Variations in Performance and Scalability: An Experimental Study in IaaS Clouds using Multi-Tier Workloads

Deepal Jayasinghe, Simon Malkowski, Jack Li, Qingyang Wang, Zhikui Wang, and Calton Pu.

Abstract—The increasing popularity of clouds drives researchers to find answers to a large variety of new and challenging questions. Through extensive experimental measurements, we show variance in performance and scalability of clouds for two non-trivial scenarios. In the first scenario, we target the public Infrastructure as a Service (IaaS) clouds, and study the case when a multi-tier application is migrated from a traditional datacenter to one of the three IaaS clouds. To validate our findings in the first scenario, we conduct similar study with three private clouds built using three mainstream hypervisors. We used the RUBBoS benchmark application and compared its performance and scalability when hosted in Amazon EC2, Open Cirrus, and Emulab. Our results show that a best-performing configuration in one cloud can become the worst-performing configuration in another cloud. Subsequently, we identified several system level bottlenecks such as high context switching and network driver processing overheads that degraded the performance. We experimentally evaluate concrete alternative approaches as practical solutions to address these problems. We then built the three private clouds using a commercial hypervisor (CVM), Xen, and KVM respectively and evaluated performance characteristics using both RUBBoS and Cloudstone benchmark applications. The three clouds show significant performance variations; for instance, Xen outperforms CVM by 75% on the read-write RUBBoS workload and CVM outperforms Xen by over 10% on the Cloudstone workload. These observed problems were confirmed at a finer granularity through micro-benchmark experiments that measure component performance directly.

Index Terms—Benchmarking, Clouds, EC2, Emulab, IaaS, multi-tier, Open Cirrus, Performance, RUBBoS, Scalability.

1 INTRODUCTION

The flexibility and scalability of clouds make them an attractive application migration target; yet, due to the associated complexity, it is difficult to make newly migrated applications run efficiently. For example, while clouds are good candidates for supplementary workloads during occasional overload of Internet applications (e.g., electronic commerce), reports on modern clouds (e.g., Amazon EC2) consistently mentioned how virtualization and network latency could affect overall system performance [17], [19], [21]. Such issues are compounded by dependencies among system components as requests are passed among the tiers, which are characteristic of real multi-tier applications. Despite some published best-practices for the popular cloud environments (e.g., [10]), the tradeoff between guaranteed performance (e.g., bounded response time) and economical efficiency (e.g., high utilization for sustained loads) remains a serious challenge for mission-critical applications. The goal of this paper is to highlight some of these challenges, provide insights through macro and micro benchmarking experiments, and propose alternative solutions. All these are to become the intermediate steps towards addressing the challenges of Cloud applications.

In this paper we study altogether six infrastructure as a service clouds (IaaS) using standard benchmark applications (RUBBoS [8] and/or CloudStone [16]) in a large state space of configuration options. First, we analyze the performance and scalability variations when a multi-tier application is migrated from a traditional datacenter environment to a public shared cloud infrastructure (public cloud comparison). Second, we evaluate the significance of performance difference on three private clouds built using three mainstream hypervisors (private cloud comparison). In public cloud comparison, we use Emulab [4] as our reference testbed and experimentally compare the performance and scalability characteristics to Open Cirrus and to Amazon EC2 [9] using RUBBoS benchmark application. In private cloud comparison, we built the three clouds using XEN, KVM, and a commercial hypervisor (referred to as CVM) and studied the performance differences using both RUBBoS and CloudStone benchmarks.

Our experiments cover scale-out scenarios under varying hardware (and software) configurations and concurrent workloads. These experiments are generated and executed automatically using the Elba toolkit [1], [14], [30]. Concretely, we use the automated experiment management tools, which set up, execute, monitor, and analyze large-scale application
deployment scenarios. To have a fair comparison among different clouds, we looked at performance trends (i.e., going up or down) rather than focusing on numbers (e.g., actual throughput and response time values). For the detailed analysis of non-trivial performance issues and system bottlenecks, we use both standard and custom built micro-benchmarks to zoom into individual system components and confirm our hypotheses at finer granularity.

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

We provide a set of candidate solutions to overcome the observed performance problems. Our solutions are solely based on application level modifications to overcome issues induced by clouds. In our study, we focused on a very small fraction of the problems, yet we observed a set of interesting phenomena. More generally, this study shows that cloud is a relatively immature technology and significantly more experimental analysis will be necessary in order for public clouds to become truly suitable for mission-critical applications.

The remainder of the paper is structured as follows. (§ 2) provides an overview of the measurement process, the six cloud platforms and the two benchmarks. In (§ 3) we analyze the three public clouds using the RUBBoS workload for performance and scalability. (§ 4) presents performance analysis results for the three private clouds using both RUBBoS and CloudStone workloads. Related work is summarized in (§ 5), and (§ 6) concludes the paper.

2 BACKGROUND

This section provides an overview of the cloud platforms, the benchmarks, the database middleware used in the experiments and the experimental processes.

