

# RAPIDLY ALTERNATING BOTTLENECKS: A STUDY OF TWO CASES IN N-TIER APPLICATIONS

<sup>1</sup>Qingyang Wang, <sup>2</sup>Yasuhiko Kanemasa, <sup>1</sup>Jack Li, <sup>2</sup>Toshihiro Shimizu, <sup>2</sup>Masazumi Matsubara, <sup>3</sup>Motoyuki Kawaba, <sup>1</sup>Calton Pu

<sup>1</sup>College of Computing, Georgia Institute of Technology

<sup>2</sup>System Software Laboratories, FUJITSU LABORATORIES LTD.

<sup>3</sup>ICT Systems Laboratories, FUJITSU LABORATORIES LTD.

{qywang, jack.li, calton}@cc.gatech.edu, {kanemasa, shimizut, matz, kawaba}@jp.fujitsu.com

## Abstract

Identifying the location of performance bottlenecks is a non-trivial challenge when scaling n-tier applications in computing clouds. Specifically, we observed that an n-tier application may experience significant performance loss when bottlenecks alternate rapidly between component servers. Such rapidly alternating bottlenecks arise naturally and often from resource dependencies in an n-tier system and bursty workloads. These rapidly alternating bottlenecks are difficult to detect because the saturation in each participating server may have a very short lifespan (e.g., milliseconds) compared to current system monitoring tools and practices with sampling at intervals of seconds or minutes. Using passive network tracing at fine-granularity (e.g., aggregate at every 50ms), we are able to correlate throughput (i.e., request service rate) and queue length (i.e., number of concurrent requests) in each server of an n-tier system. Our experimental results show conclusive evidence of rapidly alternating bottlenecks caused by system software (JVM garbage collection) and middleware (VM collocation).

**Keywords:** [bottleneck, n-tier, performance analysis, scalability, web application]

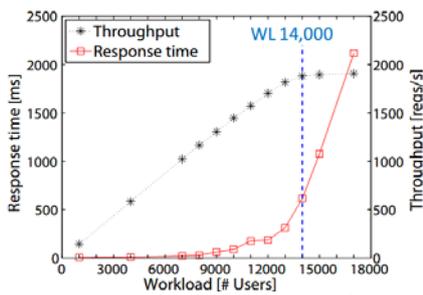
## 1. INTRODUCTION

Web-facing enterprise applications such as electronic commerce are not embarrassingly parallel (e.g., web indexing and data analytics). They are typically implemented using an n-tier architecture with web server, application server, and database server tiers. Such n-tier applications have implicit dependencies among their components, which create *alternating bottlenecks* (Balbo & Serazzi, 1997; Casale & Serazzi, 2004; Malkowski, Hedwig, & Pu, 2009; Mi, Casale, Cherkasova, & Smirni, 2008). These alternating bottlenecks are both interesting and challenging. They are interesting because they cause the entire n-tier system to reach its performance limit (i.e., flat throughput) even though all system resources are measurably below 100% utilization. They are challenging because classic queuing models that assume independent jobs predict single resource saturation bottlenecks, so they are inapplicable to alternating bottlenecks.

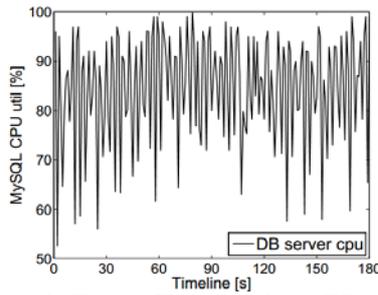
The main hypothesis of this paper is that (contrary to previously common belief) alternating bottlenecks occur naturally in real application scenarios and they can be found by experimental measurements using appropriate tools. Alternating bottlenecks constitute an important problem because there is lingering skepticism about their prevalence (and even existence) in the real world, despite early theoretical predictions (Balbo & Serazzi, 1997; Casale & Serazzi, 2004; Mi et al., 2008). In the past, observed evidence of alternating bottlenecks was rare and it was not easy to reproduce them reliably in experiments. We report consistent experimental results which suggest that

alternating bottlenecks may be far more common than previously believed. The perception of rarity is simply due to many alternating bottlenecks being short-lived (on the order of tens of milliseconds). Consequently, these interesting phenomena have been (and still are) completely invisible to normal monitoring tools that sample at time intervals measured in seconds or minutes.

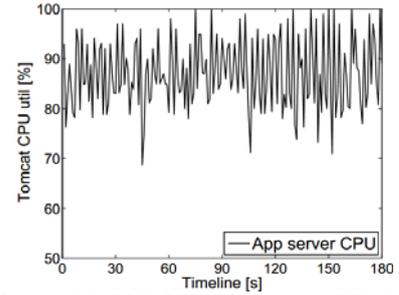
The main contribution of the paper is an unequivocal confirmation of our hypothesis through reproducible experimental observation of two rapidly alternating bottlenecks when running the standard n-tier RUBBoS benchmark ("RUBBoS: Bulletin board benchmark," 2004). Specifically, we found that bottlenecks alternate between the Tomcat tier and the MySQL tier at time interval of tens of milliseconds. Our study further shows that alternating bottlenecks can be caused by factors at the software level (e.g., JVM garbage collection, see Section 4) and middleware level (e.g., VM collocation, see Section 5). Despite its relatively short duration, the impact of this alternating bottleneck becomes significant when the frequency and intensity of the alternating pattern increase. The detection of alternating bottlenecks required a novel method that differs from traditional bottleneck detection in two main aspects. First, since alternating bottlenecks may arise without any single resource saturation, our method is completely independent of any single resource saturation measurements. Concretely, Section 2.2 shows an example in which the throughput of a four-tier system stops increasing even though the highest resource utilization in the system (MySQL CPU) is only 86.9%. Second, our



(a) Average response time and throughput at each workload



(b) Tomcat CPU utilization at WL14,000; the average is 86.9%.



(c) MySQL CPU utilization at WL14,000; the average is 84.3%.

Figure 1: A rapidly alternating bottleneck case of a 4-tier system with Browse-only workload (CPU intensive). The system achieves the highest throughput at WL 14,000 as shown in Figure 1(a) while no hardware resources are fully saturation as shown in Figure 1(b) and Figure 1(c).

method works at an unprecedented fine time granularity (milliseconds), which is more precise than normal sampling tools (e.g., dstat consumes 12% of CPU at 20ms intervals). Our method uses passive network packet tracing, which captures the arrival and departure time of each request of each server at microsecond granularity with negligible impact on the servers. By correlating the queue length and throughput of each server at millisecond granularity, our method is able to find short-lived alternating bottlenecks (lifetime of tens of milliseconds) that have been invisible to state-of-the-art sampling tools.

