Response Time Reliability in Cloud Environments: An Empirical Study of n-Tier Applications at High Resource Utilization

Qingyang Wang [1], Yasuhiko Kanemasa [2], Jack Li [1], Deepal Jayasinghe [1], Motoyuki Kawaba [2], Calton Pu [1]
[1]College of Computing, Georgia Institute of Technology
[2]Cloud Computing Research Center, FUJITSU LABORATORIES LTD.
[1]{qywang, jack.li, deepal, calton}@cc.gatech.edu
[2]{kanemasa, kawaba}@jp.fujitsu.com

***Abstract*—When running mission-critical web-facing applications (e.g., electronic commerce) in cloud environments, predictable response time, e.g., specified as service level agreements (SLA), is a major performance reliability requirement. Through extensive measurements of n-tier application benchmarks in a cloud environment, we study three factors that significantly impact the application response time predictability: bursty workloads (typical of web-facing applications), soft resource management strategies (e.g., global thread pool or local thread pool), and bursts in system software consumption of hardware resources (e.g., Java Virtual Machine garbage collection). Using a set of profit-based performance criteria derived from typical SLAs, we show that response time reliability is brittle, with large response time variations (order of several seconds) depending on each one of those factors. For example, for the same workload and hardware platform, modest increases in workload burstiness may result in profit drops of more than 50%. Our results show that profit-based performance criteria may contribute significantly to the successful delimitation of performance unreliability boundaries and thus support effective management of clouds.**

***Keywords*-performance reliability; response time prediction; n-tier; web application; profit model**

I. Introduction

When running mission-critical web-facing applications (e.g., electronic commerce) in cloud environments, one of the major requirements is predictable response time, often specified as service level agreements (SLA) such as "95% of queries will return meaningful results within 1 second." More generally, we use the term *performance reliability* to refer to the quality of service properties that denote predictable or guaranteed performance such as SLA-specified response time. Traditionally, performance reliability is a given (e.g., in queuing theory-based performance analysis) when a stable workload runs on a stable hardware and software configuration. These stability assumptions no longer hold for web-facing applications running in a cloud, introducing serious challenges for performance reliability requirements such as SLA-specified response time.

In this paper, we describe an empirical study of performance reliability focused on response time predictability, through extensive measurements of n-tier application benchmarks in a cloud environment. Our study shows that a variety of factors may affect the system performance reliability. Concretely, we observe evidence of significant variations of application response time caused by three different factors: bursty workloads typical of web-facing applications, soft resource management strategies (e.g., thread pool management), and bursts in system software consumption of hardware resources (e.g., JVM garbage collection).

The first contribution of the paper consists of this experimental demonstration of the brittleness of cloud environments with respect to performance reliability due to the variety of factors that may affect response time. This performance unreliability was observed when the nodes reach relatively high resource utilization levels. Thus our results provide an explanation for the anecdotal and reported low average cloud utilization (e.g., 18% in [6]). Low utilization is often effective in preserving quality of service objectives specified in SLAs when no better knowledge exists. Generally, with so many factors at different system levels that may affect response time, it is indeed difficult to preserve SLAs at even modestly high utilization levels.

The second contribution of the paper is an empirical evaluation of n-tier web-facing application performance reliability using a set of profit-based performance criteria derived from typical SLAs [9]. The profit model emphasizes both the positive contribution of queries returned with good response time and the penalties of very long response times (e.g., over 2 seconds). Compared to classic performance metrics, the profit-based evaluation criteria are more sensitive at predicting performance unreliability at high resource utilization levels. For example, despite a relatively flat average throughput curve, rapid deterioration of response time can quickly reduce profits at well-defined high resource utilization levels. More specifically, for the same workload and hardware platform, modest increases in workload burstiness may result in profit drops of more than 50% (see Figure 3). Consequently, our results provide strong evidence that profit-based performance criteria may contribute effectively to our ability to delimit the regions of performance unreliability.

