2012 IEEE Ninth International Conference on Services Computing

Profit-Based Experimental Analysis of IaaS Cloud Performance: Impact of Software Resource Allocation

Jack Li, Qingyang Wang, Deepal Jayasinghe, Simon Malkowski, Pengcheng Xiong, Calton Pu
Center for Experimental Research In Computer Systems
Georgia Institute of Technology
Atlanta, GA, USA
{*jack.li, qywang, deepal, zmon, pxiong3, calton*}*@cc.gatech.edu*

Yasuhiko Kanemasa, Motoyuki Kawaba
Cloud Computing Research Center
Fujitsu Laboratories Ltd.
Kawasaki, Japan
{*kanemasa, kawaba*}*@jp.fujitsu.com*

***Abstract*—High resource utilization is an important goal in achieving high return on investment in cloud environments. Guaranteed quality of service (QoS) is an important goal for web-facing applications such as e-commerce. Achieving both high utilization and high QoS simultaneously is a significant challenge, since high utilization often implies more QoS failures such as long response times. In this paper, we adopt a profit model based on response time (i.e., decreasing or negative revenues for increasing query answer response time) to represent the QoS requirements. Our data shows that such a profit model often yields different analytical results compared to traditional performance metrics such as average throughput.**

Using extensive experimental measurements (on the same hardware platform and software stack) of the standard RUBBoS n-tier benchmark, we study the impact of different allocations of software resources such as the size of thread pools in various servers in an n-tier system. First, the profit model emphasizes the importance of appropriate allocations, showing a difference of up to 48.6% when system utilization is high (over 80%). Second, our experiments show that over-allocation of thread pool may lead to unnecessary consumption of critical resources (e.g., CPU) that reduce profits by up to 84.8%. Third, we found that under-allocation of thread pool in one server may lead to under-utilization of several servers downstream in an n-tier system, also reducing profits by up to 52.8%. Our data shows that the best allocation depends on several system parameters, including resource availability. We designed an adaptive algorithm to find the best allocations and show its effectiveness through our experiments and analyses.

***Keywords*-bottleneck; configuration; n-tier; profit model; software resource**

I. Introduction

Infrastructure as a service (IaaS) provisioned by cloud computing environments offer lower costs through economy of scale in shared data centers. Cloud users can reduce or eliminate capital expenditures by a commonly adopted pay-as-you-use cost model, since they can add more hardware resources if needed during peak workloads. Virtualized platforms such as VMware and Xen are typically used to consolidate users and applications on the same hardware, increasing the hardware utilization and consequently the return on investment (RoI) for cloud providers. Unfortunately, anecdotal and reported average utilization is as low as 18% [12], primarily due to the need to preserve quality of service objectives specified in service level agreements (SLAs). Consequently, one of the critical research and practical questions in the fulfillment of IaaS model is the level of hardware utilization achievable in a cloud while maintaining the quality of service requirements of users and applications that share the same underlying infrastructure.

The first contribution of this paper is the adoption of a set of profit-based performance criteria derived from typical SLAs [11] in the evaluation of web-facing application performance. The profit model emphasizes both the positive contribution of queries returned within a short response time window (typically within a fraction of a second) and the penalties of very long response times (e.g., more than 2 seconds). The differences between profit-based evaluation criteria and traditional performance metrics (e.g., throughput without regard to response time) become significant at high resource utilization levels. Our analysis shows that despite a relatively flat throughput curve, rapid deterioration of response time quickly reduces profits at well-defined high resource utilization levels, demonstrating the importance of including response time in the evaluation of trade-offs between high quality of service and high resource utilization in IaaS clouds. We have found that for the same workload and hardware configuration, different software allocations can result in profit differences of up to 84.8%.

The second contribution of this paper is an experimental demonstration of the complexity of interactions among various systems factors at different system levels that affect the performance of web-facing applications. Concretely, we run more than 500 of experiments on the RUBBoS benchmark [1], based on Slashdot, implemented in a 4-tier configuration. In these experiments, the hardware platform and systems software (e.g., operating system) remain fixed. By varying the software resources (e.g., the sizes of thread pool and database connection pool in the Tomcat application server), we observe significantly sub-optimal performance (in terms of profits) for two simple allocation algorithms:

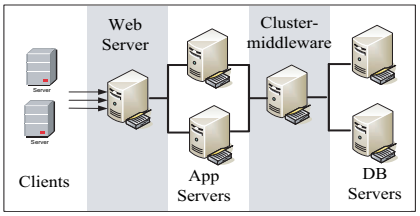
978-0-7695-4753-4/12 $26.00 © 2012 IEEE
DOI 10.1109/SCC.2012.85

Function	Software
Web Server	Apache 2.0.54
Application Server	Apache Tomcat 5.5.17
Cluster middleware	C-JDBC 2.0.2
Database server	MySQL 5.0.51a
Sun JDK	jdk1.6.0_14
Operating system	Redhat FC4 Kernel 2.6.12
System monitor	Systat 7.0.2

(a) Software setup

Hardware	Components
Server type	PC3000 in Emulab
Processor	Xeon 3GHz 64-bit
Memory	2GB
Network	1Gbps
Disk	2*146GB 10,000rpm

(b) Hardware node setup



(c) Sample topology

Figure 1: Details of the experimental setup on the Emulab cluster

conservative (small number of threads to minimize resource consumption) and liberal (large number of threads to maximize concurrent execution). Sub-optimal allocations of the application server's thread pool and database connection pool can decrease performance by 5 to 30 percent which results in a profit loss between 29 and 85 percent in our enhanced SLA profit model.