The rest of the paper is organized as follows. Section 2 introduces various kinds of bottlenecks. Section 3 shows our fine-grained performance analysis method. Section 4 and 5 show the two case studies of our experimental observations of rapidly alternating bottlenecks. Section 6 shows our concrete solutions to resolve the observed rapidly alternating bottlenecks. Section 7 summarizes the related work and Section 8 concludes the paper.

## 2. VARIOUS KINDS OF BOTTLENECKS

### 2.1 SINGLE BOTTLENECKS

A system bottleneck in an n-tier system is the place where requests start to be queued (or congested) and throughput is limited in the system. Classic queuing models assume independent jobs and predict single resource bottleneck in an n-tier system, in which the system achieves the maximum throughput when the single bottleneck resource is 100% utilized. Due to its simplicity and intuitiveness, classic queuing models have provided the foundation for system administrators to manage and predict system performance (Jung, Joshi, Hiltunen, Schlichting, & Pu, 2009; Uргаonkar, Shenoy, Chandra, & Goyal, 2005; Xiong et al., 2011). Despite their popularity, classic queuing models are based on assumptions (e.g., independent jobs among component servers in a system) that do not necessarily hold in an n-tier system in practice.

Server/Resource	CPU util (%)	Disk I/O (%)	Network receive/send (MB/s)
Apache	45.9	0.5	23.8/39.9
Tomcat	<b>86.9</b>	0.3	7.6/13.1
CJDBC	36.2	0.2	11.2/14.3
MySQL	<b>84.3</b>	0.4	0.8/4.6

Table 1: Average resource utilization in each tier at WL 14,000. Except Tomcat and MySQL CPU, the other system resources are far from saturation.

### 2.2 MULTI-BOTTLENECKS

Multi-bottlenecks describe a phenomenon where an n-tier system is saturated (i.e., achieves the maximum throughput) while no single hardware resource is fully utilized (Malkowski et al., 2009). We use an example to illustrate this phenomenon. The example was derived from a three-minute experiment of RUBBoS running on a four-tier configuration (1L/2S/1L/2S, see Figure 14(c)). The details of the experimental setup are in Appendix A.

Figure 1(a) shows the system works well from a workload of 1,000 concurrent users to 13,000. At 14,000, the average response time increases significantly and the throughput reaches a maximum. The interesting observation is that the saturated system does not have any single resource bottleneck. Since we use the CPU intensive Browse-only workload of this benchmark, we show the timeline graphs (with one second granularity) of CPU utilization. During the execution of the WL 14,000, both Tomcat (Figure 1(b) and MySQL (Figure 1(c)) show less than full CPU utilization, with an average of 86.9% (Tomcat) and 84.3% (MySQL). We also summarize the average usage of other main hardware resources of each server in Table 1. This table shows that except for Tomcat and MySQL CPU, the other system resources are far from saturation.

This example shows that simply monitoring hardware resource utilization may be unable to identify the system bottleneck, since there is no single saturated resource. Later in Section 4 we explain that this is a rapidly alternating

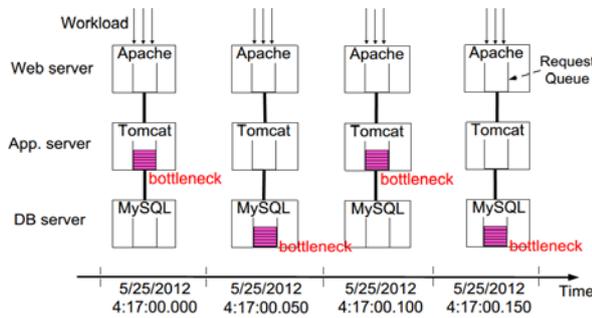


Figure 2: Illustration of a rapidly alternating bottleneck case in a 3-tier web application.

bottleneck case (a special case of multi-bottleneck) where the bottleneck alternates rapidly between MySQL and Tomcat. During normal processing, MySQL CPU is the primary system bottleneck, being fully utilized for processing requests sent from Tomcat. However, the Tomcat JVM garbage collection process freezes request processing and consumes the server CPU (at the granularity of milliseconds). Thus the Tomcat becomes the bottleneck during garbage collection. In either case, the system throughput is limited.

### 2.3 RAPIDLY ALTERNATING BOTTLENECKS

Rapidly alternating bottlenecks are a special case of multi-bottlenecks that the bottleneck in an n-tier system alternates rapidly (on the order of tens of milliseconds) among multiple system resources while at any moment one system resource becomes the main bottleneck. Rapidly alternating bottlenecks arise due to the implicit dependencies among servers in an n-tier system. For example, requests that originate from a client arrive at the web server, which distributes them among the application servers, which in turn ask the database servers to carry out the query. The dependencies among the servers are in the long invocation chain (through RPC calls) of transaction processing in the system and maintained by soft resources (e.g., threads and database connections (Wang et al., 2011)). Such dependencies may cause requests to congest in different servers at different time period. For example, Figure 2 demonstrates a rapidly alternating bottleneck case in a 3-tier system. This figure shows that the bottleneck alternates between Tomcat and MySQL at tens of milliseconds level, thus monitoring resource utilization at every second can rarely detect any resource saturation (similar to what Figure 1(b) and Figure 1(c) show).

The identification of rapidly alternating bottlenecks as an important phenomenon is due to its significant impact in cloud computing environments, where hardware resources are supposed to be “infinite” for applications to scale. Detection of rapidly alternating bottlenecks poses significant challenges to current state of the art monitoring tools, which leads to inefficient performance management of applications deployed in cloud.

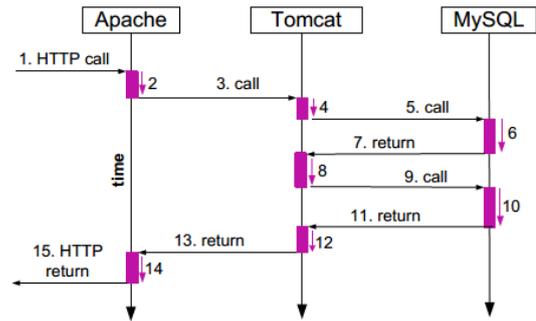


Figure 3: An example of a transaction execution trace captured by SysViz.

## 3. DETECTION OF RAPIDLY ALTERNATING BOTTLENECK

In this section, we briefly explain our fine-grained analysis to detect rapidly alternating bottlenecks. This kind of analysis is necessary to detect a bottleneck alternating on the order of tens of milliseconds among servers. Later we will show two case studies of applying our method to detect rapidly alternating bottlenecks caused by JVM garbage collection (Section 4) and VM collocation (Section 5).