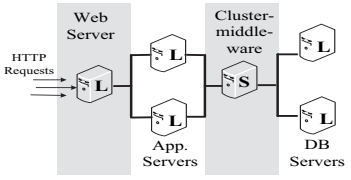
Given the high economic interest in achieving high utilization of cloud resources, the results described in this paper are significant in three ways. First, the various factors that

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	RHEL Server 5.7 (Tikanga)
System monitor	Sysstat 10.0.02, Collectl 3.5.1
Transaction monitor	Fujitsu SysViz

(a) Software setup

Hardware	Processor # cores	Freq.	L2 Cache	Memory	Disk	Network
Large (L)	2	2.27GHz	2M	2GB	200GB	1Gbps
Medium (M)	1	2.4 GHz	4M	2GB	200GB	1Gbps
Small (S)	1	2.26GHz	512k	1GB	80GB	1Gbps

(b) Hardware node setup



(c) 1L/2L/1S/2L sample topology

Figure 1: Details of the experimental setup.

affect performance reliability illustrate the importance of this research, since these factors are non-trivial roadblocks. Second, our work points to significant future research opportunities and challenges, since these factors may be scattered around in many components and layers. Third, we show concrete ways to get around these roadblocks by delimiting the boundaries of regions of performance unreliability through the adoption of profit-based performance evaluation criteria. In summary, we can see a viable path towards achieving high utilization of cloud resources *and* high quality of service simultaneously, but significant hard work and challenging research remain.

The rest of the paper is structured as follows. Section II shows a motivating example of the performance reliability evaluation at high resource utilization. Section III, V, and IV illustrate three factors that affect the system performance reliability. Section VI concludes our paper.

II. Background and Motivation

A. Background

1) Experimental Setup: We adopt the RUBBoS benchmark, based on bulletin board applications such as Slashdot.org [4]. RUBBoS can be configured as a three-tier (web server, App. server, and DB server) or four-tier (with C-JDBC [5]) system. The benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes. We use browse-only workload in this paper.

Figure 1 outlines the details of the experimental setup. We conduct the experiments by allocating a dedicated physical node to each server. A four-digit notation $\#W/\#A/\#C/\#D$ is used to denote the number of web servers, application servers, clustering middleware servers, and database servers. We have two types of hardware nodes: "L" and "S", each of which represents different processing power. Figure 1(c) shows a sample four-tier topology (1L/2L/1S/2L). We use Fujitsu SysViz [3] as a transaction monitor to precisely measure the response time and the number of concurrent requests in each short time window (e.g., 100ms) with respect to each tier of an n-tier system.

2) Profit Model: In e-commerce applications, the reliability of response time for users directly impact the profit of the corresponding service providers. Experiments at Amazon show that every 100ms increase in the page load decreases sables by 1% [7]. In practice, a service provider usually uses service level agreements (SLAs) to outline its profit model which includes earnings during SLA compliance and penalties during SLA violations.

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

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

Equation 1 presents an simple SLA profit model which considers multiple response time thresholds, with each response time interval generating a different profit value. In this equation, rt_i represents the ith request response time, $t_1, t_2, ...t_n$ represent response time boundaries. Each processed request by default generates value v; the earning of the request is v deducted by c_i, which is the late charge of the request. A penalty p is charged for the request once the response time rt_i exceeds a certain threshold. The final revenue of the service provider is summation of the earnings of all the processed requests (see Equation 2). Table I shows an instance of the profit model, which is from industry.