Our experiments also show that an iterative algorithm can find a very good allocation of software resources to maximize profit by avoiding under-allocation (conservative) and over-allocation (liberal). Perhaps unsurprisingly, the best allocation remains highly dependent on the workload distribution and profit model adopted in the evaluation. In our example, we use an enhanced SLA model which specifies different profits and penalties depending on the response time of the application.

The results described in this paper are significant for future research on high utilization of IaaS clouds in two ways. First, it is likely that the future work will benefit from a profit-based evaluation, which sharpens the effective performance impact, instead of classic throughput analysis without regard to response time. Second, the highly non-trivial interactions among the system components and system layers shows the challenges (many sub-optimal configurations) and opportunities (significant gains if "done right") of consolidating mission-critical applications with SLA at high resource utilization states.

The rest of the paper is structured as follows. Section II provides background into our experimental setup and Service Level Agreements. Section III introduces different soft resource allocation strategies and evaluates their effects on performance and profit and gives an algorithm to optimally allocate soft resources. Section IV summarizes related work and Section VI concludes our paper.

II. Background

A. Experimental Setup

We run an n-tier benchmark (RUBBoS) in Emulab testbed [2]. RUBBoS is a standard n-tier benchmark based on bulletin board applications such as Slashdot. The RUBBoS benchmark application can be implemented as three-tier (web server, application server, and database server) or four-tier (addition of clustering middleware such as C-JDBC [7]) system. The workload consists of 24 different interactions such as view story. The benchmark includes two kinds of workload modes: browsing-only and read/write interaction mixes. Performance measurements (e.g., CPU utilization) are taken during the runtime period using SysStat at one second granularity. We use the functionality provided by JVM to monitor thread status in Java applications. To conveniently monitor the utilization of the DB connection pool, we let all servlets share a global DB connection pool instead of an individual connection pool for each servlet.

Figure 1 outlines the choices of software components, hardware node, and a sample network topology used in our experiments. The experiments were carried out by allocating a dedicated physical node to each server. We use a four-digit notation #W/#A/#C/#D to denote the number of web servers, application servers, clustering middleware servers, and database servers. A sample topology of experiments is shown in Figure 1(c). In our experiments, we focus on how the allocation of soft resources affects n-tier system performance. Thus, we change the allocation of those soft resources by changing thread pool size in Apache servers, the thread pool and DB connection pool size in Tomcat servers. For each hardware provisioning #W/#A/#C/#D, we use #WT-#AT-#AC to represent the thread pool size in web server, the thread pool size in application server, and the DB connection pool size in application server. For example, the hardware provisioning can be 1/2/1/2. The corresponding soft resource allocation #WT-#AT-#AC can be 200-100-100, which means the thread pool size in a web server, the thread pool size and the DB connection pool size in each application server is 200, 100, 100, respectively. The allocation of other soft resources are fixed in order to limit the exponential experiment space.

B. Profit Model

In e-commerce applications, the response time for users can directly impact the profit for that application. Amazon found that for every 100ms a page took to load, it loses 1% in sales. Akamai reported that 40% of users will abandon a site that takes more than 3 seconds to load and 47% expects a website to load in 2 seconds or less. Typically in cloud environments, service level agreements (SLAs)

are used by the service provider to outline its revenue model to the customer and includes the earnings of the provider which includes revenue during SLA compliance and penalties during SLA violations.

In our previous work, we studied a simplified SLA model solely based on a single response time threshold to illustrate the profit trade-offs between throughput and response time [15]. We showed that increasing throughput without regards to other factors can actually lead to decreased provider revenue due to the lower revenues incurred by high response times. In this simplified SLA model, requests that had response times below or equal to the set threshold is termed **goodput**. Requests with higher response time than the threshold is **badput**. The traditional definition of throughput is then the sum of goodput and badput.

$$rev(rt_i) = \begin{cases} v & \text{if } 0 \leq rt_i \leq t_1 \\ v - c_1 & \text{if } t_1 < rt_i \leq t_2 \\ \dots & \\ v - c_n & \text{if } t_n < rt_i \leq t_{n+1} \\ p & \text{otherwise} \end{cases} \quad (1)$$

$$\text{provider revenue} = \sum_i rev(rt_i) \quad (2)$$

In this paper, we present an enhanced SLA profit model which considers multiple response time thresholds, with each response time interval generating a different profit value (see Equation 1). In this equation, rt_i means the response time of the ith request, $t_1, t_2, ...t_n$ represent response time boundaries. Each processed request generate value v by default; the earning of the request is v deducted by c_i, which is the late charge of the request. Once the response time rt_i exceeds a certain threshold, a penalty p is charged for the request. The final revenue of the service provider is summation of the earnings of all the processed requests (see Equation 2).