### 3.1 TRACE MONITORING TOOL

Our fine-grained analysis is based on the tracing of client transaction executions of an n-tier system. Before we start the details of the fine-grained analysis method, we first briefly explain our tool (Fujitsu SysViz (“Fujitsu SysViz: System Visualization,” 2010)) for the tracing of transaction executions in an n-tier system. A client transaction services an entire web page requested by a client and may consist of multiple interactions between different tiers. Figure 3 shows an example of such a trace (numbered arrows) of a client transaction execution in a three-tier system. SysViz is able to reconstruct the entire trace of each transaction executed in the system based on the traffic messages (odd-numbered arrows) collected through a network switch which supports passive network tracing. In our experimental environment, all the servers are connected to the network switch, which forwards all the traffic messages to a dedicated SysViz server. Thus, the arrival/departure timestamps of each request (small boxes with even-numbered arrows) for any server can be recorded by the SysViz server.

SysViz requires no modification on application source code and has a negligible performance impact on the target n-tier application. We note that since the timestamps of all messages are assigned by one dedicated SysViz server, the precision of the derived processing time of each request in any tier in the system is close to microsecond level. Thus, the influence of clock errors between machines caused by limited accuracy of NTP can be avoided.

In fact the transaction tracing technology has been

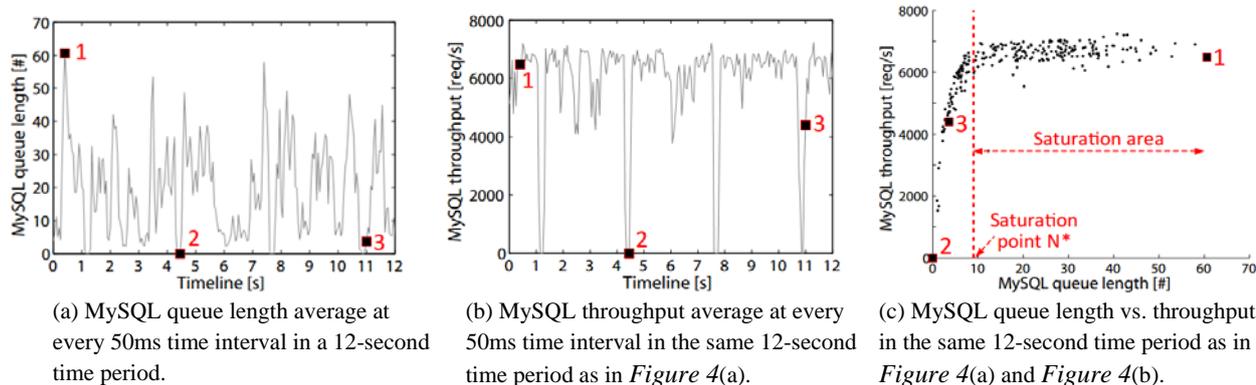


Figure 4: Fine-grained analysis of MySQL when the system is at WL 14,000. Figure 4(a) and Figure 4(b) show the MySQL queue length and throughput measured at the every 50ms time interval. Figure 4(c) is derived from Figure 4(a) and Figure 4(b); each point in Figure 4(c) represents the MySQL queue length and throughput measured at the same 50ms time interval during the 12-second experimental time period.

studied intensively in previous research (Barham, Donnelly, Isaacs, & Mortier, 2004; Fonseca, Porter, Katz, Shenker, & Stoica, 2007; Sambasivan et al., 2011; Sigelman et al., 2010); the ongoing research trend is how to use the captured tracing information to diagnose system performance problem.

### 3.2 FINE-GRAINED QUEUE-LENGTH/THROUGHPUT ANALYSIS

Since each participating server in a rapidly alternating bottleneck case only presents short-term saturations, a key point of detecting the rapidly alternating bottleneck is to find the participating short-term saturated servers. Instead of monitoring hardware resource utilizations, our approach measures a server's queue length and throughput in continuous fine-grained time intervals. The throughput of a server can be calculated by counting the number of completed requests in the server in a fixed time interval, which can be 50ms, 100ms, or 1s. Queue length is the average number of queued requests over the same time interval<sup>1</sup>. Both these two metrics for each server in the system can be easily derived from the trace captured by SysViz.

Figure 4(a) shows the MySQL queue length average at every 50ms time interval over a 12-second time period for the 1L/2S/1L/2S configuration case at WL 14,000 (See the case in Figure 1). This figure shows that a large number of requests are queued in MySQL from time to time, which suggests MySQL frequently presents short-term saturation. Figure 4(b) shows the MySQL throughput average at every 50ms time interval over the same 12-second time period as

in Figure 4(a). This figure shows that in some time intervals MySQL even produces zero throughput, which suggests MySQL is frequently under-utilized.

To precisely diagnose in which time intervals a server presents short-term saturation, we need to correlate the server's queue length and throughput as shown in Figure 4(c). This figure is derived from Figure 4(a) and Figure 4(b); each point in Figure 4(c) represents the MySQL queue-length/throughput measured at the same 50ms time interval during the 12-second experimental time period. This figure shows the clear trend of queue-length/throughput correlation (we call the trend as main sequence curve), which is consistent with Denning et al.'s (Denning & Buzen, 1978) operational analysis result for the relationship between a server's queue length<sup>2</sup> and throughput. Specifically, a server's throughput increases as the queue length on the server increases until it reaches the maximum throughput  $TP_{max}$ .<sup>3</sup> The saturation point  $N^*$  is the minimum queue length beyond which the server starts to saturate.

Once  $N^*$  is determined, we can judge in which time intervals a server is saturated based on the measured queue length. For example, Figure 4(c) highlights three points labeled 1, 2, and 3, each of which represents the queue-length/throughput in a time interval that can match back to Figure 4(a) and Figure 4(b). Point 1 shows that MySQL is saturated in the corresponding time interval because the long queue length far exceeds  $N^*$ . Point 2 shows that MySQL is not saturated due to the zero queue length and throughput. Point 3 also shows that MySQL is not saturated

<sup>1</sup> At each time tick, we know how many requests for a server have arrived, but not yet departed. This is the number of concurrent requests being processed by the server. Concurrent requests can also be thought as "queued" requests. More detailed fine-grained queue-length/throughput calculation can be found in (Wang, Kanemasa, Li, Jayasinghe, et al., 2013)

<sup>2</sup> The queue length in their definition is the *load* in a system, which means the number of jobs being processed concurrently.

<sup>3</sup> Due to the Utilization Law, the maximum throughput  $TP_{max}$  of a server is fixed by the bottleneck resource in terms of  $1/d$ , where  $d$  is the service demand for the bottleneck resource per job (Denning & Buzen, 1978).

