Challenges and Opportunities in Consolidation at High Resource Utilization: Non-monotonic Response Time Variations in n-Tier Applications

Simon Malkowski*, Yasuhiko Kanemasa†, Hanwei Chen*, Masao Yamamoto‡, Qingyang Wang*,
Deepal Jayasinghe*, Calton Pu*, and Motoyuki Kawaba†‡
*Center for Experimental Computer Systems, Georgia Institute of Technology, Atlanta, GA, USA
{zmon, hanwei, qywang, deepal, calton}@cc.gatech.edu
†Cloud Computing Research Center, Fujitsu Laboratories Ltd., Kawasaki, Japan
{kanemasa, kawaba}@jp.fujitsu.com
‡IT Systems Laboratories, Fujitsu Laboratories Ltd., Kawasaki, Japan
masao.yamamoto@jp.fujitsu.com

***Abstract*—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.**

***Keywords*—application co-location; cloud; consolidation; experimental study; n-tier; performance; RUBBoS; sharing.**

I. 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 [1], [2]. 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 [3].

Where a classic approach uses (ad hoc) system tuning to find and eliminate resource bottlenecks to resolve the discrepancy between concept and practice [4], 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., [5]) 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 [6]). 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.

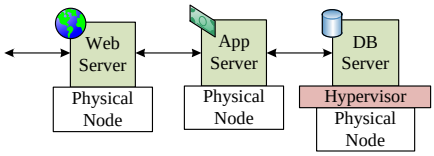
Our main contribution is an in-depth experimental study of the challenges and opportunities in performance management when consolidating n-tier applications in cloud environments at high utilization. We illustrate and explain non-trivial response time degradation phenomena, particularly and importantly, 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 2c). 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.

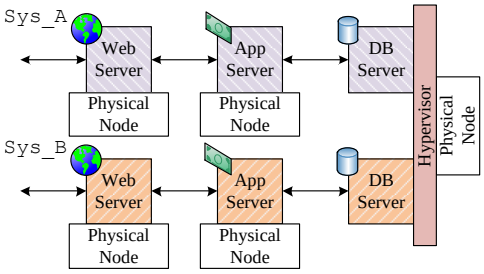
The remainder of this paper is structured as follows. In Section II we describe our experimental setup, detailing our n-tier application deployment and our testbed. Subsequently, Section III introduces our results on challenges and opportunities in consolidation of n-tier applications in clouds. Section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

II. 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



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

servers[1], 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 1a), 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 [7]. 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 1a exemplifies a dedicated n-tier deployment with one web server, one application servers, and one database server. In contrast, if the 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 1b). In our experiments, we denote the first RUBBoS system as `Sys_A`

[1]In 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.

Table I: 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 [6] 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

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 I.

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 [6].

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.

III. 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.

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 [8].

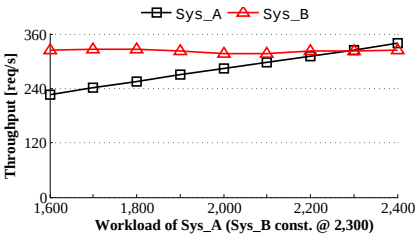
Methodologically, we consolidate two RUBBoS systems (i.e., `Sys_A` and `Sys_B`) in our experimental testbed (see Section II) to investigate how increasing resource competition may affect the performance of n-tier systems. Concretely, we increase the workload of `Sys_A` from 1,600 to 2,400 concurrent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

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 8c). `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) [8]. `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 8c). Please refer to Section III-C for further discussion of software resource allocation.

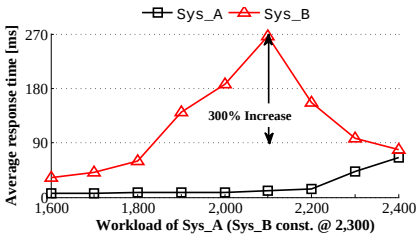
The rest of this section is organized in three parts. First, we introduce the challenging performance anomaly that is at the core of this paper in Section III-A. In Section III-B we present and test hypotheses for the explanation of the performance anomaly. Finally, we discusses opportunities

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

(a) 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.



(b) 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.



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

Figure 2: Workload (a), throughput (b), and response time (c) of 3-tier systems Sys_A and Sys_B. The deployment of the two consolidated systems corresponds to Figure 1b.

for high utilization consolidation of n-tier applications in clouds facilitated by means of appropriate software resource allocation in Section III-C.