The enhanced profit model holistically emphasizes both the positive contribution of queries with short response times (typically within a fraction of a second) and the penalties of queries with very long response times (e.g., more than 2 seconds). This profit model is appropriate for many e-commerce applications which sees variations in profit depending on request response times (e.g., Amazon). Table I illustrates an instance of the enhanced profit model. In the rest of the paper we will use this model to conduct our profit-based performance analysis. We note that in this model, the service provider incurs a penalty in profit when a request's response time is greater than two seconds. This penalty may be considered harsh since it is greater than the maximum revenue, but it is actually quite conservative. In real applications such as Rackspace [3], the penalty incurred on the provider is much higher (e.g. Rackspace incurs a 5% penalty in monthly profits for every 30 minutes of downtime which equates to 0.068% of the month).

Response time interval	Revenue/Penalty
[0s, 200ms]	$0.27
(200ms, 500ms]	$0.20
(500ms, 1s]	$0.13
(1s, 2s]	$0.07
$>$ 2s	-$0.33

Table I: A sample profit model used in our experiments.

III. Profit Analysis of Soft Resource Allocations

A. Impact of Soft Resource Allocations Using Different Performance Models

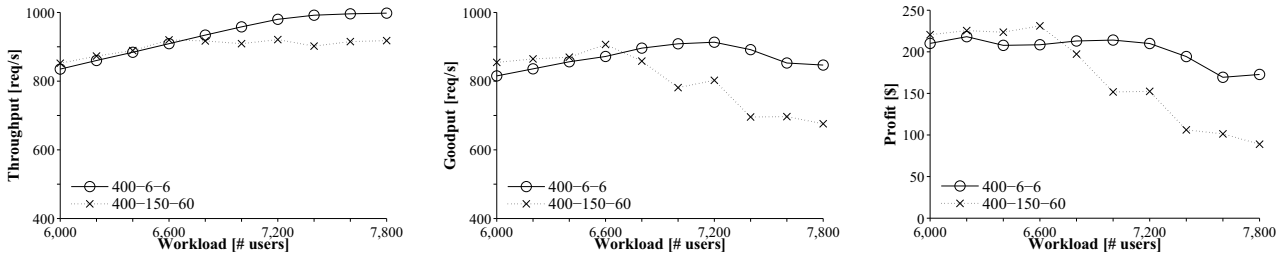
Soft resources refer to system software components that use hardware resources (CPU, memory, disk and network) or synchronizes the use of hardware resources [15]. For example, *threads* are soft resources that use CPU and *TCP connections* use network I/O. The goal of soft resources is to increase hardware utilization by creating a means to share them. Therefore, the allocation of soft resources can impact the efficiency of hardware utilization.

In this section, we evaluate the impact of soft resource allocations on system performance using three different performance models: the traditional throughput model, the goodput model, and our profit model (see Section II-B). Our results show that, while the performance impact of different soft resource allocations is insignificant using the traditional throughput model or even the goodput model, such impact becomes momentous once our profit model is applied. The hardware configuration of the experiments in this section is 1/4/1/4. We compare the performance of two different soft resource allocations (400-150-60 and 400-6-6). We note that the 400-150-60 allocation is considered a good choice by practitioners from industry.

Figure 2(a) shows the throughput comparison between the two soft resource allocations from workload 6,000 to 7,800. Such workload range depicts the plateau of the system throughput before it stops increasing. This figures shows that only until workload 6,800 these two soft resource allocation cases show the throughput difference. For example, the 400-6-6 case achieves only 8.0% higher throughput than that of the the 400-150-60 case at workload 7,800.

Figure 2(b) shows the goodput (with 1 second threshold) comparison for the same two soft resource allocations. In this figure, we see that the goodput for either of these two cases begins to decrease after certain workload; this is because of the increased number of badput as the workload increases while the system throughput keeps the same (due to saturation). At workload 7,800, the 400-6-6 case achieves 20.2% higher goodput than the the 400-150-60 case.

Figure 2(c) depicts the two soft resource allocations when applying our profit model (see Table I). Although the shape



(a) Throughput difference at WL 7,800 is 8.0%.

(b) Goodput(1s) difference at WL 7,800 is 20.2%.

(c) Profit difference using our profit model (see Table I) at the WL 7,800 is 48.6%.