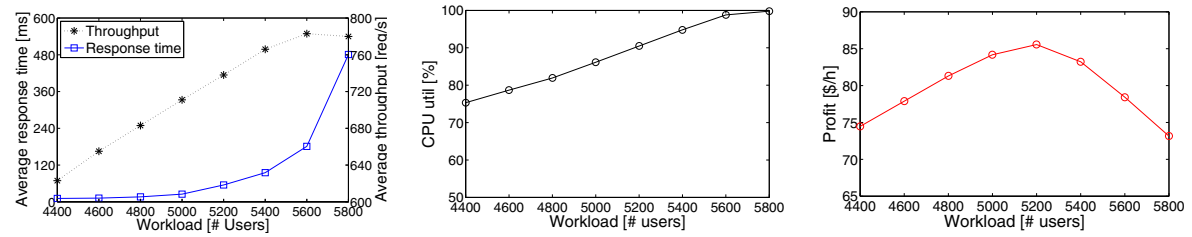
Response time interval	Revenue/Penalty
[0s, 200ms]	$0.0020
(200ms, 500ms]	$0.0015
(500ms, 1s]	$0.0013
(1s, 2s]	$0.0005
$>$ 2s	-$0.0047

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

B. Motivation

To illustrate the importance of the problem, we apply our profit model to our data and show that system performance is unreliable when there are large response time fluctuations. The results shown here are based on three-minute runtime experiments running in a four-tier system (see Figure 1(c)).

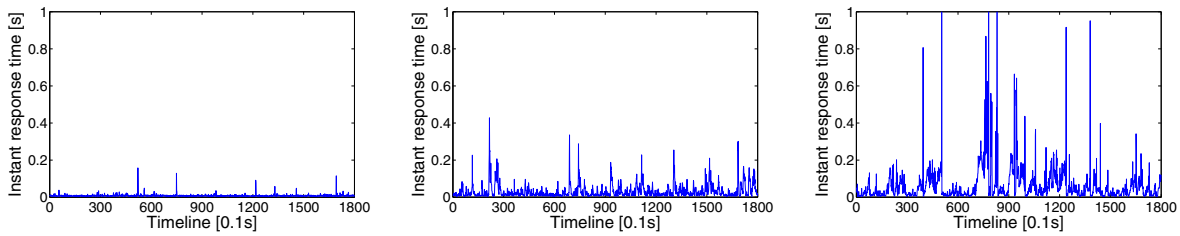
Figure 2(a) shows the average response time and throughput of the system from workload 4400 to 5800 (concurrent user sessions). This figure shows that the system is only saturated at workload 5600 since the throughput stops increasing as further workload increases. This figure also shows that



(a) Average response time (RT) and throughput.

(b) Bottleneck server (CJDBC) CPU utilization under each workload.

(c) Profit (based on our profit model) starts to decrease after workload 5200.



(d) Small RT fluctuations in workload 4400 (average RT 0.012s).

(e) Mediam RT fluctuations in workload 4800 (average RT 0.025s).

(f) Large RT fluctuations in workload 5200 (average RT 0.068s).

Figure 2: Performance unreliability due to large response time fluctuations of the system with 1L/2L/1S/2L configuration. This figure shows that as workload increases, the bottleneck resource utilization (see Subfigure (b)) increases, which leads to more fluctuations of system response time (see Subfigures (d)(e)(f)); thus the profit (see Table I) eventually drops after workload 5200 (see Subfigure (c)).

the average system response time is low before workload 5600. However, Figure 2(c) shows that the service provider's profit based on our profit model already starts to drop after workload 5200 even though the average response time is low and the throughput has not reached the peak value. We note that the average utilization of the bottleneck resource (CJDBC CPU) in the system is just 90% at workload 5200 (see Figure 2(b)).

Figure 2(d), 2(e), and 2(f) show the instant response time (average in every 100ms) of the system at workload 4400, 4800, and 5200, respectively. Compared to Figure 2(d) and Figure 2(e), Figure 2(f) shows that the system has large response time fluctuations even though the average response time is only 0.068s. In fact our analysis for individual request response time shows that many requests took more than 2 seconds to complete. Based on our profit model as shown in Table I, requests with the long response time significantly degrade the service provider's profit. Thus, the reason why system profit starts to drop after workload 5200 (see Figure 2(c)) is clear.

In general, profit models combine economic goals with technical performance data to derive intuitive domain insights at the actual performance reliability provided by service providers. In the following Sections III, IV, and V we will explain three factors for the performance unreliability of an n-tier system at high resource utilization.