2.1 Clouds Platforms

We used three public IaaS clouds for the study and corresponding hardware configurations are outlined in Table 1. We evaluated both horizontal (increasing the number of nodes) and vertical (the same number of nodes but with better hardware settings) scalability on EC2 and we limited our study to horizontal scalability in Emulab and Open Cirrus. To validate our findings on the public clouds, we extended the study by building three private clouds using three mainstream hypervisors, KVM, XEN, and CVM respectively. Table 2 provides an overview of hardware and virtual machine (VM) configurations. A typical deployment scenario for Tomcat server in VM deployment and bare metal deployment is shown in Figure 1.

2.1.1 Amazon Elastic Computing Cloud (EC2)

Perhaps the best known example of a commercial cloud is Amazon EC2 [9], which enables users to flexibly rent compute resources for use by their applications. EC2 provides resizable computing capacity in the cloud, including the number of instances (horizontal scalability) and different types of nodes (vertical scalability, with the choices such as Small, Large, Extra Large). Amazon EC2 is built using a highly modified and optimized version of the XEN hypervisor.

2.1.2 OpenCirrus

Open Cirrus is a cloud computing testbed that federates distributed data centers aiming to spur innovation in systems and applications research, build an open source service stack for the cloud and create an ecosystem for cloud services modeling.

2.1.3 Emulab Network Testbed

Emulab is a universally available time and space-shared network emulator. Several hundred PCs in racks, combined with secure, user-friendly web-based tools, and driven by ns-compatible scripts, allow one to remotely configure and control machines and link

![Fig. 1. Tomcat Deployment: VM vs. Native Hardware.](image-url)
down to the hardware level. Packet loss, latency, bandwidth, queue sizes—all can be user-defined.

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

2.1.5 XEN
Xen is an open source VM monitor which is structured with the Xen hypervisor as the lowest and most privileged layer. Above this layer are located one or more guest operating systems, which the hypervisor schedules across the physical CPUs. Xen can work both in para-virtualized or HVM mode and paravirtualization helps to achieve very high performance.

2.1.6 CVM - Commercial Virtual Machine Monitor
There are many different competing commercial virtualization technologies. Yet, licensing and copy rights issues prevent publications of performance and comparison data. Hence, we selected one of the commonly used commercial hypervisors (masked as CVM) and compared the performance with both XEN and KVM.

2.2 Benchmark Applications
During the course of our study, we used two standard benchmark applications. Concretely, for public cloud comparison we used RUBBoS and for private cloud comparison we used both RUBBoS and Cloudstone.

2.2.1 RUBBoS Benchmark Application
RUBBoS [8] is a multi-tier e-commerce system modeled on bulletin board news sites such as Slashdot, and it is widely used in large scale experiment studies. RUBBoS’s workload consists of 24 different interactions such as register user, view story and post comments. The benchmark can be deployed as a 3-tier or 4-tier system and place a high load on the database tier. RUBBoS workload generator acts as a closed system. A typical RUBBoS deployment with MySQL Cluster is shown in Figure 2.

2.2.2 Cloudstone Benchmark Application
Cloudstone [16] is a Web 2.0 benchmark, which includes a Web 2.0 social-events application Olio and can be considered as a multi-platform, multi-language performance measurement tool. More than a benchmarking service itself, it has created an open-source framework for analyzing clouds. Its workload generator is implemented using Faban, however, in our study we used the Rain workload generator which work as an open system. A typical Cloudstone deployment is shown in Figure 3.

2.3 MySQL Cluster
All the experiments discussed in this paper were run with MySQL Cluster [2] as the clustering middleware. It is an open source transactional database designed for scalable high performance access to data through both partitioning and replication. It is implemented by a combination of three types of nodes. A Management node maintains the configuration of SQL nodes and Data nodes, plus starting and stopping of those nodes. A Data node stores cluster data, and an SQL node handles the internal data routing for partitioning and consistency among replicas for replication. Figure 2 illustrates three types of nodes and how they are connected. The total number of Data nodes is the product of the replication factor and the number of partitions. For example, with a replication factor of 2 and a database divided into 2 partitions, the resulting MySQL Cluster will have 4 Data nodes. Notice, all the experiments described in this paper were run with replication factor of 2.

2.4 Experiment Measurement Process
The performance measurement process for an enterprise application consists of multiple closely related experiments by varying a few configurable (preferably one) parameters at a time. For example, finding the maximum throughput of an e-commerce application deployed on EC2 may require multiple experiments where the difference between any two would be the number of concurrent requests (e.g., 1000 vs. 2000 users). In fact, two different software/hardware configurations may produce the same throughput [13], [31], thus increasing the potential state space. Also, components of a distributed application can interact in complex ways, yielding a virtually infinite number
of possible states. In reality, evaluating all possible states is a practically infeasible exercise. Our goal is to maximize the number of configurations that can be evaluated with limited time and resource constraints. Hence, to enable the scale of thousands of experiments with hundreds of nodes, we created a set of software tools [1] [14] [30] to automate the process of setting-up, executing, monitoring, and data collection.

### 2.5 Experimental Procedure

In order to achieve fair comparisons, we used similar, if not identical, software configurations and hardware settings in experiments. We used the same operating system (Fedora16 for public clouds and Fedora16 for private clouds) and the same values for system settings (e.g., `net.core.rmem_max`, `net.ipv4.tcp_mem`). We also used the same software packages (e.g., JDK, Tomcat, MySQL etc.) and software configurations (e.g., threadpool sizes and timeout). For a given hardware configuration, we used identical nodes for all the servers.