Figure 2: Performance comparison between two soft resource allocations with 1/4/1/4 (400-x-y) configuration. The number 400 means the thread pool size in one web server; x and y means the Tomcat thread pool size and DB connection pool size respectively. Three different performance models are used: (a) throughput model, (b) goodput model, and (c) profit model. This figure shows the significant impact of soft resource allocations on system performance using our profit model.

of the two curves is similar to the goodput curves in Figure 2(b), note that the profits for the two allocations at workload 7,800 actually differ by 48.6%. Unlike goodput which only considers requests below a certain threshold as being valid, our profit model considers requests for all response times, but separates them out into five different response time intervals. Requests with lower response times are rewarded by having a higher profit, while requests with higher response times have lower profit and even a penalty on profit if the response time is greater than two seconds. Even though the throughputs and goodputs of the two allocations differ by 8.0% and 20.2% respectively, the economic difference between the two is a staggering 48.6%. Therefore, it is important to consider soft resource allocations not only in terms of throughput and goodput, but also profit.

In the next sections, we look at two more scenarios where conservative and liberal allocations of soft resources can be sub-optimal in terms of performance and profit.

B. Conservative Allocation of Soft Resources

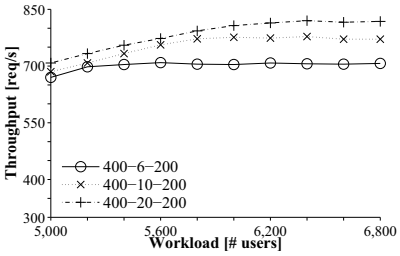
The first allocation strategy is a conservative strategy that aims to minimize the amount of software resources as to ensure that the system is not overloaded. Setting a small allocation of soft resources does mitigate the overhead caused by soft resources [15]; however, too few soft resources can limit software performance which creates a software bottleneck analogous to a hardware bottleneck when not enough resources are available.

In this experiment, we aim to create a software bottleneck by limiting the number of threads in the Tomcat thread pool. The thread pool in Tomcat is varied between 6 to 20 threads while the number of threads in Apache server and the number of DB connections is fixed to 400 and 200 respectively. We purposefully set the number of threads in Apache and number of DB connections to be large so that they will not be the system bottleneck. The hardware configuration we use is 1/2/1/2. Figure 3(a) shows the throughput of three software configurations. At every workload, decreasing the thread pool from 20 to 6 leads to decreases in throughput. This throughput decrease is due to lower hardware utilization when less threads are used; the exact reason for this is for small thread pool values in Tomcat, all threads are always being used even though the CPU is not fully utilized. For curious readers, we have analyzed this phenomenon in-depth in another paper [15]. Specifically, the system throughput decreases 13.6% when we decrease the number of Tomcat threads from 20 to 6.

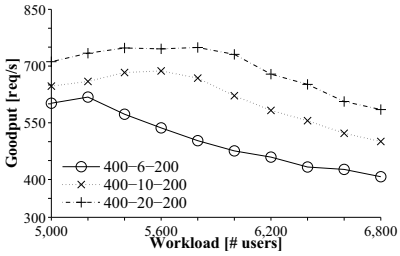
Even though the throughput decreases from the 400-20-200 to the 400-6-200 allocation, it might be overlooked because the percentage difference is only 13.6%. Figure 3(b) shows the goodput for all configurations for a goodput threshold of one second. Similar to throughput, at every workload, goodput decreases as the thread pool in Tomcat decreases; At workload 6,800, the system goodput decreases by 30.4% when we derease the thread pool size in Tomcat from 20 to 6, more than double the throughput percent decrease. This percent decrease is magnified even more when looking from a profit perspective.

Figure 3(c) shows the profit from each software configuration using our profit model. The profit model's curves have a similar shape to the curves in the goodput graph of Figure 3(b). Similar to the goodput comparison, the 400-6-200 soft resource allocation configuration is shown to have the lowest profit at the 6,800 workload while the 400-20-200 soft resource allocation configuration has the highest profit at workload 6,800. The values for the two are $12.44 and $81.83 respectively, a difference of 84.8%.

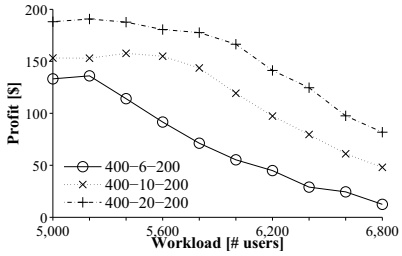
Figure 3(d) illustrates the performance degradation between the 400-6-200 and 400-20-200 configurations in terms of percent difference. The throughput percent difference between the two configurations slightly increases from 5.4% to 13.5% as workload increases; the goodput percent difference is more than double the throughput percent difference at every workload and goes from 15.4% to 30.4%. The profit model reveals the most astonishing difference between the



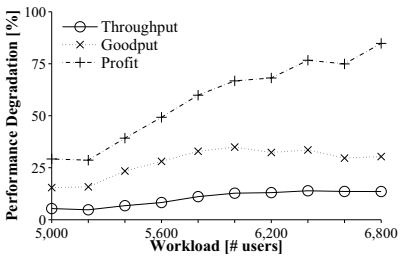
(a) Throughput decreases as Tomcat threadPool decreases from 20 to 6; throughput difference at WL 6,800 between the 6 thread case and the 20 thread case is only 13.6%.