A. Non-monotonic Response Time in Consolidation

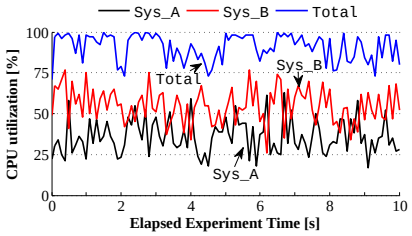
The performance anomaly, which is at the core of this paper, is depicted in Figure 2. The average throughput and response time of Sys_A and Sys_B are shown in Figures 2b and 2c, respectively. Both throughput rates are largely linear and proportional to the system workloads. Meanwhile, the average response 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. 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 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 3 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 4 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).

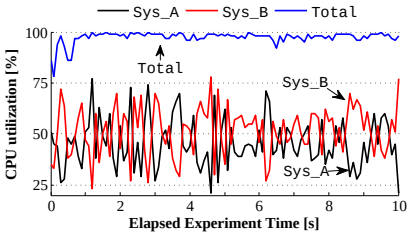
In Figure 3a Sys_A CPU utilization is significantly lower than Sys_B CPU utilization, which corresponds to the significantly lower workload of Sys_A (1,600 users) compared to Sys_B (2,300 users). Consequently, the occasionally varying App queue length in Figure 4a suggests a DB tier bottleneck with occasional 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 2c.

In Figure 3b the DB CPU utilization of Sys_A has increased significantly compared to Figure 3a. 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 4b 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 2c.

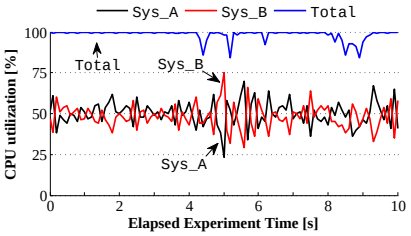
Finally, the mean values of the two DB CPU utilizations are approximately equal in Figure 3c, and the two graphs are mirror images; however, the utilizations fluctuate significantly less compared to Figure 3b. Correspondingly, Figure 4c shows the disappearance of the additional waiting time in the Web tier. This suggests that the bottleneck in the App server has been resolved. Moreover, the App queue length varies analogous to Figure 4a, which is reflected by the significant overall response time decrease shown in Figure 2c.



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

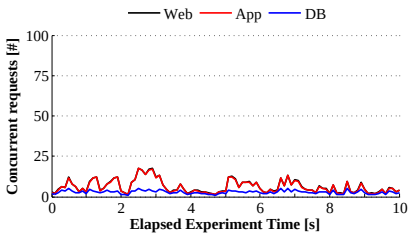


(b) Sys_A CPU utilization is only slightly lower than Sys_B, and the two graphs are mirror images, which suggests that one system waits for the other.

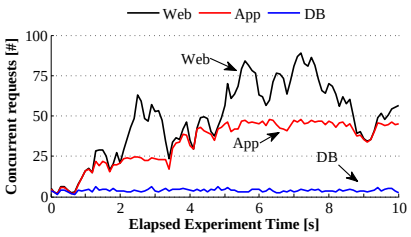


(c) The two utilizations are approximately equal on average, and the two graphs are mirror images, which suggests that one system waits for the other.

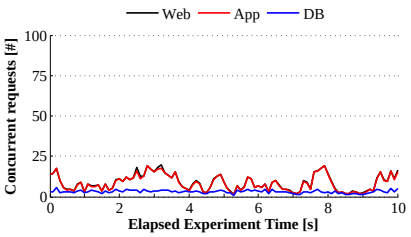
Figure 3: Trace snapshots (10 seconds) of the CPU utilizations in DB node of the backend consolidation scenario with constant Sys_B workload of 2,300 users and Sys_A workloads of 1,600 users (a), 2,000 users (b), and 2,300 users (c).



(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 4a.

Figure 4: The number of concurrent requests (processing and waiting) in Sys_B tiers (same experiment traces as in Figure 3).

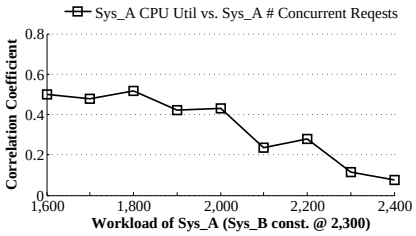
B. *Hypotheses and Correlation Analysis*

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 2c.

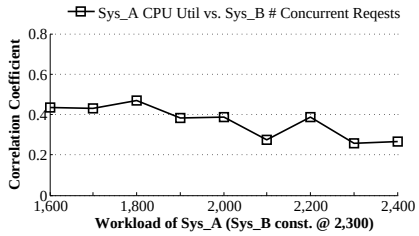
The first dimension is a result of the average-utilization-based CPU scheduling policy in the DB node hypervisor, and there are two different regions of system properties that are distinguishable in our experiments. For low workloads of Sys_A, the CPU scheduler in the DB node hypervisor preferentially 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 3b). These fluctuations of Sys_A DB CPU requirement occur naturally due the stochastic nature of the client request process. However, once Sys_A workload is increased significantly, the mean CPU utilizations become equal (Figure 3c), and the CPU scheduling priority also becomes equal. Consequently, the variation in Sys_B DB CPU utilization reduces significantly, and the the long waiting queues in Sys_B disappear (compare Figures 4b and 4c).

