75, 76].

A well-established approach to performance prediction of computer systems is queuing models. Due to typical assumptions made (e.g., stationary arrival rates and workload-independent utilization ratios among system modules) to facilitate analytical solutions [46], classic queuing models have limitations with elastic workloads on n-tier applications [48, 93, 98]. Several research efforts have attempted to address these limitations (e.g., Mi et al.'s work on modeling burstiness in n-tier applications [71, 72], Urgaonkar et al.'s predictive and reactive provisioning, and Thereska et al.'s IRONModel [95]). As an example, the IRONModel is a robust queuing modeling architecture based on principles such as construction during system design and continuous reevaluation of accuracy during operation. Nonetheless, queuing theory requires a priori modeling of all performance relevant queuing stations and interaction types. This is not required by an empirical model such as ours, but our multi-model controller can integrate queuing models to handle the more stable workload periods.

Our empirical model includes an automated learning module that continuously inserts measured data into the ODS. This is different from machine learning-based approaches (e.g., Cohen et al. [31, 32] and Sahai et al. [87]), since we use the measurement data without necessarily finding a pattern or analytical representation. More recently, Armbrust et al. introduced SCADS [18] that uses machine learning for elastic storage system scaling. Our empirical model avoids the typical difficulties of machine learning with data noise-sensitivity

and over-fitting since we use the measured data directly.

8.5 Conclusion

We described a multi-model controller for appropriately mapping virtual machine nodes to elastic n-tier applications in clouds. The main advantage of the controller is its support for an empirical model that provides assured adaptation decisions using measured performance data from previous runs of the application. An automated learning module continuously augments the empirical model knowledge base with measured data from new application runs. For configurations not yet covered by measurements, a simple heuristic model provides approximate solutions.

We implemented a prototype of the multi-model controller with the empirical model on a database platform (called ODS). The experimental evaluation of our prototype implementation suggests that the empirical model is able to adapt to changing performance determinants and to maintain constantly high SLA-satisfaction with guaranteed performance predictions based on measured data. As a concrete example, the empirical model demonstrated adaptation cost savings over the heuristic model by suggesting a multi-node adaptation action when multiple tiers approach saturation simultaneously. Our positive initial results suggest that an empirical model can be integrated with other adaptation algorithms (e.g., [48,49]) using the multi-model controller, to achieve “the best adaptation of all models”.

Although this paper focuses on horizontal scaling in IaaS clouds, our work is also applicable to other cloud service offerings. In the case of Software as a Service (SaaS), for example, our work can help alleviate the management problems faced by the cloud provider. In the case of vertical scaling scenarios, the configuration specification in the data store can be extended to include the number of VM instances, their type, and other descriptive information to support finer granularity adaptation.

CHAPTER IX

CONCLUSION

In this dissertation I argued that the multi-level complexity of automated performance management for elastic n-tier applications in computing clouds can be successfully addressed through a systematic empirical approach—embedded into a framework of three incremental building blocks. First, I provided a proof of concept for the feasibility of large-scale experimental system performance analyses, and identified several complex performance phenomena that may escape classical system performance analysis methodology (Chapters 3 and 4). Second, I built on these initial findings and provided a proof of concept for formulating performance phenomena detection based on large-scale experimental data, and I abstractly defined multi-bottlenecks (Chapters 5 and 6). Third, as a proof of concept for automated online performance management based on empirical data, I implemented the CloudXplor tool and built a system that leverages the IAE infrastructure to automatically manage elastic n-tier application performance efficiently with respect to the previously abstracted complex performance phenomena (Chapters 7 and 8).

The main technical contributions of my dissertation research can be arranged according to the previously introduced three-building-block framework (see Figure 3) to present strong evidence in support of my thesis. These technical contributions and their relevance within the context of this dissertation are spelled out explicitly in the following itemized list.