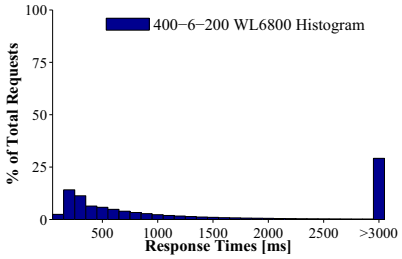
(b) Goodput(1s) decreases as Tomcat threadPool decreases from 20 to 6; goodput difference at WL 6,800 between the 6 thread case and the 20 thread case is 30.4%.



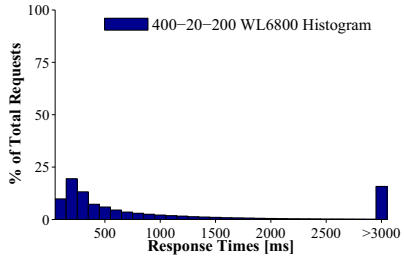
(c) Profit decreases as Tomcat threadPool decreases from 20 to 6; profit difference at WL 6,800 between the 6 thread case and the 20 thread case is a 84.8%.



(d) Performance degradation (in percentage) comparison among the three models at each workload after decreasing Tomcat threadPool from 20 to 6. The profit model shows much bigger degradation than the other two.



(e) Response time histogram for the 6 thread case at workload 6,800. This figure shows 32% requests with response time greater than 3s, which generate significant amount of negative profit in our profit model.



(f) Response time histogram for the 20 thread case at workload 6,800. This figure shows 18% requests with response time greater than 3s. which is significantly smaller than that of the 6 thread case.

Figure 3: Performance degradation due to conservative allocation of soft resources with 1/2/1/2 (400-x-200) configuration. x means the Tomcat thread pool size; we decrease x from 20 to 6.

two configurations; the profit difference ranges from 29.2% to 84.8%. For the same hardware configuration, there can be slight throughput and modest goodput degradation depending on how soft resources are allocated, but it can resultingly yield large differences in profit. The huge difference in profit can be explained by looking at each individual request.

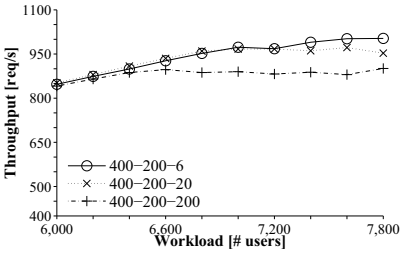
Figure 3(e) shows the response time histogram of the 400-6-200 configuration at workload 6,800. There is a large number of requests (approximately 164,000) that take longer than two seconds to process which accounts for 32% of the total number of requests in the experiment. Figure 3(f) shows the response time histogram of the 400-20-200 configuration at workload 6,800. The 400-20-200 configuration has a larger number of requests than the 400-6-200 configuration completed within two seconds and only 18% (approximately 110,000) of the total requests have a higher response time than two seconds. One interesting phenomenon to note is the spike in requests with a response time greater than 3 seconds for both configurations. Those requests are due to TCP retransmissions and we discuss this more in a future work. As workload increases, there will be a larger percentage of requests that timeout and are retransmitted, which causes our profit to peak and decrease after a certain workload.

C. Liberal Allocation of Soft Resources

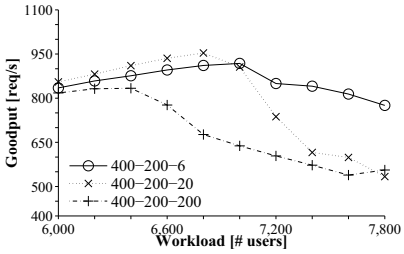
The second allocation strategy we employ is liberally setting a large capacity of software resources so that we ensure that the system is not being bottlenecked by software resources. This strategy illustrates the differences between hardware and software resources. Unlike hardware resources which do not consume resources when they are idle, software resources may still use system resource (e.g., CPU and memory) when they are idle; however, the cost of unused soft resources is usually considered to be negligible, so setting a high soft resource capacity when allocating resources is often reasonable. In our experiments, we found that setting a high soft resource allocation can lead to severe degradation of system performance.

In this experiment, we vary the size of the DB connection pool in Tomcat from 6 to 200 while keeping the thread pool in Apache and Tomcat fixed at 400 and 200 respectively. We also use a hardware configuration of 1/4/1/4.