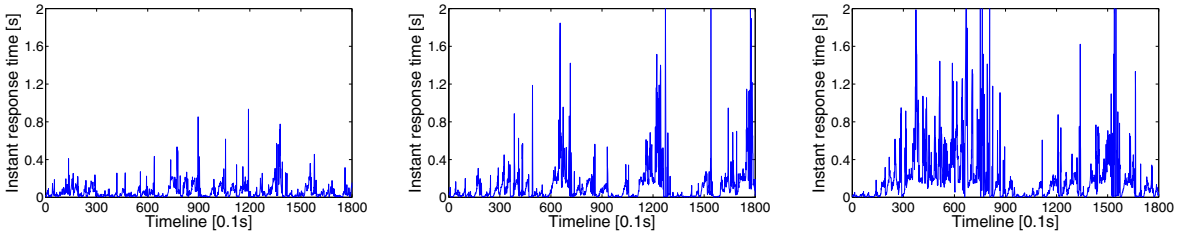
III. Bursty Workload

In this section we show how a bursty workload impacts the system performance reliability even though the average resource utilization of the system is far from saturation.

Bursty workload is very common in real world n-tier web-facing applications. For instance, the popular term *Slashdot effect* describes a phenomenon where a web page linked by a popular blog or media site suddenly experiences a huge increase in web traffic [1]. Mi et al. [10] proposed a bursty workload generator which takes into account the Slashdot effect. The bursty workload generator uses one parameter to characterize the intensity of the traffic surges: *index of dispersion*, which is abbreviated as I. The larger the I is, the longer the duration of the traffic surge. In this section, we use the bursty workload generator (with I = 100, 400, and 1000) to evaluate the system performance reliability under bursty workload conditions. The burstness level of the original RUBBoS workload generator is I = 1.

Figure 3 shows the approximately instant response time (average in every 100ms) of the system under workload 4800 with different burstiness levels. The hardware configuration is 1L/2L/1S/2L (see Figure 1(c)). Although the workload burstiness levels among the three cases are different, the average CPU utilizations of the bottleneck server (CJDBC) are the same (86%). Figure 3(a), 3(b), and 3(c) show that as the burstiness level of workload increases, the instant response time of the system fluctuates more significantly, which indicates less reliability of system performance.

In order to manage performance reliability, system administrators may want to know at which workload the system performance starts becoming unreliable due to the large response time fluctuations. Table II shows the minimum workload (with different burstiness levels) beyond which the service provider's profit starts to drop. This table shows that



(a) I=100; mean RT = 0.063s, standard deviation = 0.081s, the profit is 80.7 $/h.

(b) I=400; mean RT = 0.134s, standard deviation = 0.226s, the profit is 70.9 $/h.

(c) I=1000; mean RT = 0.222s, standard deviation = 0.349s, the profit is 53.4 $/h.

Figure 3: Response time fluctuations of the system (1L/2L/1S/2L configuration, see Figure 1(c)) in workload 4800 with different burstiness levels; the average CPU utilization of the bottleneck server (CJDBC) is 86% in three-minute runtime experiments for all the three cases.

Burstiness level	Threshold workload	CJDBC CPU util.
I =1	5200	90.6%
I =100	5000	87.3%
I =400	4600	83.6%
I =1000	3800	74.6%

Table II: Workload (with different burstiness levels) beyond which the profit based on our profit model starts to decrease.

both the threshold workload and the corresponding average CPU utilization of the bottleneck server decrease as the burstiness level of workload increases. This result further justifies that the evaluation of performance reliability using a profit model is an important and necessary step in designing and managing reliable n-tier systems.

IV. Soft Resource Management Strategies

Performance unreliability can be caused by soft resource management strategies in an n-tier system. *Soft resources* refer to system software components (e.g., threads, database connections) that use hardware resources (e.g., CPU) or synchronizes the use of hardware resources [8]. In general there are two strategies to manage soft resources: use a global pool (e.g., thread pool in Tomcat) or multiple pools (e.g., an individual database connection pool for each servlet in Tomcat). Although a global pool reduces the runtime management cost of soft resources (e.g., shrink or increase the pool size), it may significantly delay the overall request processing when the system is at high resource utilization.