Experimental Analysis of Elastic System Scalability and Consolidation:

- ★ My study of scale-up and scale-out scenarios with up to nine replicated and / or partitioned database servers on dedicated hardware nodes [62, 67] presented a proof of concept for large-scale experimental system performance studies.
- ★ The analyzed multi-bottleneck phenomena in n-tier applications such as oscillating resource saturation [62, 67] illustrated the opportunities in the discovery of complex performance phenomena in empirical data generated through scalability experiments.
- ★ The analyzed non-monotonic response time variations in co-located n-tier application systems [68] illustrated the opportunities in the discovery of complex performance phenomena in empirical data generated through consolidation experiments.

Modeling and Detection of Non-trivial Performance Phenomena in Elastic Systems:

- ★ My bottleneck detection approach for n-tier applications using statistical intervention modeling [65] provided a proof of concept for formulating successful performance phenomena detection based on large-scale experimental data.
- ★ My multi-bottleneck classification [66] illustrated the abstraction of complex performance phenomena based on experimental scalability evaluation.

Automated Control and Configuration Planning of Elastic Systems:

- ★ CloudXplor [63], my tool for configuration planning in clouds through iterative refinement of empirical performance data, provided a concrete proof of concept for offline cloud configuration management based on large-scale data.
- ★ My implementation of the IAE architecture and the multi-model controller [64] for managing elastic n-tier applications in clouds based on the empirical model provided a concrete proof of concept for automated online performance management for elastic n-tier cloud applications based on empirical data.

While the concrete technical contributions made by this dissertation can be regarded as the first step towards a new empirical system analysis and system performance management methodology for cloud systems, this dissertation makes an important meta-level contribution. In fact, by explicitly and concretely following an incremental research framework this dissertation paves the way for promising and truly innovative follow-up work. This point can be illustrated based on the following examples of potential follow-up research, which are a direct consequence of this dissertation.

My bottleneck-analysis work directly leads to the challenge of designing a generic bottleneck detection method that is conceptually capable of handling any kind of multi-resource saturation phenomenon. More concretely, given the practically infinite hardware availability in a cloud computing environment, all single hardware resource bottlenecks can be easily solved (in principle) by adding more hardware. A natural and important research question is: If hardware will never be a bottleneck, where and what will the bottleneck be? Therefore, I propose to focus on the next performance limiting factors after removing all single hardware resource bottlenecks. My research results suggest that for applications with dependencies among their components, multi-bottlenecks will be the next limiting factor in overall system performance. While I anticipate that the search for the next limiting factors may produce unexpected discoveries, which will superset multi-bottlenecks as limiting factors in cloud application scalability, multi-bottlenecks—as defined by this dissertation—are the first concrete starting point in this line of research.

Another consequence of my work is the challenge to comprehensively investigate the discrepancies between theory and practice of consolidation. Concretely, I propose to continue to examine the validity of the common assumption made in state of the art consolidation research papers that can be called the *linear consolidability hypothesis*—in analogy to the linear scalability of applications running on multi-processors. Under the linear consolidability hypothesis, the estimated consolidated throughput of a set of applications is a simple sum of the measured throughput of each application when running

alone. Similarly, the consolidated resource utilization of the set is the sum of the individual resource utilizations. In this case, the system phase transition into a saturation state happens when a critical resource (e.g., CPU) reaches saturation by summing up to 100% utilization. At first glance, linear consolidability appears to be obvious; therefore, it has been applied widely without being called into question.

As a first step in this line of research, I suggest to experimentally evaluate the linear consolidability hypothesis to show the non-trivial and often non-obvious limitations of its validity, particularly and importantly, with respect to the onset of system state phase transitions. In fact, a central contribution of this work should be a conceptual model of the performance loss phenomenon during consolidation. This phenomenon could be termed *performance friction*—in analogy to physical friction that causes dissipation of energy without generating work. In application consolidation, performance friction causes dissipation of resources that leads to limited overall performance. Under ideal conditions (low system resource utilization), linear consolidation can be achieved and computational friction equals zero. As system utilization increases, a decreased performance can be observed in consolidated configurations in which performance friction becomes positive. While the rise of performance friction depends on several factors that remain to be analyzed more deeply in the future, the abstract concept of performance friction would allow for such an analysis with several concrete, but disparate causes.