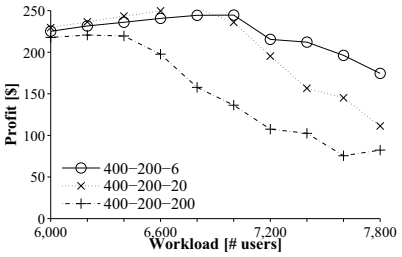
Figure 4(a) shows the throughput of three different soft resource allocations from workload 6,000 to 7,800. Note that the 400-200-200 configuration here is the same configuration used in the previous section; however, the hardware in this section has changed. At workload 7,800, the highest throughput is achieved with the configuration (400-200-6) with the lowest number of threads in the DB connection



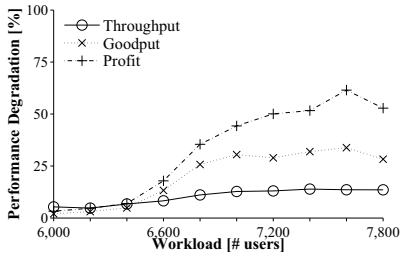
(a) Throughput decreases as Tomcat DBconn pool increases from 6 to 200; throughput difference at WL 7,800 between the 6 case and the 200 case is only 11.3%.



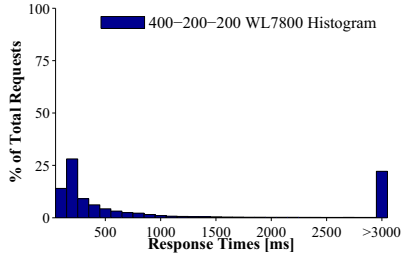
(b) Goodput(500ms) decreases as Tomcat DBconn pool increases from 6 to 200; goodput difference at WL 7,800 between the 6 case and the 200 case is 28.3%.



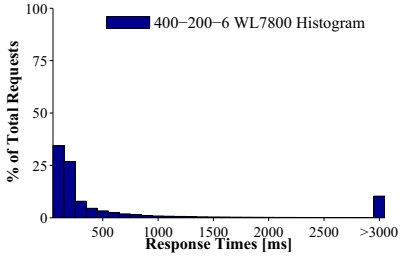
(c) Profit (using our profit model) decreases as Tomcat DBconn pool increases from 6 to 200; profit difference at WL 7,800 between the 6 case and the 200 case is 52.8%.



(d) Performance degradation (in percentage) comparison among the three models at each workload after increasing Tomcat DBconn from 6 to 200. The profit model shows much bigger degradation than the other two.



(e) The 200 DBconn case time histogram at workload 7,800. This figure shows 23.7% requests with response time greater than 3s. which which generate significant amount of negative profit in our profit model.



(f) The 6 DBconn case response time histogram at workload 7,800. This figure shows 11.4% requests with response time greater than 3s, which is significantly smaller than that of the 200 thread case.

Figure 4: Performance degradation due to liberal allocation of soft resources with 1/4/1/4 (400-200-x) configuration. x means the Tomcat DBconn pool size; we increase x from 6 to 200.

pool while the lowest throughput is achieved with the configuration (400-200-200) with the highest number of threads in the DB connection pool. At 7,800, the C-JDBC server's CPU becomes the bottleneck for configurations with a high DB connection pool number. This is due to the time the C-JDBC server spends for JVM garbage collection. A more in-depth analysis of this issue can be found in [15]. One interesting thing to note is that the 400-200-6 configuration does not achieve the highest throughput for all workloads. For workloads 6,000 to 6,800, the 400-200-20 configuration actually yields slightly greater throughput than the 400-200-6 configuration. This provides motivation for administrators to monitor the traffic to their system and to adjust the soft resource allocations in their systems accordingly.

We show the 500ms goodput for each configuration in Figure 4(b). At the 7,800 workload, similar to the throughput comparison, the 400-200-6 comparison yields the highest goodput out of all configurations. In addition, the 400-200-20 configuration again outperforms the 400-200-6 in terms of goodput for workloads between 6,000 to 6,800 users; however, interestingly, the 400-200-20 configuration takes a large dip after workload 6,800 and becomes the worst performing configuration at workload 7,800. Figure 4(c) shows the profit for the same configurations. The profit is highest for 400-20-20 between workloads 6,000 to 6,800. After workload 6,800, the 400-200-6 configuration achieves the highest profit. Similar to the previous profit comparison in Figure 3(c), each configuration has a peak profit before it decreases as workload increases.

Figure 4(d) shows the performance degradation between the 400-200-6 and 400-200-200 configurations. The degradation for throughput, goodput, and profit is almost equal for workloads 6,000 to 6,400. Starting from workload 6,600, we see greater performance degradation in terms of goodput and profit. At the 7,800 workload, the throughput, goodput, and profit percent difference between the two allocations is 13.6%, 28.3%, and 52.8% respectively. The performance degradation in this liberal allocation scenario mirrors the conservative allocation scenario where we see almost a double amplification of degradation from throughput to goodput and from goodput to profit.