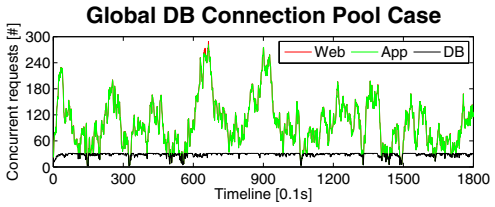
We use the database connection pool in Tomcat App tier to illustrate how soft resource management strategies impact the system performance reliability. The experiments described here have two DB connection management strategies in Tomcat: each servlet has its own local database connection pool with pool size 2, or all servlets share one global database connection pool with pool size 16 [1]. The hardware configuration is 1L/2L/1L where the MySQL CPU is the bottleneck resource in the system.

Our measurements show that when the system is at high resource utilization, the global database connection pool case causes significant response time fluctuations. This is because, in the global connection pool case, the requests for light transactions of the system are largely delayed by the requests for heavy transactions, which in turn increases the overall waiting time of request processing in the system [11]. A transaction services an entire web page requested by a client and may include multiple interactions between different tiers in a n-tier system. For an RPC style n-tier system, a database connection in the App server tier is occupied most of the time during an transaction processing period, regardless of heavy transactions or light transactions. Thus, the database connections in the global pool case are likely to be filled by heavy transactions when the system is heavily utilized. In this case the requests for light transactions frequently have to wait for available database connections in the App server tier. On the other hand, since each servlet has its own database connections in the local pool case, the requests for light transactions do not need to compete for DB connections with the requests for heavy transactions; the corresponding waiting time is significantly reduced.

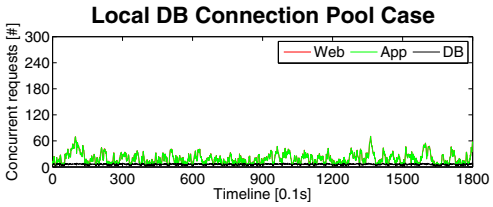
Figure 4 shows the response time fluctuations of the system in workload 5600 for both the two cases. Figure 4(a) and Figure 4(b) show the number of concurrent requests in each tier of the three-tier system for these two cases. These two figures show that the fluctuations of the number of concurrent requests in the Tomcat App tier and the Apache web tier are much higher in the global pool case than those in the local pool case. This is because in the global pool case the database connections in the Tomcat tier are frequently occupied by heavy requests (e.g., ViewStory requests), which makes a large amount of light requests wait in the Tomcat App tier. Figure 4(c) and Figure 4(d) show that the instant response time in the global pool case has much larger fluctuation than that in the local pool case. The high peaks of response time in Figure 4(c) and Figure 4(d) perfectly match the high peaks of the number of stacked requests in upper tiers as shown in Figure 4(a) and 4(b), which indicates that the long waiting time in upper tiers causes the large fluctuation of the end-to-end response time.

Table III shows the service provider's profit at high resource utilization for both the two cases. This table shows

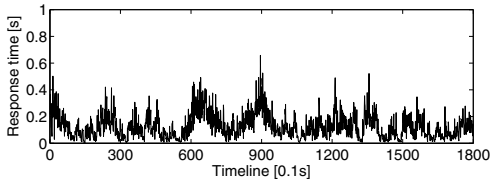
[1]There are eight active servlets for RUBBoS browse-only workload.



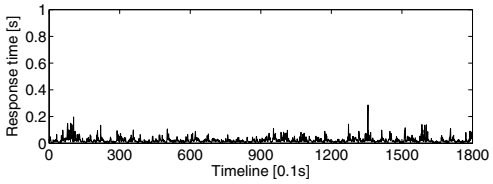
(a) Instant # of concurrent requests in each tier



(b) Instant # of concurrent requests in each tier



(c) Instant end-to-end response time



(d) Instant end-to-end response time

Figure 4: Response time fluctuations under two soft resource management strategies with 1L/2L/1L configuration in workload 5600.