The second dimension is a result of Sys_B not being able to process the queue of waiting CPU-intensive (i.e., long processing time) 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 4b and causes the disproportionately large response time increase in Figure 2c.

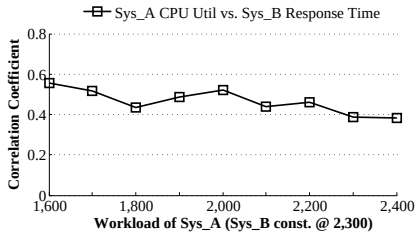
To provide quantitative evidence for our hypotheses, we



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

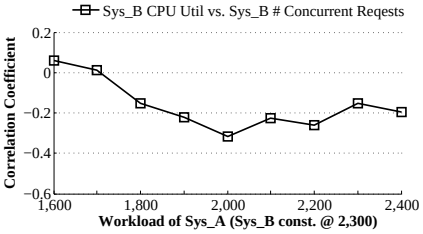


(b) Constantly high positive correlation suggests that Sys_B requests queue up when Sys_A is scheduled on the CPU for all analyzed workloads.

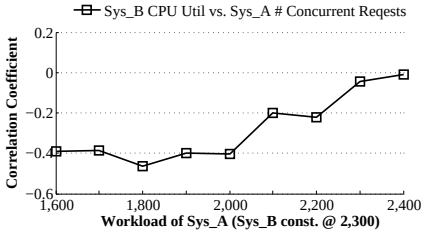


(c) Constantly high positive correlation suggests that Sys_B response time grows when Sys_A is scheduled on the CPU for all analyzed workloads.

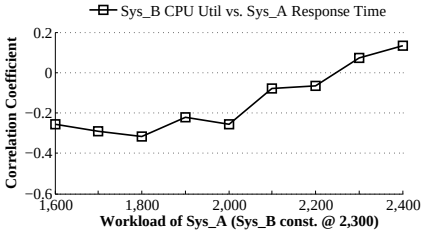
Figure 5: Correlation analysis based hypothesis testing of Sys_A properties. Pearson's correlation coefficients are calculated based on CPU utilization, number of concurrent jobs, and response time traces collected in the DB node at 0.1s granularity.



(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.



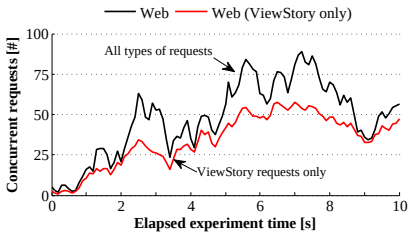
(c) Negative correlation suggests that Sys_B CPU utilization drops when the response time of Sys_A grows due to request queues. Past Sys_A workload of 2,000 users, this dependence diminishes.

Figure 6: 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.

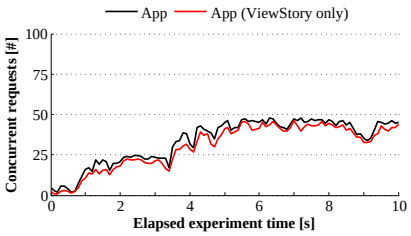
analyze the linear relationships between the main metrics in the preceding explanation (i.e., CPU utilization, number of concurrent jobs, and response time). Figures 5 and 6 show the corresponding correlation coefficient graphs for Sys_A and Sys_B, respectively. The coefficients are calculated based on traces at 0.1s granularity collected in our experiment runs. More specifically, Figures 5a and 6a illustrate the correlation between the DB CPU utilization and the number of concurrent DB requests; Figures 5b and 6b show the correlation between the DB CPU utilization and the DB response time of the co-located VM; and Figures 5c and 6c 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 5a 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 6a, 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 5b the constantly high positive correlation suggests that Sys_B requests queue up when Sys_A is scheduled on the CPU for all analyzed workloads. Moreover, the constantly high positive correlation in Figure 5c 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 6b 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 6c 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.

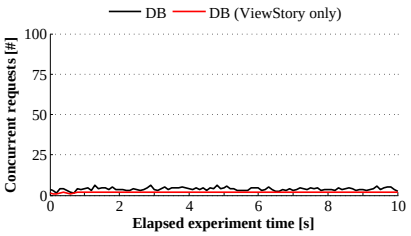
To provide quantitative evidence for our assertions about the role of long-processing requests, we illustrate the prominence of ViewStory request in all tiers of Sys_B (Figure 7). The traces in this figure correspond to the scenario shown in Figure 4b. There is a significant amount of queued requests in the Web tier of Sys_B that are not of the type ViewStory (Figure 7a). 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 7b). This suggests that most threads in the



(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 7: The number of concurrent ViewStory requests in Sys_B tiers (same experiment traces as in Figures 3 and 4).