- **Hardware Configuration** is given by the total number of nodes and the number of nodes in each server types (e.g., Tomcat and MySQL). Two hardware configurations, for instance $H_1$ and $H_2$, are different possibly in that, $H_1$ and $H_2$ have different total number of nodes or 2) $H_1$ and $H_2$ have a different number of server types (e.g., $H_1$ has two Tomcat servers while $H_2$ has three Tomcat servers).

- **Software configuration** is the aggregation of all the software settings used for a given hardware configuration. For a given hardware configuration we say it has two different software configurations if at least one software configuration parameter varies from one to another (e.g., the size of the thread pool). In our experiments we try to use the same software configurations, and when it is needed we changed them according to the algorithms described in [31].

- **Workload** corresponds to the maximum number of concurrent users that generate requests at a given time interval (i.e., 1s). In our experiments, we first select a configuration (hardware and software) and then generate requests against the configuration. We start with a small workload (e.g., 1000 users) and keep increasing the workload until system saturates.

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

Table 3 provides a high level summary of a subset (only the results we used for our publications) of different experiments performed in six cloud platforms.

<table>
<thead>
<tr>
<th>Type</th>
<th>Emulab</th>
<th>EC2</th>
<th>O.Cirrus</th>
<th>KVM</th>
<th>XEN</th>
<th>CVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiments</td>
<td>8124</td>
<td>1436</td>
<td>430</td>
<td>289</td>
<td>256</td>
<td>238</td>
</tr>
<tr>
<td>Nodes</td>
<td>95682</td>
<td>25848</td>
<td>4480</td>
<td>1287</td>
<td>1310</td>
<td>1263</td>
</tr>
<tr>
<td>Contigs</td>
<td>342</td>
<td>86</td>
<td>23</td>
<td>31</td>
<td>32</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 3: Number of Different Experiments used in the Paper

### 2.6 Notation

We use a notation #A-#T-#M-#D-ppp to represent each concrete configuration. #A denotes the no. of Web servers, #T for no. of Tomcat servers, #M for no. of SQL nodes and #D for no. of Data nodes in the experiment. A short string (ppp) denotes the platform (ec2 for EC2, em for Emulab, and oc for Open Cirrus). For example, Figure 2 shows a 1-2-2-4-em configuration with 1 Web server, 2 Tomcat servers, 2 SQL nodes and 4 data nodes in Emulab. In EC2 we add the instance (node) type at the end (small, large, exlarge, or cluster). In all of our experiments, they run with one Apache server since we found that a single server can handle all the workloads described in the paper.

### 3 Comparison on Public Clouds

We first present an experimental analysis on the three public cloud platforms. Section 3.1 presents performance on the reference platform (i.e., Emulab) with a comparison with Open Cirrus, and Section 3.2 provides scalability and performance analysis of EC2.

#### 3.1 Performance on Reference Platforms

To compare and contrast the performance complications it is essential to have a good reference, and...
we used Emulab for this purpose. We start with the smallest possible hardware configuration and gradually scale up based on the observed results.

In Emulab we started with 1-2-2-2-em configuration (the smallest configuration that can withstand 1000 users) and performed experiments from 1000 to 10,000 users with 1000 user increments. As shown in Figure 4(a), for the 1-2-2-2-em configuration, throughput saturated at 3000 users. Detailed analysis of monitoring data shows that the CPU utilization of the SQL node had become the bottleneck (see Figure 6(a)). To resolve it, we scale-out the application and add two additional Data nodes (1-2-2-4-em configuration) subsequently partitioning the database into two groups. From the experiment with 4 Data nodes and 2 SQL nodes, we observed that the throughput saturation point shifted from a workload of 3000 to 5000. Next, we added more SQL nodes to get 1-2-4-4-em (not shown in the figures), and we continued the horizontal scalability process until we resolved the hardware bottlenecks. Our analysis shows 1-4-16-16-em as a reasonable configuration that can sustain a RUBBoS workload without system hardware resources being bottlenecked. Yet, 1-4-16-16-em encounters software resource bottlenecks (interested readers should refer to our previous work [31]). In summary, the RUBBoS application scales well in Emulab.

In a web application, throughput indicates how much workload that the application can handle; however, to understand actual end-user experience it is necessary to see the response time characteristics (i.e., latency). The recorded average end-to-end response time values are illustrated in Figure 5(a). As shown in the figure, 1-4-16-16-em has the lowest response time. In fact, we observed a low average response time until the workload reaches 6000 users when the response time started to increase sharply. This ob-
We used the same methodology as in Section 3.1, and which implies that the bottleneck shifts when the observed throughput and response time values are illustrated in Figure 6(b). We extended the bottleneck analysis to other configurations and a high level summary of our results is shown in Figure 7. The figure shows each configuration and observed average CPU utilization when the system is saturated. The highlighted cells represent the bottlenecked servers in question, which implies that the bottleneck shifts when the configuration and/or the workload vary [22], [29].

We used the same procedures as in Emulab and performed a similar set of experiments on Open Cirrus. As shown in Table 1, we used more powerful machines in Open Cirrus, and thus as expected, we achieved comparatively higher throughput values. The observed throughput and response time values are illustrated in Figure 4(b) and Figure 5(b), respectively. As shown in the figures, Open Cirrus also shows good performance and scalability.