Workload	Global ConnPool		Local ConnPool	
	CPU util. [%]	Profit [$/h]	CPU util. [%]	Profit [$/h]
4600	86.4	78.4	86.0	78.5
5200	95.8	86.7	94.3	86.5
5400	98.9	*86.6*	96.8	*89.9*
5600	99.9	*83.9*	98.2	*93.5*
5800	99.9	*68*	99.5	*85.9*

Table III: The global DBconn case shows more reliable performance than the local DBconn case.

that the profit in the global pool case starts to drop at workload 5200 while the profit in the local pool case starts at workload 5600; at workload 5800, the profit difference between these the global pool case and the local pool case is 26.3%. These results show that the system performance in the former case is less reliable than that in the latter case.

V. Bursts in System Software Consumption of Hardware Resources

Performance unreliability can be caused by bursts in system software consumption of hardware resources. In this section we will use JVM garbage collection (GC) in Java-based servers as an illustration example.

The JVM GC process impacts the system response time fluctuations in two ways. First, GCs in JVM consumes critical system resources such as CPU for cleaning garbage in the memory while such resources could be used for request processing. Second, a GC is a synchronous process and it stops the actual request processing until the garbage collection is finished [2]. This delay significantly lengthens the pending requests and causes response time fluctuations.

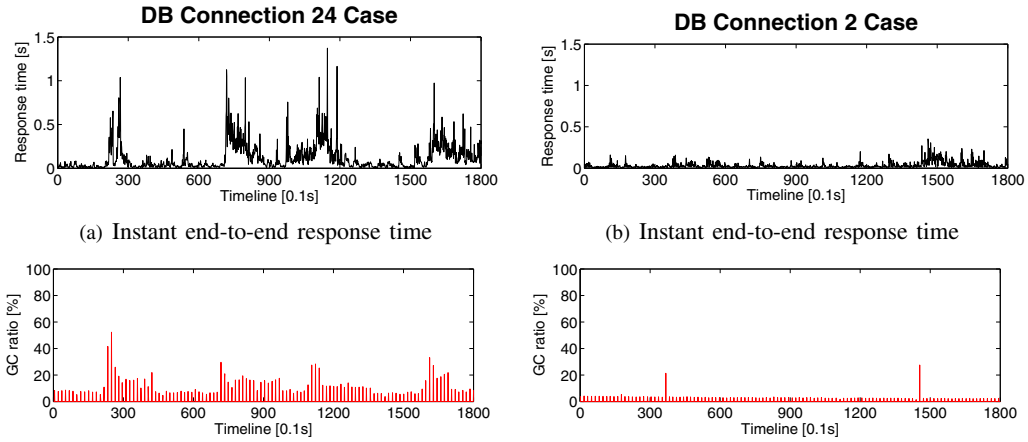
Our measurements show that the JVM GC time of a Java-based server increases non-linearly as workload increases when the server is at high resource utilization. The hardware configuration of the experiments in this section is 1L/2L/1S/2L (see Figure 1(c)) and the CJDBC CPU is the bottleneck resource of the system. We note that CJDBC is a Java-based middleware; each DB connections in Tomcat corresponds to one threads in CJDBC. Thus the larger the size of the database connection pool in the Tomcat tier, the more threads will be created in CJDBC.

Figure 5 shows the total JVM garbage collection time of the CJDBC server during the three-minute run-time experiments under two different database connection allocations. The DBconn24 case means the database connection pool size for each servlet in Tomcat is 24 while the DBconn2 case means 2 for each servlet. This figure shows the DBconn24 case takes a longer time for garbage collection than the DBconn2 case under higher workload range. The reason is because the CJDBC server consumes more memory to maintain a higher number of active threads in the DBconn24 case than in the DBconn2 case at high resource utilization, which leads to more GCs in the DBconn24 case.