Finally, Figure 4(f) and Figure 4(e) show the response time histograms of the 400-200-6 and 400-200-200 configurations. A majority of the requests are completed within 500ms for the 400-200-6 configuration and only 11.4% of the requests have response times greater than three seconds. In comparison, 23.7% of the total requests for the 400-200-200 configuration take longer than three seconds to process.

Algorithm 1: Pseudo-code for soft resource allocation algorithm to maximize profit.

```
1  procedure FindGoodSoftResource(workload, profitModel)
2  profit_new = 0, profit_max = -1;
3  S = S_0, H = H_0;
4  while profit_new > profit_max do
5      profit_max = profit_new;
6      (RT_dist) = runExperiment(H, S, workload);
7      profit_new = calculateProfit(RT_dist, profitModel);
8      if (profit_new > profit_max) then
9          /* find better allocation */
10         profit_max = profit_new;
11         S = 2S;
12     else
13         /* find best allocation */
14         return (profit_max, S/2);
15     end
16 end
```

D. Soft Resource Allocation Algorithm

So far we have shown that both conservative and liberal soft resource allocations in an n-tier system may lead to significant performance degradation, especially when applying a response time aware profit model. This is due to the fact that soft resources directly impact the efficiency of critical system resource usage (e.g., CPU) [1]; overly conservative soft resource allocations under-utilize the critical system resources while excessively liberal soft resource allocations waste critical system resources [15]. In this section we will discuss a practical iterative algorithm for a soft resource allocation that maximizes profit given a fixed hardware configuration and profit model.

Our algorithm assumes that the critical hardware resource of the system is known a priori [2] and mainly focuses on the iterative search process of soft resource allocations for profit maximization. Algorithm 1 shows the pseudo-code for the algorithm and a detailed description is given below.

The key idea of the algorithm is to iteratively increase the allocation of soft resources that directly use the critical system resource until reaching the maximum profit. The initial soft resource allocation and hardware provisioning are S_0 and H_0, respectively. Function $runExperiment(H, S, workload)$ initiates an experiment with input hardware/software configuration at a specific $workload$, and at the same time, the response times of all requests are recorded. Function $calculateProfit(RT_{dist}$,profitModel) calculates the profit given the response time distribution and predefined profit model. If the new profit $profit_{new}$ is larger than the previous maximum profit $profit_{max}$, we find a better soft resource allocation. In this case, the entire process is repeated by doubling the soft resource allocations. Otherwise the entire process ends and $profit_{max}$ is returned with the best soft resource allocations S. Although our paper focuses mainly on maximizing profit through better software resource allocation on the same hardware configuration, it is important to note that our algorithm can also adapt to finding a profit-maximizing hardware configuration if you substitute H for S in the equation's while loop. Our previous paper [10] has looked into finding the best hardware configuration through scale out to maximize profit.

[1]Critical system resource means the bottleneck resource of a system, which is the main determinant of system performance.

[2]This step can be done using the critical system resource algorithm introduced in [15].

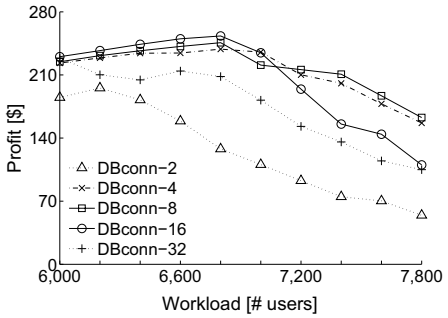


Figure 5: Searching of the "best" soft resource allocation for 1/4/1/4 (400-200-x) configuration using our algorithm. x means Tomcat DBconn pool size. This figure shows the best allocation of DB connections is workload dependent given our profit model; the DBconn-16 case outputs the highest profit in workload range 6,000 to 7,000 while the DBconn-8 case performs the best after workload 7,000.

Figure 5 shows the results of applying the algorithm to the previous 1/4/1/4 configuration case (see Section III-C). In this configuration, the critical system resource is the CJDBC CPU. Since the number of active threads in CJDBC (that directly use the CJDBC CPU) is controlled by the DB connection pool size in the Tomcat App tier, we run our algorithm for the "best" number of DB connections given the testing workload and our profit model. We note that we allocate a large number of threads in the Apache web tier and the Tomcat App tier in order to avoid hardware resource under-utilization [3]. This figure shows that the 16 DBconn case outputs the highest profit in the workload range from 6,000 to 7,000 while the 8 DBconn case outputs significantly higher profit than the other cases once the workload exceeds 7,000. The results show that the best allocation remains highly dependent on the workload in the evaluation.

IV. Related Work

The impact of soft resource allocations on system performance has been studied in [6], [14], [15]. Wang et al. [15] conducted experimental analysis of the impact of soft resource allocations on n-tier system throughput and goodput.