In summary, both Emulab and Open Cirrus show good scalability characteristics, where the observed average throughputs go up as the number of nodes increased. Notice, in Open Cirrus we observed very low resource utilization (e.g., less than 30% CPU), thus, we have omitted utilization graphs.

### 3.2 Amazon EC2 Performance

#### 3.2.1 Horizontal Scale-Out (same nodes)

We used the same methodology as in Section 3.1, and performed a similar set of experiments in EC2. We kept as many experimental parameters untouched as possible (except platform specific parameters). Our initial expectation was to achieve better or similar results on EC2 than those from Emulab since we used better hardware in EC2. Surprisingly, our assumptions were shown to be wrong when we observed quite different behavior in EC2. As shown in Figure 4(a) and (b) RUBBoS scales well (i.e., increasing the number of nodes gives better performance) on both Emulab and Open Cirrus; in contrast, in EC2, increasing the number of nodes reduces the performance. As shown in Figure 4(c) scaling from 1-2-2-2-ec2 to 1-2-2-4-ec2 significantly reduced the throughput and scaling from 1-2-2-4-ec2 to 1-2-2-8-ec2 further reduced the throughput. In summary, RUBBoS on Emulab and Open Cirrus shows good scalability, but it shows poor scalability on EC2.

The observed response time values for EC2 are shown in Figure 5(b). As illustrated, after a workload of 2000, the response time increases significantly. In addition, 1-2-2-2-ec2 shows the lowest response time values and 1-2-2-8-ec2 shows highest response time values amongst three configurations. These response time values and their trend support the observed throughput trend where 1-2-2-2-ec2 shows the best performance amongst the three. In fact, as compared to Emulab’s response time values (Figure 5(a)), EC2 shows larger response times. The response time behavior of Emulab was explainable using the resource monitoring data (e.g., CPU); in contrast, EC2 shows very low resource utilizations (far below saturation), yet, considerably higher average response time values. Figure 6(a), 6(b) and 6(c) illustrate the CPU utilizations for both Emulab and EC2. Different from Emulab and Open Cirrus, EC2 shows a very low CPU utilization yet significantly worse performance with the multiple configurations and different server types as shown in the next section.

#### 3.2.2 Vertical Scalability (different nodes)

We started our analysis with 1-2-2-2-small (similar hardware configurations as Emulab) and observed comparatively low throughput, our results are shown in Figure 8. Node level data analysis revealed that for small instances, the Data node showed a large number of disk I/O compared to that of large instances. There are two main reasons causing this issue. First, as argued in [19], EC2 small instances always get 40% to 50% of the physical CPU sharing (i.e., they get only 40-50% of the CPU cycles from what they are supposed to get). Second, the physical memory (Table 1) may not be enough. According to MySQL cluster documentation [2], small instances do not have sufficient memory to keep the complete RUBBoS database in memory; thus, it frequently uses virtual memory while processing requests.

![Fig. 8. Throughput: EC2 Vertical Scalability](image-url)

As a practical solution, we migrated to large instances and followed the same procedure; likewise,
we extended the same process for the other EC2 instance types as well. As shown in Figure 8, EC2 shows good vertical scalability and we achieved the highest throughput with cluster instances (the most powerful instances available at the time we performed our experiments).

As illustrated in Figure 6(a), (b) and (c), CPU utilization alone is unable to explain EC2's performance degradation. For each experiment we collected over 24 types of monitoring data (e.g., CPU, Memory, Disk and network IO, context switches), and thus, we extended our analysis on monitoring data and micro-benchmarks. We observed two key issues: overhead associated with concurrent threads (e.g., context switching and scheduling overhead) and network driver overhead (e.g., jitter and queuing delays).

3.3 Concurrent-threading Costs in EC2

In this section, we provide the related analysis for EC2 by using the RUBBoS workload generator (a multi-thread workload driver) as a case study. Section 3.3.1 quantifies the context switching overhead, and Section 3.3.2 shows how to redesign the application to guard against the multi-threading issue.

Complex dependencies of multi-tier applications complicate the performance analysis process. As previously discussed (Section 3.2), the RUBBoS application saturated in EC2 due to effects of initially hidden resources utilization. Consequently, we extended our systematic analysis with micro benchmarks. Subsequently, we observed an interesting phenomenon when using multi-threading applications on EC2. To illustrate, we selected a RUBBoS client workload generator and measured the round trip time (RTT) for each request for different workloads. We observed a sudden increase in RTT when the workload increases. Typically, these types of observations lead to assumptions of back-end server saturation; despite this, we observed the issue to be at the workload generator.

A typical request-response flow for a RUBBoS is shown in Figure 9(a). For each request, the client sees \( \sum_{i=1}^{6} t_i \) (say \( RTT_1 \)) as the RTT, where as RTT at Apache is \( \sum_{i=2}^{3} t_i \) (say \( RTT_2 \)). We recorded RTT for each individual request at the workload generator (i.e., \( RTT_1 \)) and the Apache server (i.e., \( RTT_2 \)). Next, we used two log files and calculated average RTT separately. The results are shown in Figure 9(b). In the figure, the original client represents the response time recorded at client and web server represents the response time recorded at Apache. As shown in the figure, for higher workloads, the calculated RTT at the client differs significantly from that of Apache. In contrast, for lower workloads, the RTT difference is negligible (e.g., workloads less than 3000). Figure 9(b) illustrates how the increase of workloads causes the recording of larger RTTs at the client.