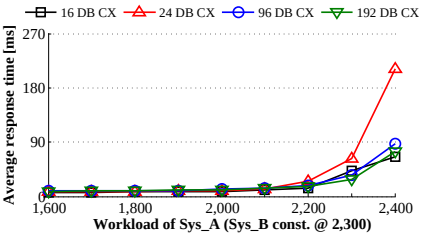
App tier are blocked on queued ViewStory requests, which explains the request backlog in the Web tier. Finally, the number of DB connections in the App server (i.e., two per servlet) determines the maximum DB request concurrency; thus, the number of requests in Sys_B DB tier remains stable (Figure 7c).

C. Software Resource Allocation in Consolidation

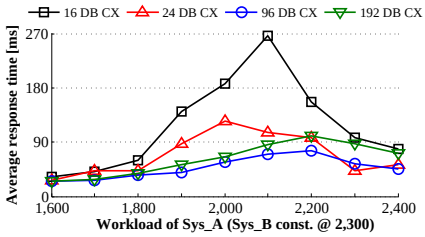
In the following we compare system response time in various software resource allocation scenarios in both consolidated and dedicated deployments. While Figures 8a and 8b show the response time of Sys_A and Sys_B in a consolidated deployment (see Figure 1b), Figure 8c shows the response time for a dedicated deployment (see Figure 1a).

Very little variation is apparent for the different software configuration settings in Figure 8a 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 8b 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 8c 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) [8].

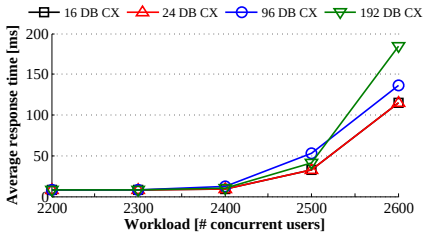
More generally, our results suggest that allocating additional software-resources in n-tier applications when deploying them in clouds may allow for higher resource utilization when sharing, without significant performance loss. Especially when compared to classic datacenter scenarios, software resource allocations should be increased to mitigate the impact of the second dimension response time amplification described in Section III-B. 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.



(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 8: 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 1.

IV. Related Work

Many related research papers on application consolidation in cloud environments (e.g., [9], [10]) 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 [5]. 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, [11]) or trace-driven simulation [12]. 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 [2]. 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 [13], Astrolabe [14], and SDIMS [15]. The second area is cloud benchmarking. In addition to RUBBoS [7], we are currently analyzing other cloud-benchmarks such as CloudStone [16] and MRBench [17].

V. 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.

References

[1] B. Snyder, "Server virtualization has stalled, despite the hype," *InfoWorld*, 2010.

[2] C. Curino, E. P. Jones, et al., "Workload-aware database monitoring and consolidation," *SIGMOD '11*.

[3] M. Aggar, "The IT energy efficiency imperative," Microsoft Corp., White Paper, 2011.

[4] R. Jain, "The art of computer systems performance analysis," John Wiley & Sons, 1991.

[5] T. Ferreto, M. Netto, et al., "Server consolidation with migration control for virtualized data centers," *Future Gener. Comput. Syst. '11*.

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

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

[8] Q. Wang, S. Malkowski, et al., "The impact of soft resource allocation on n-tier application scalability," *IPDPS '11*.

[9] Z. Gong and X. Gu, "Pac: Pattern-driven application consolidation for efficient cloud computing," *MASCOTS '10*.

[10] X. Meng, C. Isci, et al., "Efficient resource provisioning in compute clouds via VM multiplexing," *ICAC '10*.

[11] R. Zhang, R. Routray, et al.,"IO tetris: Deep storage consolidation for the cloud via fine-grained workload analysis," in *CLOUD '11*.

[12] F. Yan, X. Mountrouidou, et al., "Toward automating work consolidation with performance guarantees in storage clusters," *MASCOTS '11*.

[13] E. Thereska, B. Salmon, et al., "Stardust: tracking activity in a distributed storage system," in *SIGMETRICS '06*.

[14] R. Van Renesse, K. P. Birman, et al., "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst. '03*.

[15] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," *SIGCOMM '04*.

[16] W. Sobel, S. Subramanyam, et al., "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," *CCA '08*.

[17] K. Kim, K. Jeon, et al., "MRbench: A benchmark for mapreduce framework," in *IPDPS '08*.