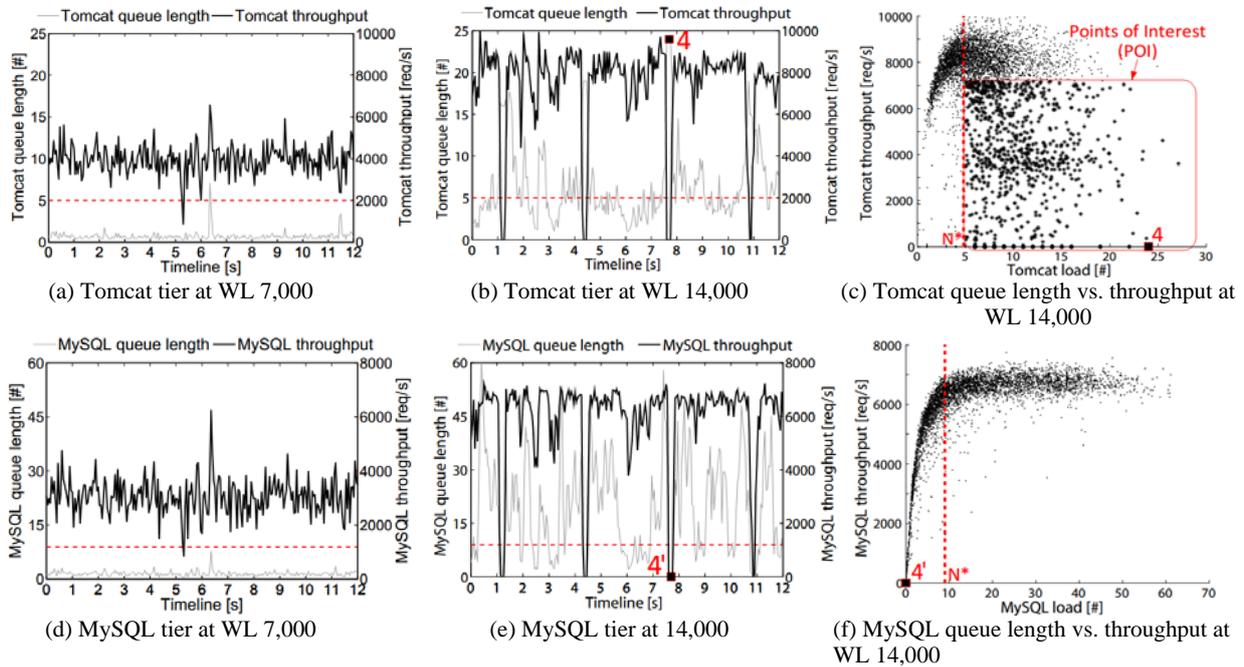


Figure 5: Fine-grained queue-length/throughput analysis for Tomcat and MySQL. Figure 5(c) and Figure 5(f) are derived from Figure 5(b) and Figure 5(e) respectively, with 3-minute experimental data. Figure 5(b) shows that there are some time intervals that Tomcat has high queue length but low throughput (see the point labeled 4), which correspond to the low queue length and throughput of MySQL as shown in Figure 5(e)(see the point labeled 4).

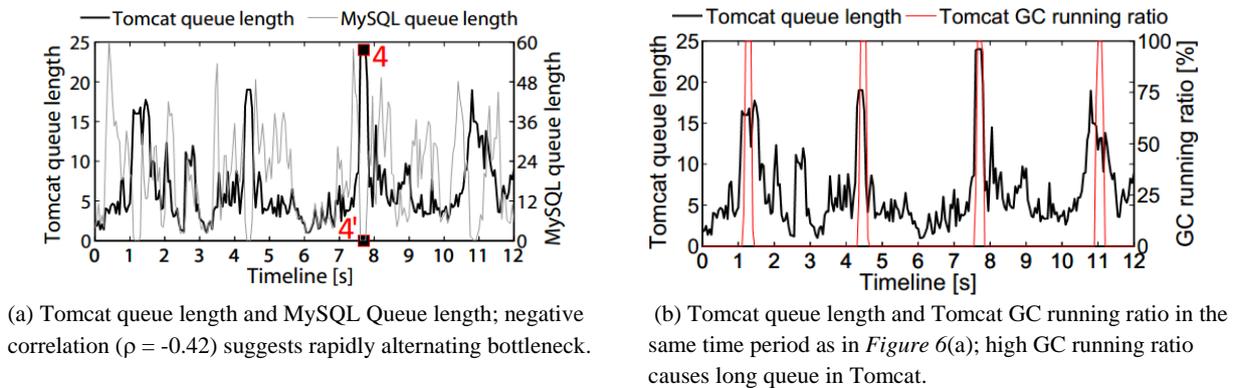


Figure 6: Correlation analysis of the rapidly alternating bottleneck between Tomcat and MySQL at WL 14,000. Figure 6(a) shows that Tomcat and MySQL queue length have strong negative correlation. Figure 6(b) shows that the peaks of Tomcat queue length are due to frequent JVM GCs.

because the corresponding queue length is less than  $N^*$  though it generates relatively high throughput.

After we detect all the short-term saturated servers, the next step is to analyze whether the short-term saturation of each participating server occurs in an alternating pattern. We will illustrate this point in the following two case studies.

#### 4. RAPIDLY ALTERNATING BOTTLENECK CAUSED BY JVM GC

In this section we explain the rapidly alternating bottleneck mentioned in Section 2.2. In that example, the poor system performance is caused by the frequent short-term saturations of both Tomcat and MySQL. Our further correlation analysis shows that the frequent JVM GCs in

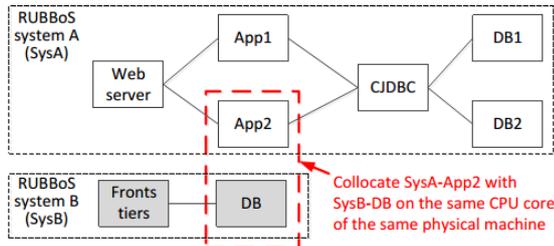


Figure 7: Collocation strategy between SysA and SysB; SysA-App2 is collocated with SysB-DB.

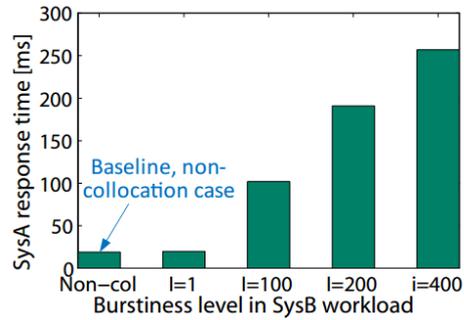


Figure 8: SysA response time (at WL 14,000) when collocated with SysB (at WL 400 but with increased burstiness level).

#	SysA-App2			SysB-DB		
	WL (users)	Burstiness level	CPU (%)	WL (users)	Burstiness level	CPU (%)
1	14,000	I = 1	74.1	0	Non-col	0
2	14,000	I = 1	74.9	400	I = 1	10.2
3	14,000	I = 1	74.7	400	I = 100	10.6
4	14,000	I = 1	75.5	400	I = 200	10.5
5	14,000	I = 1	75.2	400	I = 400	10.8

Table 2: Workload of SysA and SysB in each of the five collocation experiments. SysA is at constant stable WL 14,000 and SysB is at constant workload but with different burstiness levels (from I = 1 to 400).