3.3.1 Context Switching Overhead

To further analyze the observed phenomenon we used LMBench [6]. We started with the default settings and performed all supported tests on Emulab and EC2. We found surprising results for context switching overhead. It recorded the overhead in EC2 to be twice as high as in Emulab. Figure 10(a) shows the time for a single switch when varying the number of threads. As generally accepted, measuring context switching time with high accuracy is a difficult task. Nevertheless, analysis shows that for a few number of threads (< 40) EC2 takes less time for a switch compared to Emulab; in contrast, with a greater number of threads, EC2 takes considerably longer time.

In RUBBoS, each client gets a large number of threads which is in fact equal to the workload; therefore, context switching becomes a key issue. To illustrate the significance, we calculated the average number of context switches when running the benchmark. Figure 10(b) shows the calculated averages for both EC2 and Emulab. As shown in the figure, Emulab has a significantly higher number of context switches compared to that of EC2. Moreover, combining Figure 4(a), Figure 4(c) and Figure 10(b) we can observe a positive correlation between the throughput and the number of context switches.

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

3.3.2 Proposed Candidate Approach

The maybe most trivial solution in commercial clouds is to rent more nodes so that each gets less users (threads). This helps to reduce the overhead caused by concurrent threads (see Section 3.3.1). While this solution is acceptable for occasional users that run an application only a few times, for long term users who run their applications often or continuously, this may not be the most economical.

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

Our solution shows a significant improvement, both in RTT (Figure 9(b)) and throughput (Figure 9(c)). As shown in the figure, with our solution we were able to bring $RTT_1$ much closer to $RTT_2$. In addition, as shown in Figure 9(c) our solution produces a significantly higher throughput. We confirmed our hypothesis by calculating the average RTT using Apache logs and the client logs. With the thread reduction, the recorded response time at client is much closer to that of the Apache. The remaining difference can be explained with the queuing effects that naturally take place between the two tiers. While this solution is practical in many cases, it depends on the nature of the application whether the desired concurrency level can still be reached in this manner.

### 3.4 Network Driver Overhead

We increased the number of client nodes to 10 and resolved the client issue, and subsequently shifted the bottleneck to the back-end. Section 3.4.1 shows the network traffic increase through database partitioning and subsequent network transmission overhead on EC2. In Section 3.4.2 we provide our solutions to achieve higher performance in public clouds.

#### 3.4.1 Analysis of Network Driver Overhead

We extended our analysis to the network layer, and our results revealed two important findings. First, increasing the number of Data nodes (partitioning the database) caused a significant increase in the network traffic generated at the Data nodes (see Figure 11 (a)). Moving from 2 Data nodes to 4 Data nodes doubles the network traffic. Second, transmission queue sizes for EC2 were significantly higher compared to that of Emulab (see Figure 11 (b)). Nevertheless, for the 1-2-2-2 configuration, both Emulab and EC2 show similar network traffic patterns and also somewhat similar throughput values; however, 1-2-2-4 shows different behaviors both in network traffic and throughput (Figure 4(a) and (c)).

We selected 1-2-2-4-em and 1-2-2-4-ec2 and used a number of network analysis tools to observe
network behavior. Through our data, we observed an interesting behavior in the Data nodes. As shown in Figure 11(b), we observed that the sending queue size (at the network layer) of the Data node in EC2 is much higher than in Emulab. In contrast, the receiving sides of both EC2 and Emulab show similar characteristics, and both have negligible queue sizes.

In a RUBBoS application with MySQL Cluster, the Data node is the first server which generates a large amount of data in the request propagation pipeline. As shown in Figure 9(a), initially, users send an HTTP request to Apache HTTPd, and then Apache forwards the request to Tomcat. Tomcat processes the request and generates SQL statements that are sent to the SQL servers. The SQL servers send the query to the Data nodes, which then processes these queries and generates results. In this process, especially in the read-only scenario, the message sizes of the generated output are significantly higher than the incoming message (SQL query vs. a result set). Therefore, when the message rate is high, the Data node generates more data; however, when there is a problem at the transmitting side, the Data node cannot send the data as required, which then results in a long queue at the network buffers. Next, in the other tiers (e.g., Tomcat and Apache HTTPd), the connections start to wait as well. Therefore, the overall message transmission rate reduces. Eventually this behavior affects the entire system performance.

To further analyze observed hypothesis, we used NetPIPE [7] and measured the achievable bandwidth. NetPIPE is a protocol-independent performance tool that visually represents the network performance under a variety of conditions. Our results show that the two systems have similar bandwidth, thus, confirming that the sending buffer issue is not caused by network bandwidth. We then used the Unix Ping (ICMP) program and measured the RTT between the two Data nodes. First, we evaluated ping RTT without generating any application load (i.e., without generating any RUBBoS client requests), thus, making ping program the only process running on both the nodes. Second, we evaluated RTT while generating a representative application load (i.e., while running RUBBoS). We observed a very interesting behavior. When there is no load on the node, EC2 shows significantly less RTT compared (approximately 20 times less) to Emulab (see Figure 12(a)). In contrast, when running with the RUBBoS workload, EC2 shows very large RTT that varies between 0.4 ms to 30ms (see Figure 12(b)), but Emulab shows similar results for both cases (average of 0.2 ms). The first scenario simply means that the two nodes have good network connectivity as compared to that of Emulab; however, in EC2 when the load on the network driver increases, RTT goes up significantly whereas Emulab maintains the same RTT regardless of the load. This provides additional evidence to confirm network driver overhead in EC2 and potential performance degradation.