As a second step in this line of research, I suggest to argue that *performance friction is a feature, not a bug*. Because performance friction reduces overall performance, it is tempting to think of it as a bug. More concretely, bugs ought to be fixed if one can find their causes and sources in order to restore the “expected” overall performance (under linear consolidability). Given the variety of scenarios where performance friction can be observed, I would argue that performance friction is a highly variable phenomenon, depending on both the platform (e.g., hardware and software resource utilization levels) and the applications involved (e.g., specific combination of co-located applications) [54, 82].

And while a classical approach (in mainframe-based datacenters) would have been the tuning of hardware and software resources to maximize the performance of stable applications and workloads [89], the underlying hardware in cloud environments is typically chosen for its good cost / performance ratio, which makes tuning or balancing through design a nonviable solution.

REFERENCES

- [1] “Slony-I.” <http://www.slony.info>.
- [2] “Animoto’s Facebook Scale-up.” <http://blog.rightscale.com/2008/04/23/animoto-facebook-scale-up/>.
- [3] “Emulab: Network Emulation Testbed.” <http://www.emulab.net>.
- [4] “MySQL Cluster.” <http://www.mysql.com/products/database/cluster/>.
- [5] “RUBBoS: Rice University Bulletin Board System.” <http://jmob.objectweb.org/rubbos.html>.
- [6] “RUBiS: Rice University Bidding System.” <http://rubis.objectweb.org/>.
- [7] “The Elba Project.” <http://www.cc.gatech.edu/systems/projects/Elba/>.
- [8] “Fujitsu SysViz: System Visualization.” “<http://www.google.com/patents/US7873594>”, 2010.
- [9] AGGAR, M., “The IT energy efficiency imperative,” white paper, Microsoft Corporation, June 2011.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] AMZA, C., COX, A. L., and ZWAENPOEL, 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.
- [14] AMZA, C., CECCHET, E., CH, A., COX, A. L., ELNIKETY, S., GIL, R., MARGUERITE, J., RAJAMANI, K., and ZWAENPOEL, W., “Bottleneck characterization of dynamic web site benchmarks.” http://www.cs.rice.edu/CS/Systems/DynaServer/dyna_bottleneck.pdf, 2002.

- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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*, www.cidrdb.org, 2009.
- [19] BALBO, G. and SERAZZI, G., “Asymptotic analysis of multiclass closed queueing networks: multiple bottlenecks,” *Perform. Eval.*, vol. 30, no. 3, pp. 115–152, 1997.
- [20] 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.
- [21] BLAKE, R. and BREESE, J. S., “Automatic bottleneck detection.” <ftp://ftp.research.microsoft.com/pub/tr/tr-95-10.ps>, 1995.
- [22] 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.
- [23] BROCKWELL, P. and DAVIS, R., *Introduction to time series and forecasting*. Springer texts in statistics, Springer, 2002.
- [24] 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.
- [25] 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.
- [26] 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.

- [27] CECCHET, E., CANDEA, G., and AILAMAKI, A., “Middleware-based Database Replication: The Gaps Between Theory and Practice,” in *ACM SIGMOD’08*, 2008.
- [28] CECCHET, E., CH, A., ELNIKETY, S., MARGUERITE, J., and ZWAENEPOEL, W., “Performance comparison of middleware architectures for generating dynamic web content,” in *In Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, 2003.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] FEITELSON, D., “Workload modeling for computer systems performance evaluation.” <http://www.cs.huji.ac.il/~feit/wlmod/>, 2011.
- [37] 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.