Tomcat cause the bottleneck to alternate between Tomcat and MySQL.

Figure 5 shows the fine-grained load/throughput analysis for Tomcat and MySQL at WL 7,000 and 14,000 with the same system configuration as in Section 2.2. Figure 5(a) and Figure 5(d) show that both Tomcat and MySQL are not saturated at WL 7,000 since the load of each tier is below the corresponding saturation point N\*, which is derived from Figure 5(c) and Figure 5(f) respectively.

The interesting figures are Figure 5(b) and Figure 5(e), which show that at WL 14,000 both the Tomcat tier and the MySQL tier frequently present short-term saturations. Specially, Figure 5(b) shows that in some time intervals the Tomcat load is high (e.g., the point labeled 4) but the corresponding throughput is zero, which means that many requests are queued in Tomcat but no output responses (throughput). Figure 5(c), which is derived from Figure 5(b) but based on the 3-minute runtime experiments, shows that there are many time intervals when Tomcat has a high load but low or even zero throughput (POI inside the rectangular area). Since Tomcat is the upstream tier of MySQL, the output responses of Tomcat feeds the input requests of MySQL; thus having fewer output responses from Tomcat means there will be fewer input requests sent to MySQL, which leads to the under-utilization of MySQL. For example, the point labeled 4 in Figure 5(b) illustrates zero throughput in Tomcat, which leads to the zero throughput and load of MySQL (see the point labeled 4' in Figure 5(e)).

To illustrate the rapidly alternating bottleneck between Tomcat and MySQL, Figure 6(a) shows the correlation

between the Tomcat load and the MySQL load over the same 12-second time period. This figure shows that these two metrics have a negative correlation (the Pearson correlation is -0.42), which suggests that the short-term saturation alternates between Tomcat and MySQL. Thus, the reason for the limited system throughput is clear: at any moment either Tomcat or MySQL becomes the bottleneck in the system.

Our further analysis shows that the short-term saturations of Tomcat are caused by frequent JVM GC. In this set of experiments, the JVM in Tomcat (with JDK 1.5) uses a synchronous garbage collector; it waits during the garbage collection period and only starts processing requests after the garbage collection is finished. To confirm that JVM GC causes the bottleneck in Tomcat, Figure 6(b) shows the timeline graph which correlates the Java GC running ratio<sup>4</sup> with the Tomcat load (50ms). This figure shows the occurrence of Tomcat JVM GC has a strong positive-correlation with the high load in Tomcat. The high peaks of JVM GC in Figure 6(b) stop Tomcat and make requests queued in Tomcat dramatically. We note that such long freeze times in Tomcat do not happen frequently when the system is under low workload as shown in Figure 5(a). This is because JVM GC has a non-linear relationship with the amount of workload (Wang, Kanemasa, Kawaba, & Pu, 2012)

## 5. RAPIDLY ALTERNATING BOTTLENECK CAUSED BY VM COLLOCATION

In this section we show another rapidly alternating bottleneck case due to VM collocation, i.e., collocating multiple under-utilized VMs into the same physical host so that VMs can share hardware resources. Although VM collocation can reduce infrastructure/maintenance

<sup>4</sup> Java GC running ratio means the total time spent on Java GC during each monitoring time interval to the total monitoring time interval length. JVM provides a tool recording the starting/ending timestamp of every GC activity.

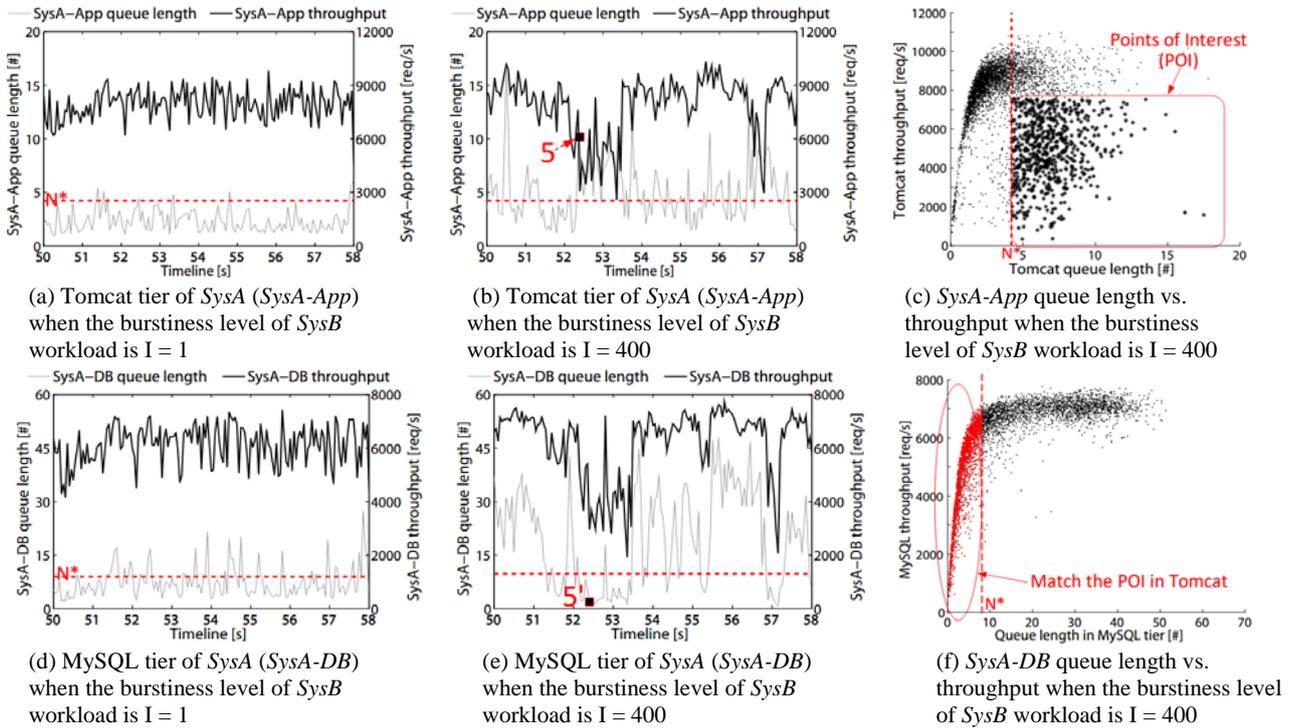
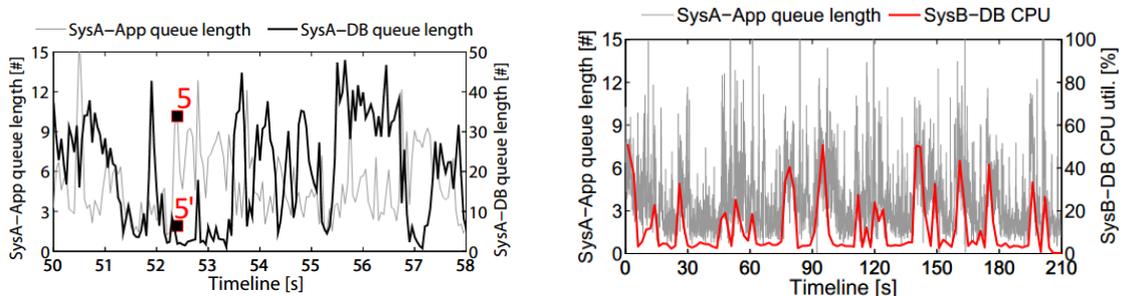