3.4.2 Proposed Candidate Approach
As we discussed in Section 3.4, due to this network overhead, the scalable software system is unable to scale in EC2. In general, MySQL Cluster is a sophisticated but heavily used database middleware in industry. It has a number of advantages including availability, reliability, and scalability compared to other alternatives. Unfortunately, due to higher network traffic generation of MySQL cluster, the complete application shows poor scalability.

To illustrate the significance and to provide an alternative solution, we used the C-JDBC [20] middleware and performed the same set of experiments as in Section 3.1 on EC2. The observed throughput values are shown in Figure 13(a). As shown in the figure C-JDBC shows very good scalability and achieves very high throughput. Next, we measured the amount of data generated at the database tier by the two approaches, and our results are shown in Figure 13(b). As demonstrated in the figure, C-JDBC generates a significantly smaller amount of data compared to MySQL Cluster. Consequently, the middleware results in lower pressure on the network, which causes better performance. In contrast, MySQL Cluster produced a large network traffic and higher pressure on the network driver.
4 Comparison on Private Clouds

To validate our findings on public clouds, we built three private clouds and performed similar set of experiments using two multi-tier benchmark applications. In this section we compare and contrast the observed performance results on three private clouds and also on bare metal hardware.

4.1 Base Performance Comparison

The goal of our study is to obtain a fair performance comparison among three clouds and to validate the results with native hardware. Hence, for all our experiments (regardless of the hypervisor) we selected an identical VM configuration (i.e., CPU, Memory, Disk and Operating System) and an identical workload. More specifically, throughout our study we generate workloads from a fixed set of physical machines (to keep the workload generator as constant). Each of our experiments, we have used dedicated deployment i.e., only one VM per physical host. We used that approach to reduce the interference from other VMs. In addition, to observe the performance characteristics on native hardware, we disable six cores and brought it to similar configuration as VMs, and then repeated the same procedure on native hardware as well.

4.2 RUBBoS - Read Only

We first deployed RUBBoS application with MySQL cluster database middleware and executed workload from 1000 concurrent users to 7000 concurrent users (with step of 1000 users) on all the three clouds. The observed throughput results and response time values are shown in Figure 14(a) and Figure 15(a), respectively. As illustrated in the two figures, native hardware gave the best performance, which is in fact the expected behavior. Among three clouds, XEN and CVM show similar performance characteristics (both response time and throughput), in contrast, KVM shows significantly less throughput and comparatively high response time. As shown in the Figure 14(a), throughput on KVM remains constant against increasing workload; this is a clear indication of system saturation. Additionally, both XEN and CVM started to saturate at the workload of 3000 concurrent users while native hardware continued to perform better.

4.3 RUBBoS - Read Write

We then used the same approach as in read-only scenario and extended our experiments to read-write mix workload, the goal was to introduce significant amount of disk writes. Due to the nature of transactions types in RUBBoS, we expected to see relatively higher throughputs values compared to that of read-only. The observed results (i.e., throughput and response time values) are shown Figure 14(b) and Figure 15(b) respectively. We observed a noticeable difference for CVM, where for read-only workloads both CVM and XEN show similar results (throughput and response time), in contrast, with read-write XEN outperformed CVM. The throughput difference between CVM and XEN for 6000 workload was 78%, and the same workload CVM show 200% higher response time than XEN. KVM behavior remains unchanged when moving from read-only to read-write mix workloads, and in both the cases it shows very poor performance.

We extended our analysis to resource utilization data on all the servers to explain the observed performance differences. Due to the space constraints, we presented only the most interesting results. In RUBBoS deployment (see Figure 2), data node is the last component in the request pipeline and which does most disk/network I/O. Hence, its resource utilization significantly affect the overall system performance. Thus, we looked at CPU utilization, in particularly CPU breakdowns (i.e., \texttt{usr}, \texttt{sys}, \texttt{wait}, \texttt{sig} and \texttt{hiq}) from which we were able to explain the observed performance difference.

The observed CPU utilization for KVM, CVM and XEN for read-only workloads are illustrated in Figure 16 (a), (b) and (c), respectively. As shown in the figure, both CVM and XEN has their highest utilization for \texttt{usr} component, while KVM has its highest for \texttt{sys}. In addition, KVM spends more CPU cycles for software interrupts (\texttt{sig}), hardware interrupts (\texttt{hiq}) and \texttt{io waits}, and CVM spends similar amount of CPU cycles for software interrupts. Noticeably, XEN spends less CPU cycles on software interrupts while (as expected) it spends zero CPU cycles on hardware interrupts. Among all three clouds, KVM has considerably higher CPU \texttt{wait}, while XEN has lowest aggregated CPU utilization. As shown in Figure 14(a), both XEN and CVM gave similar throughput values, yet, CVM spends more CPU cycles to achieve the same throughput as compared to XEN.