Figure 6(a) and Figure 6(b) show the instant end-to-end response time for the two database allocation cases under workload 5500. Figure 6(c) and Figure 6(d) show the corresponding timeline graphs of JVM GC ratio [2] in the CJDBC server. The GC peaks in these two figures match very well with the large response time peaks as shown in Figure 6(a) and Figure 6(b), which indicates that the JVM GCs actually cause large response time fluctuations. Since the DBconn2 case performs less JVM GCs than the DBconn24 case, the response time is relatively stable.

Table IV shows the service provider's profit at high resource utilization for both the two cases. This table shows that the profit in the DBconn24 case starts to drop earlier (at workload 5200) than the profit in the DBconn2 case (at workload 5500), indicating that the system performance in the DBconn24 case is less reliable than that in the DBconn2 case at high resource utilization.

[2]JVM GC ratio means the ratio of JVM GC duration over each monitoring time window (0.1 second in our case). JVM provides a logging function recording the starting/ending timestamps of each GC activity.



(a) Instant end-to-end response time

(b) Instant end-to-end response time

(c) JVM GC ratio on CJDBC, DBconn24 case

(d) JVM GC ratio on CJDBC, DBconn2 case

Figure 6: Response time fluctuations caused by JVM garbage collection with the 1L/2L/1S/2L configuration in workload 5500. The DBconn24 case shows larger response time fluctuations than the DBconn2 case, which indicates less reliability in system performance.

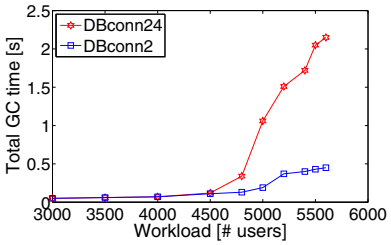


Figure 5: The JVM GC time of the DBconn24 case increases faster than the DBconn2 case under high workload range.

Workload	DBconn24		DBconn2	
	CPU util. [%]	Profit [$/h]	CPU util. [%]	Profit [$/h]
5000	86.1	84.2	84.8	84.2
5200	90.5	*85.6*	87.4	*87.3*
5400	94.2	*83.2*	91.1	*90.8*
5500	98.3	*81.1*	93.4	*91.6*
5600	98.8	*78.4*	95.4	*89.1*

Table IV: The DBconn24 case shows more reliable performance than the DBconn2 case.

VI. Conclusion

Achieving simultaneously high resource utilization and reliable application performance is an important goal and significant challenge in cloud environments. We conducted an empirical study of a variety of factors (Section III IV V) that significantly impact the application performance reliability (response time predictability). We evaluated the performance reliability using a profit model derived from typical SLAs (Section II) for each factor that may cause the significant response time variations at high resource utilization. Our results provide strong evidence that profit-based performance criteria may contribute effectively to our ability to delimit the regions of performance unreliability of large-scale distributed applications running in cloud.

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.

References

[1] S. Adler. The Slashdot Effect: An Analysis of Three Internet Publications. "http://jmob.ow2.org/rubbos.html", 2004.

[2] J2SE 6 Perf. White Paper. http://java.sun.com/performance/reference/whitepapers/6_performance.html, 2010.

[3] Fujitsu SysViz: System Visualization "http://www.google.com/patents?id=0pGRAAAAEBAJ&zoom=4&pg=PA1#v=onepage&q&f=false", 2010.

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

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

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

[7] K. Ron and L. Roger. Online Experiments: Lessons Learned. In *IEEE Computer 2007*.

[8] 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*.

[9] 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*.

[10] N. Mi, C. Casale, L. Cherkasova, and E. Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *ICAC '09*.

[11] Q. Wang, Y. Kanemasa, C. Pu, and M. Kawaba. When Average is Not Average: Large Response Time Fluctuations in n-Tier Systems. In *ICAC '12*.