Figure 9: Fine-grained queue-length/throughput analysis for the Tomcat and MySQL of SysA in the collocation experiments. Figure 9(c) and Figure 9(f) are derived from Figure 9(b) and Figure 9(e) respectively, with 3-minute experimental data. Figure 9(a) and Figure 9(d) show that the queue length of the SysA Tomcat/MySQL tier is low when SysB workload is stable ( $I = 1$ ). As the burstiness of SysB workload increases ( $I = 400$ ), there are some time intervals that SysA-App has high queue length but low throughput (see the point labeled 5 in Figure 9(b)), which correspond to the low queue-length/throughput of SysA-DB (see the point labeled 5' in Figure 9(e)).



(a) The queue length of SysA-App and SysA-DB when SysB is at bursty workload ( $I = 400$ ); negative correlation suggests rapidly alternating bottleneck in SysA. (b) SysA-App queue length vs. SysB-DB CPU; burst of SysB-DB CPU utilization causes long queue in the SysA-App tier.

Figure 10: Analysis of rapidly alternating bottleneck between SysA-App and SysA-DB. Figure 10(b) shows that the peaks of SysA-App queue length are caused by the burst of SysB-DB CPU utilization because SysA-App and SysB-DB are competing for CPU resources.

costs(Barham et al., 2003; Govindan, Liu, Kansal, & Sivasubramaniam, 2011), it may significantly hamper the performance of the collocated applications in a non-trivial way, especially when the workload for the collocated applications becomes bursty (Kanemasa, Wang, Li, Matsubara, & Pu, 2013; Malkowski et al., 2012). For example, rapidly alternating bottlenecks may occur in a

foreground application if it is collocated with another application facing a bursty workload.

We illustrate this problem by collocating two VMs, each of which is from a separate n-tier application, into the same host and with each VM sharing the same CPU core. Figure 7 shows our collocation strategy of the two applications; SysA with 1L/2S/1L/2S configuration (4-tier) and SysB

with 1S/1S/1S configuration (3-tier). SysA keeps the same hardware configuration as in the previous section but with JDK1.6 in Tomcat.<sup>5</sup> The VM of SysA-App2 is collocated with the VM of SysB-DB on the same ESXi host and they share the same CPU core; the VMs of the front tiers (one Apache and one Tomcat) of SysB are deployed in separate ESXi hosts from SysA in order to simplify the analysis. Table 2 shows the workload conditions and average CPU utilization of both SysA and SysB in the collocation experiments. SysA is at a constant stable workload of 14,000 in all five experiments. Except for the first experiment (the non-collocation case), SysB is under constant WL 400 but with varying burstiness levels, which is represented by  $I$ .<sup>6</sup> This table shows that the average CPU utilization of both the collocated VMs SysA-App2 and SysB-DB are almost constant and the total of the CPU utilization is less than 90%, which justifies the collocation strategy based on traditional bin packing practices.

Figure 8 shows the average response time of SysA in all the five cases. This figure shows that the SysA response time is almost the same when the collocated SysB has the relatively stable workload ( $I = 1$ ), and increases significantly when the burstiness level of the workload for SysB becomes high (e.g.,  $I = 400$ ). The significant increase in SysA response time may seem strange since the average CPU utilization remains constant as seen in Table 2.

Figure 9(b) shows a similar interesting phenomenon as in the previous rapidly alternating bottleneck case that in some time intervals (e.g., between 52s and 53s) the SysA-App has a long queue length but low throughput; the low throughput of SysA-App leads to the low queue length and throughput in SysA-DB during the same time period as shown in Figure 9(e). Figure 9(c) (derived from Figure 9(b) but based on 3-minute experimental data) suggests that there are many time intervals when SysA-App has a long queue length but low throughput (points in POI). During these time intervals, SysA-App presents short-term saturations and SysA-DB is under-utilized due to the low number of input requests sent from SysA-App (see Figure 9(f)).

Figure 10(a) shows the correlation of the queue length between SysA-App and SysA-DB over the same 8-second time period. This figure shows that SysA-App queue length has a negative correlation ( $\rho = -0.46$ ) with SysA-DB queue length, which suggests the bottleneck alternates rapidly between SysA-App and SysA-DB.

Our further analysis shows that the short-term saturation of SysA-App is caused by the burst of SysB-DB CPU utilization. Figure 10(b) shows the timeline graph of the CPU utilization of SysB-DB (measured using VMware

esxtop with 2s granularity) and the SysA-App queue length (measured at every 50ms time interval). This figure shows that the SysA-App queue length increases significantly when there is a spike in the SysB-DB CPU utilization<sup>7</sup>, which indicates that the Tomcat tier of SysA temporarily becomes the bottleneck due to the interference of SysB-DB. The detailed research about the race of CPU scheduling between collocated VMs has been provided in (Kanemasa et al., 2013).

## 6. RESOLVING RAPIDLY ALTERNATING BOTTLENECKS

Once we detect a rapidly alternating bottleneck, we can resolve the bottleneck through various ways, depending on whether we can find the exact cause for the rapidly alternating bottleneck. Specifically, we can simply scale-out/up the participating servers if we cannot find the exact cause, or we can resolve the bottleneck by addressing the exact cause. For example, we can resolve the rapidly alternating bottleneck caused by frequent JVM GCs in Tomcat through upgrading the JDK version from 1.5 to 1.6, which has more efficient garbage collectors.<sup>8</sup> We can also resolve the rapidly alternating bottleneck caused by VM collocation through VM migration.

In this section we show the performance gain after we resolve the rapidly alternating bottleneck caused by frequent JVM GCs in Tomcat (Section 4). We choose to resolve the bottleneck by upgrading the JDK version from 1.5 to 1.6 in Tomcat since we know the exact cause. The experiments have the same hardware/software configuration as in Section 3.2 except for the Tomcat JDK version.