We then extended our analysis to read-write data, our results for KVM, CVM and XEN are shown in Figure 17 (a), (b) and (c), respectively. There we observed three noticeable differences: 1) CPU \texttt{wait} changes for KVM, 2) CPU \texttt{usr} behavior changes for CVM, and 3) increase of utilization difference between \texttt{usr} and \texttt{sys} for both XEN and CVM. For read-only workloads, KVM shows around 3% utilization for CPU \texttt{wait}, in contrast, that value went to 8% for read-write workloads. The difference for XEN and CVM were negligible. As shown in Figure 14(b), XEN performed better as compared to CVM, yet CPU utilization data for \texttt{usr} shows the opposite. That is both XEN and CVM showed increase of \texttt{usr} component, while CVM has around 25% when XEN as only 20%. As shown in Figure 16(b), CVM has very small gap between its \texttt{usr} and \texttt{sys} component for read-only workloads, however as shown in Figure 17(b) the difference become significantly large. Interestingly, we
observed similar results for XEN as well.

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

### 4.4 Comparison with Cloudstone

Cloud is used for vast variety of applications. To better understand the behavior we selected CloudStone, a new benchmark with completely different workload characteristics and interaction patterns compared to RUBBoS. We used the same set of VMs and follow similar procedures as previous cases i.e., measuring the performance while increasing the workload.

One of the noticeable difference between RUBBoS and Cloudstone is the workload generators, where RUBBoS uses a closed system while Cloudstone (Rain) uses an open system. Thus, in Cloudstone it records the number of effective requests (completed successfully) vs. offered workload. The observed value for effective load when increasing number of thread is shown in Figure 14(c) and observed response time values are shown in Figure 15(c). As we observed in two previous scenarios, XEN and CVM out performed KVM, yet we observed an interesting phenomenon with XEN which we will discuss later in this paper.

#### 4.4.1 Cloudstone Performance Issues in XEN
As we discussed, in Figure 14(c) we observed an interesting effective load characteristics in XEN, we repeated the experiment for multiple times but observed the same behavior. As shown in the figure, KVM and CVM show more stable curve for the effective load.
To explain the observed phenomenon we extended our analysis to different resource monitoring data (i.e., disk I/O, network I/O, CPU). Notably, we observed a matching pattern in the CPU utilization, which is illustrated in Figure 18(a). Nevertheless, average CPU utilization graph alone does not provide better insight to the problem, thus, we analyzed individual CPU utilization (i.e., sys, user, idle etc...).

Through our analysis we noticed that, while both KVM and CVM have almost zero CPU utilization for wait, XEN shows higher utilizations for some workloads. We then use effective load and CPU wait distribution and inserted both into a correlation model and observed a complete negative correlation between CPU wait and effective load. The observed results for the correlation is shown in Figure 18(c), and Figure 18(b) illustrates CPU wait for three hypervisors.

In summary, performance comparison results on three hypervisors using multi-tier workloads show interesting results. More precisely, the observed performance were highly sensitive to the application workloads. Nevertheless, XEN and CVM show similar results and outperformed KVM.

5 RELATED WORK

The increasing popularity of cloud computing has spawned very interesting research. Li et al. presented a method for achieving optimization in clouds by using performance models in the development, deployment, and operation of applications that run in the cloud [11]. They illustrated the architecture of the cloud, the services offered by the cloud to support optimization, and the methodology used by developers to enable runtime optimization of the clouds. Thomas et al. analyzed the cloud for fundamental risks that may arise from sharing physical infrastructure between mutually distrustful users, even when their actions are isolated through machine virtualization [12]. Ward et al. discussed the issues and risk associated with migrating workloads to clouds, and the authors also proposed an automated framework for “smooth” migration to cloud [15].

There are many studies focus on workflow execution and performance considerations. For example, Ewa et al. [23] examined the tradeoffs of different workflow execution modes and provisioning plans. Alexandru et al. [24] analyzed the problem of provisioning Cloud instances to large scientific workflows. They also proposed an extension to the dynamic critical path scheduling algorithm to consider general resource leasing model. Simon et al. [28] argued that initial target workloads of clouds does not match the characteristics of MTC-based scientific computing workloads, and then through empirical analysis they showed that while current cloud services are insufficient for scientific computing at large they may still be a good solution for the scientists who need resources instantly and temporarily.

Guohui et al. have analyzed the network overhead on EC2 and presented a quantitative study on end-to-end network performance among different EC2 instances [19]. Similar to our findings, his experiments also revealed the network as one of the key overhead sources. As part of our analysis, we have shown the significance of virtualization and associated network overhead. Padma et al. have looked at both the transmit and receive aspects of this workload. Concretely, they analyzed virtualization overheads and found that in both cases (i.e., transmit and receive) the limitations of VM scaling are due to dom0 bottlenecks for servicing interrupts [3]. Koh et al. also analyzed the I/O overhead caused by virtualization and proposed an outsourcing approach to improve the performance [18].