[3]Since the Apache web tier and the Tomcat App tier is not the bottleneck of the system, it is necessary to allocate more soft resources to buffer the large fluctuations in request traffic from clients [15].

Franks et al. [6] propose a layered queuing network which models the simultaneous soft/hardware resource possession in an n-tier system and develop a framework for systematic detection of bottlenecks caused by soft resources in the system. Urgaonkar et al. [14] propose a flexible queuing model to estimate how to allocate resources to each tier of an n-tier application by considering realistic factors (e.g., load imbalance, concurrency limits). All these previous works focus on the impact of soft resource allocations on traditional performance metrics (e.g., throughput) rather than profit. As shown in the paper, the differences between profit-based evaluation criteria and traditional performance metrics become significant at high resource utilization levels. For example, our results show that the profit-maximizing soft resource allocation is highly workload dependent.

Profit based performance analysis for distributed systems has been studied before. For example, Malkowski et al. [10] presented a configuration management tool which combines empirical data and SLA and find the best hardware configurations to maximize the profit. Xi et.al [5] proposed a profit model to describe revenues specified by the Service Level Agreement (SLA) and costs generated by leased resources. Lee et al. [9] discussed profit aware service request scheduling. Konstantinos et. Al [13] propose an interesting work which is based on two premises: the cloud-consumer always has a budget and physical hardware resources are limited. Different from these previous works, we mainly focus on soft resource allocations on profit maximization. More specifically, our work is complementary to the previous hardware-based profit maximization research.

Software misconfiguration of n-tier systems have been studied in [4], [8]. Attariyan et al. [4] present a tool that locates the root cause of configuration errors by applying dynamic information flow analysis within process (mainly) in the runtime. Kiciman et al. [8] describe Pinpoint, a methodology for automating fault detection in Web applications by identifying which components in a distributed system are most likely to be the cause of a fault. All these work differs from our work in that they focus more on faults or anomaly behavior of system caused by misconfiguration of software, rather than the performance problem.

V. Acknowledgement

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.

VI. Conclusion

Achieving simultaneously high resource utilization and guaranteed quality of service (QoS) is an important goal and significant challenge in cloud environments. Using a response-time based profit model, we showed that modest differences in traditional performance metrics (e.g., average throughput) often translate into much more significant differences in profits. This sensitive profit model showed the significant impact of different allocation policies in an experimental study of soft resource allocation impact on n-tier applications using the RUBBoS benchmark. Over-allocation of soft resources (thread pool size in a server) causes unnecessary resource consumption, and reduces profits when the resource is near saturation. Under-allocation of soft resources causes potential under-utilization of downstream servers, also reducing profits. Our data shows that the best soft resource allocation is highly dependent on several system parameters, including the workload distribution, system configuration, and profit model adopted in the evaluation. Our results show that the overall profits (economic goals) of a cloud environment should be taken into consideration explicitly when deciding on resource allocation (maximizing utilization) to optimize n-tier application performance (maximizing throughput while guaranteeing QoS).

References

[1] *Rice University Bulletin Board System*. "http://jmob.ow2.org/rubbos.html", 2004.

[2] *Emulab–Network Emulation Testbed*. "http://www.emulab.net/", 2010.

[3] *Rackspace–Hosting Provider with Industry Leading Service Level Agreement*. "http://www.rackspace.com/managed_hosting/support/servicelevels/managedsla/", 2012.

[4] M. Attariyan and J. Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *OSDI '10*.

[5] X. Chen, H. Chen, Q. Zheng, W. Wang, and G. Liu. Characterizing web application performance for maximizing service providers' profits in clouds. In *CSC '11*.

[6] G. Franks, D. Petriu, M. Woodside, J. Xu, and P. Tregunno. Layered bottlenecks and their mitigation. In *QEST '06*.

[7] E. C. Julie, J. Marguerite, and W. Zwaenepoel. *C-JDBC: Flexible Database Clustering Middleware*. 2004.

[8] E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks*, 2005.

[9] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou. Profit-driven service request scheduling in clouds. In *CCGrid '10*.

[10] S. Malkowski, D. Jayasinghe, M. Hedwig, J. Park, Y. Kanemasa, and C. Pu. Cloudxplor: A tool for configuration planning in clouds based on empirical data. In *SAC '10*.

[11] S. J. Malkowski, M. Hedwig, J. Li, C. Pu, and D. Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *ICAC '11*.

[12] B. Snyder. Server virtualization has stalled, despite the hype. In *Infoworld*, 2010.

[13] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis. Flexible use of cloud resources through profit maximization and price discrimination. In *ICDE '11*.

[14] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *SIGMETRICS '05*.

[15] Q. Wang, S. Malkowski, Y. Kanemasa, D. Jayasinghe, P. Xiong, C. Pu, M. Kawaba, and L. Harada. The impact of soft resource allocation on n-tier application scalability. In *IPDPS '11*.