Figure 11 shows the fine-grained queue-length/throughput analysis for Tomcat and MySQL at WL 14,000 and 16,000. Recall from Section 4 the system throughput reaches the maximum at WL 14,000 due to the rapidly alternating bottleneck between Tomcat and MySQL before the JDK version upgrade. After the JDK version upgrade, Figure 11(a) and Figure 11(d) show that only MySQL presents frequent short-term saturations at WL 14,000; further workload increase to 16,000 leads to the full saturation of MySQL as shown in Figure 11(e) (the queue length is above the  $N^*$  most of the time). Thus, the rapidly alternating bottleneck is resolved after the JDK upgrade in Tomcat. Specifically, the POIs in Figure 5(c) do not appear in Figure 11(c), which means Tomcat does not have long “freezing” periods. Accordingly, there are only a few time intervals with low queue length and low throughput in MySQL (see Figure 11(e)), which means MySQL is fully utilized.

<sup>5</sup> The upgrade of JDK version in Tomcat solves the rapidly alternating bottleneck caused by frequent JVM GCs; see Section 6 for more details.

<sup>6</sup> Mi et al. (Mi, Casale, Cherkasova, & Smirni, 2009) introduced index of dispersion (abbreviated as  $I$ ) to characterize the intensity of the traffic surges. The larger the  $I$  is, the longer the duration of the traffic surge. The burstiness level of the by default RUBBoS workload is  $I = 1$ .

<sup>7</sup> We couldn't measure the fine-grained CPU usage of SysB-DB because 2s is the finest granularity that the latest esxtop supports.

<sup>8</sup> JDK 1.6 uses garbage collection algorithms which support both parallel and concurrent garbage collection.

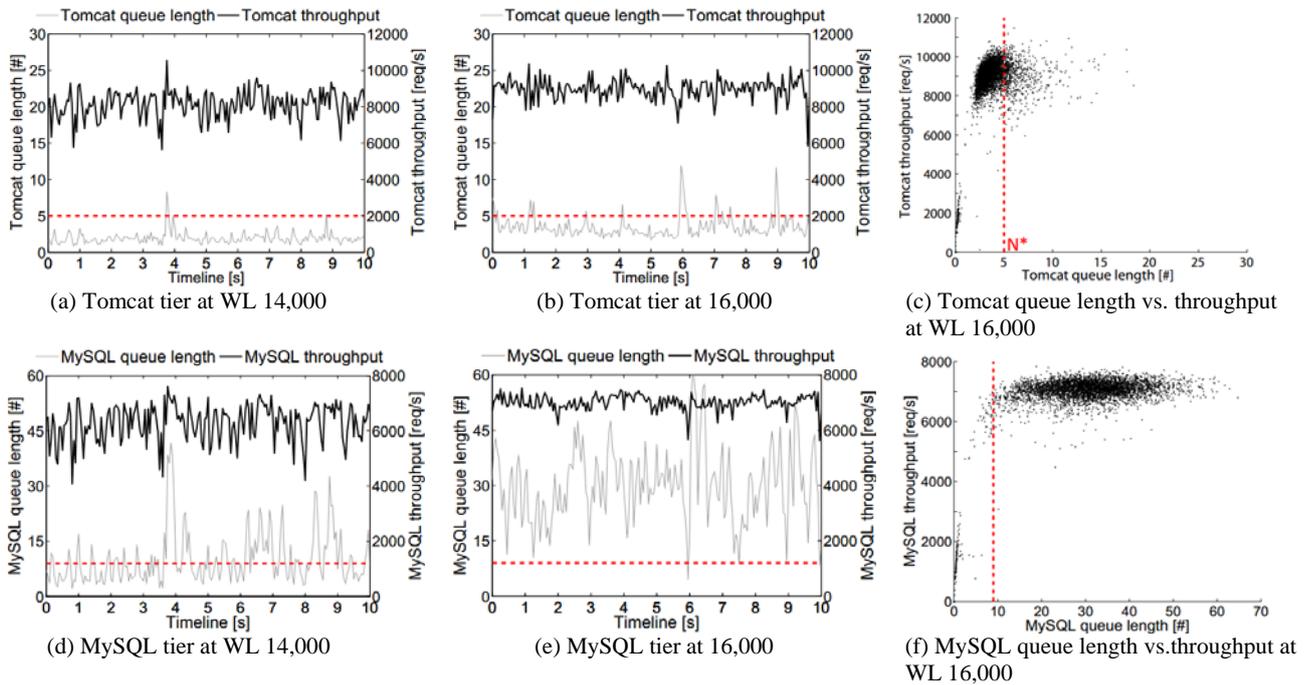


Figure 11: Fine-grained queue-length/throughput analysis for Tomcat (with JDK 1.6) and MySQL. Figure 11(c) and Figure 11(f) are derived from Figure 11(b) and Figure 11(e) respectively, with 3-minute experimental data. Figure 11(b) and Figure 11(e) show that the rapidly alternating bottleneck is resolved and the MySQL tier becomes the single bottleneck.

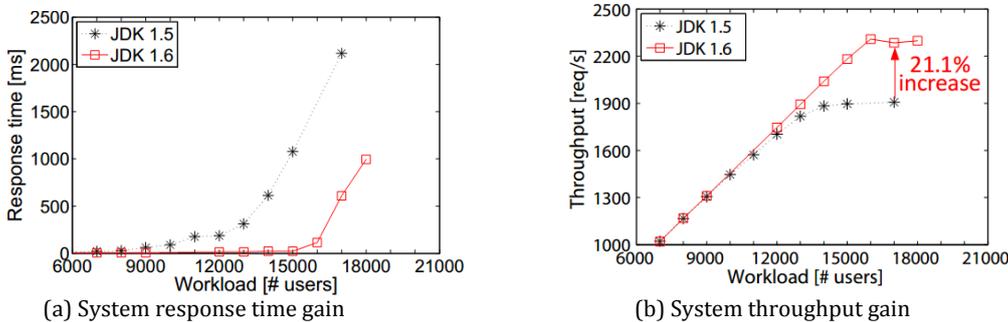


Figure 12: Performance improvement after resolving rapidly alternating bottleneck.

Figure 12(a) and Figure 12(b) show the system response time and throughput gain after we resolve the rapidly alternating bottleneck. At WL 17,000, the system with JDK 1.6 outperforms the system with JDK 1.5 by a 21.1% higher throughput while achieving an average response time that is about 71% lower (1500ms faster).

We note we can resolve the rapidly alternating bottleneck caused by VM collocation described in Section 5 through migrating the collocated VM to a different ESXi host. We should use this solution only when we detect the performance interference caused by collocated VMs.

## 7. DISCUSSION

So far we illustrated that rapidly alternating bottlenecks can be caused by system software level factors (e.g., JVM

GC) and middleware level factors (VM collocation). In fact there are many other factors that can cause rapidly alternating bottlenecks such as Dynamic Voltage and Frequency Scaling (DVFS) technology at the architecture level (Wang, Kanemasa, Li, Lai, et al., 2013). The reason why rapidly alternating bottlenecks can happen frequently is because there are many factors that can cause short-term saturations of component servers in real systems, especially when the system is at high utilization. The short-term saturations of component servers, compounded with dependencies among tiers in an n-tier system, create rapidly alternating bottlenecks in the system.