Empirical studies on virtualization and related technologies have been an active research area in the fast decade. For example, Deshane et al. [26] focus on three aspects of benchmarking Xen and KVM: overall performance, performance isolation, and scalability. They illustrate that Xen has excellent scalability while KVM has substantial problems with guests crashing when hosts more than 4 guests. KVM outperforms Xen in isolation. In overall performance test, Xen has a better performance than KVM on a kernel compile test while KVM outperforms Xen on I/O-intensive tests. Camargos et al. [27] analyze the performance and scalability of six virtualization technologies (QEMU, KVM, Linux-VServer, OpenVZ, VirtualBox and Xen) for Linux. They find Linux-Server delivers little or even no overhead in all test. Meanwhile, Adams
et al. [25] compare software VMM (binary translation) with hardware-assisted VMM. They show that software and hardware VMMs both perform well on compute-intensive workloads. However, if workloads include progressively more privileged operations such as context switches, memory mapping, I/O, interrupts and system calls, both VMMs suffer overheads while software outperforms hardware.

6 CONCLUSION

In this paper we presented an experimental analysis of performance and scalability variations on six cloud platforms. We employed RUBBoS and/or Cloudstone applications to measure the performance on the three public clouds (Emulab, OpenCirrus, and EC2) and the three private clouds built using the three mainstream hypervisors (XEN, KVM, and CVM). Especially, the comparison of EC2 and Emulab yielded some surprising results. In fact, the best-performing configuration in Emulab became the worst-performing configuration in EC2 due to a combination of several factors. Moreover, in EC2 the network sending buffers limited the overall system performance. For computation of intransitive workloads, a higher number of concurrent threads performed better in EC2 while for network based workloads, high threading numbers in EC2 showed a significantly lower performance. Our data also exemplified the significance of context switching overheads. Similarly, we observed significant performance variations among three hypervisors, more precisely, Xen outperforms the commercial hypervisor by 75% on the read-write RUBBoS workload and the commercial hypervisor outperforms Xen by over 10% on the Cloudstone workload.

More generally, as shown by the significant variance of both the public and private clouds, this work enhances the understanding of the risks and rewards when migrating multi-tier application workloads into clouds. The results imply that clouds will require a variety of further experimental analysis to be fully understood and accepted as a mature technological alternative. As future work, we are to continue the experimental analysis and come with more general methodology to identify the bottlenecks for cloud platforms and to mitigate the bottlenecks.

ACKNOWLEDGMENTS

This research has been partially funded by National Science Foundation by IUCRC/FRP (1127904), CISE/CNS (1138666), RAPID (1138666), CISE/CRI (0855180), NetSE (0905493) programs, and gifts, grants, or contracts from DARPA/I2O, Singapore Government, Fujitsu Labs, Wipro Applied Research, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

REFERENCES


Deeap Jayasinghe received the BSc degree in computer science and engineering from University of Moratuwa Sri Lanka before coming to Georgia Tech. He is currently working toward the PhD degree in the College of Computing at Georgia Institute of Technology. He is affiliated with the Center for Experimental Research in Computer Systems (CERCS) where he works with Professor Calton Pu. His research focuses on large scale experiment automation through code generation and experimenting on system performance and scalability. Deeap is a long term contributor to Apache Software Foundation and has over six years of industrial experiences on SOA and Web Services. He has written two books, more than 30 technical magazine articles and research papers, and has delivered speeches in various SOA and Web Services conferences.

Simon Malkowski is a PhD candidate in the College of Computing at Georgia Institute of Technology. His advisor is Prof. Calton Pu, and Simon is affiliated with the Center for Experimental Research in Computer Systems (CERCS). His research interests include analysis of large enterprise application systems in clouds and system support for autonomic computer systems.

Jack Li is a PhD student in the College of Computing at the Georgia Institute of Technology. He received a BSE in electrical and computer engineering from Duke in 2010. His current research interests distributed and cloud computing, hypervisors and virtualization, MapReduce, and n-tier application performance and scalability.

Qingyang Wang received the BS degree in computer science from Wuhan University, China, in 2004, and the MS degree in computer science from Chinese Academy of Sciences, in 2007. He is currently a computer science PhD candidate in College of Computing, Georgia Institute of Technology, Atlanta, Georgia. His research focuses on performance debugging and scalability analysis for distributed systems, especially multi-tier enterprise systems.

Zhihui Wang received his Bachelor of Engineering degree in Process Automation and Automatic Instrumentation from Tsinghua University, Master of Engineering degree in Industrial Automation from Chinese Academy of Sciences both in Beijing, China. He received his Ph.D. degree from University of California, Los Angeles in Electrical Engineering, Control Systems. He then joined HP Labs in 2005. He is now a senior research scientist in Sustainable Ecosystem Research Group, HP Labs in Palo Alto, California. His research interests are in application of control and optimization techniques in sustainable IT ecosystems, including application performance control, workload management, data center power management, data center cooling control and networking management.

Calton Pu was born in Taiwan and grew up in Brazil. He received his PhD from University of Washington in 1986 and served on the faculty of Columbia University and Oregon Graduate Institute. Currently, he is holding the position of Professor and John P. Imlay, Jr. Chair in Software in the College of Computing, Georgia Tech. He has worked on several projects in systems and database research. His contributions to systems research include program specialization and software feedback. His contributions to database research include extended transaction models and their implementation. His recent research has focused on automated system management in clouds (Elba project) and document quality, including spam processing. He has collaborated extensively with scientists and industry researchers. He has published more than 70 journal papers and book chapters, 200 conference and refereed workshop papers.