- [38] 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.
- [39] 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.
- [40] GUSTAFSSON, F., *Adaptive Filtering and Change Detection*. John Wiley & Sons, Inc., 2001.
- [41] HANSON, B. E., “Bandwidth selection for nonparametric distribution estimation.” <http://www.ssc.wisc.edu/~bhansen/papers/wp.htm>, May 2004.
- [42] HEDWIG, M., MALKOWSKI, S., BODENSTEIN, C., and NEUMANN, D., “Datacenter investment support system (DAISY),” in *HICSS ’10: Proceedings of the 43rd Hawaii International Conference on System Sciences*, (Washington, DC, USA), IEEE Computer Society, 2009.
- [43] 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.
- [44] HEDWIG, M., MALKOWSKI, S., and NEUMANN, D., “Towards autonomic cost-aware allocation of cloud resources,” in *ICIS ’10: International Conference on Information Systems*, 2010.
- [45] HUANG, C., COHEN, I., SYMONS, J., and ABDELZAHER, T., “Achieving scable autoamted diagnosis of distributed systems performance problems,” tech. rep., HP Labs, 2007.
- [46] 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.
- [47] 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 *Cloud Computing (CLOUD), 2011 IEEE 4th International Conference on*, 2011.
- [48] 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.
- [49] 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.

- [50] 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.
- [51] 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.
- [52] 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.
- [53] 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.
- [54] 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.
- [55] 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.
- [56] 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.
- [57] 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.
- [58] LILJA, D. J., *Measuring Computer Performance - A Practitioner's Guide*. New York, NY, USA: Cambridge University Press, 2000.
- [59] 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.
- [60] LITOIU, M., “A performance analysis method for autonomic computing systems,” *ACM Trans. Auton. Adapt. Syst.*, vol. 2, March 2007.

- [61] 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.
- [62] 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.
- [63] 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.
- [64] 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.
- [65] 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.
- [66] 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.
- [67] 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.
- [68] MALKOWSKI, S., KANEMASA, Y., CHEN, H., YAMAMOTO, M., JAYASINGHE, Q. W. D., PU, C., and KAWABA, M., “Variations in performance and scalability when migrating n-tier applications to different clouds,” in *Submitted at CLOUD, 2012 IEEE 5th International Conference on*, 2012.
- [69] 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.
- [70] MI, N., “Performance impacts of autocorrelated flows in multi-tiered systems,” *SIG-METRICS Performance Evaluation Review*, vol. 35, no. 3, pp. 44–45, 2007.

- [71] 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.
- [72] 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.
- [73] 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.
- [74] 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.
- [75] 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.
- [76] 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.
- [77] 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.
- [78] 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.
- [79] 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.
- [80] 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.

- [81] 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.
- [82] 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.
- [83] 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.
- [84] 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.
- [85] 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.
- [86] 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.
- [87] 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.
- [88] SCHULZ, G., *The Green and Virtual Data Center*. Boston, MA, USA: Auerbach Publications, 2009.
- [89] SHASHA, D. E., *Database tuning: a principled approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992.
- [90] SINGH, A., HAYWARD, B., and ANDERSON, D., “Green IT takes center stage.” Springboard Research, 2007.
- [91] SNYDER, B., “Server virtualization has stalled, despite the hype,” *InfoWorld*, December 2010.
- [92] 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.
- [93] STEWART, C., KELLY, T., and ZHANG, A., “Exploiting nonstationarity for performance prediction,” *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 31–44, 2007.
- [94] 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.

- [95] 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.
- [96] 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.
- [97] 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.
- [98] 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.
- [99] 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.
- [100] 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.
- [101] VELTE, T., VELTE, A., and ELSERPETER, 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.
- [102] 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 *25th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, May 2011.
- [103] 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.
- [104] WU, C., HAMADA, M., and WU, C., *Experiments: planning, analysis, and parameter design optimization*. Wiley New York, 2000.
- [105] 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.

- [106] 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.
- [107] 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.
- [108] YAN, F., MOUNTRUIDOU, X., RISKAKI, 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.
- [109] 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.
- [110] ZHANG, R., ROUFRAY, 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.