The short-term saturation of a component server in an n-tier system has a significant impact on the servers in other tiers of the system due to resource dependencies among tiers

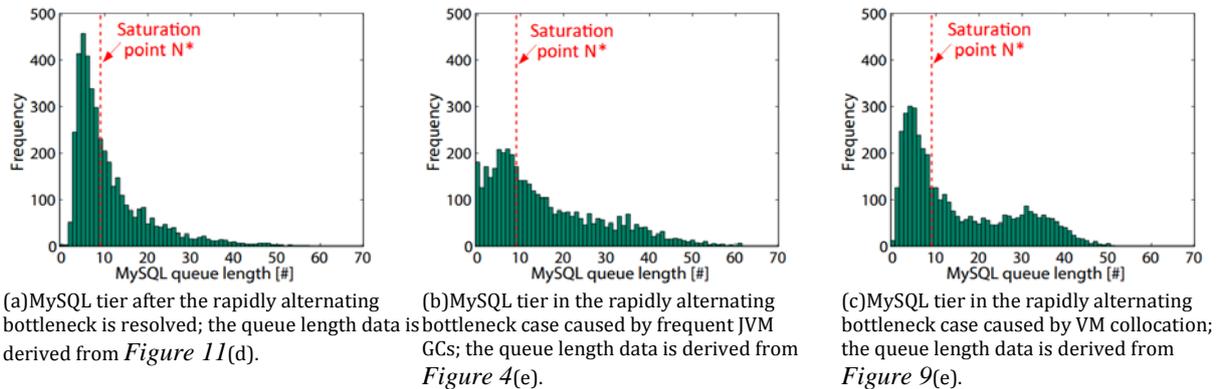


Figure 13: The comparison of MySQL queue length distribution at WL 14,000 before and after resolving the rapidly alternating bottlenecks. Figure 13(b) and Figure 13(c) show that the MySQL tier encounters more frequent long queue length compared to the case in Figure 13(a), which suggests that the Tomcat tier sends more bursty workload into the MySQL tier and causes more frequent congestion in MySQL in a rapidly alternating bottleneck case.

in the system. For example, the short-term saturation (either caused by JVM GC or VM collocation) of a server causes a bursty workload to the downstream tiers, causing downstream tiers to switch between a saturation and non-saturation state. Specifically, during the short-term saturation of a server, many requests start to queue in the server, which causes the downstream tiers to be under-utilized (non-saturation state); after the short-term saturation period, the queued requests are pushed to the downstream tiers in a batch mode, which cause the short-term saturation of the downstream tiers; the short-term saturation may present an alternating pattern between the server and the downstream tiers once the frequency of the short-term saturations of the server becomes high.

Figure 13 shows the comparison of MySQL queue length distribution before and after resolving the rapidly alternating bottlenecks caused by the short-term saturations of Tomcat (see Section 4 and Section 5). The system in all these three cases is under the same hardware configuration (1L/2S/1L/2S), at the same WL 14,000, and has the same amount of experimental time (3-minute). However, Figure 13(b) and Figure 13(c) show that MySQL has more frequent long queue length compared to the case after we resolve the alternating bottleneck as shown in Figure 13(a). Specially, Figure 13(c) even shows a clear bi-modal distribution. This is because in a rapidly alternating bottleneck case the frequent short-term saturations of Tomcat augment the burstiness level of the workload to MySQL, which causes more frequent short-term saturations in MySQL.

## 8. RELATED WORK

Performance debugging and scalability analysis for distributed systems such as n-tier systems has been widely studied in previous research efforts.

Shifting/Alternating bottlenecks have been studied before in either multiclass queueing networks or n-tier enterprise systems. Balbo et al. (Balbo & Serazzi, 1997) and

Casale et al. (Casale & Serazzi, 2004) use analytical approaches to illustrate that bottlenecks in a multiclass queueing network with load independent servers can switch to different servers, depending on the current workload mix. Malkowski et al. (Malkowski et al., 2009) showed an alternating bottleneck case where the bottleneck alternates among eight MySQL databases due to the unbalanced query dispatching from the upper tiers. As shown in this paper, alternating bottlenecks can be far more common than previously believed. The reason behind is that the detection of an alternating bottleneck becomes extremely difficult as the frequency of alternating pattern increases; such interesting phenomena are completely invisible to normal monitoring tools that sample at time intervals measured in seconds or minutes.

Analytical models have been proposed for bottleneck detection and performance prediction of n-tier systems. Urgaonkar et al. (Urgaonkar, Pacifici, Shenoy, Spreitzer, & Tantawi, 2005; Urgaonkar, Shenoy, et al., 2005) present a novel dynamic provisioning technique for n-tier systems that employs a flexible queuing model to determine how much resources to allocate to each tier of the application. However, such model is based on Mean Value Analysis (MVA), which has difficulties dealing with alternating bottleneck cases in the system. Mi et al. (Mi et al., 2008) propose a more sophisticated analytical model that considers bottleneck shifting in an n-tier system due to bursty workloads. One challenge of this work is to precisely map the bursty characteristics of a workload to the queuing model with multiple service rates for each queue in the system. As shown in this paper, without fine-grained monitoring (sub-second level) granularity, the bursty characteristics of a workload can be largely masked.

Software mis-configuration and failure detection of distributed system have been studied in (Attariyan & Flinn, 2010; Oliveira, Tjang, Bianchini, Martin, & Nguyen, 2010). Attariyan et al. (Attariyan & Flinn, 2010) present a tool that

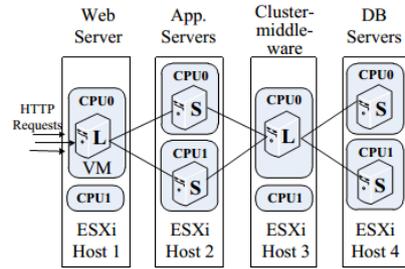
ESXi Host Configuration	
Model	Dell Power Edge T410
CPU	2* Intel Xeon E5607, 2.26GHz Quad-Core
Memory	16GB
Storage	7200rpm SATA local disk

VM Configuration					
Type	# vCPU	CPU limit	CPU shares	vRAM	vDisk
Large (L)	2	4.52GHz	Normal	2GB	20GB
Small (S)	1	2.26GHz	Normal	2GB	20GB

(a) Software setup

Software Stack	
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	jdk1.5.0_07, jdk1.6.0_14
Operating system	RHEL 6.2 (kernel 2.6.32)
Hypervisor	VMware ESXi v5.0
System monitor	esxtop 5.0, Sysstat 10.0.0

(b) ESXi host and VM setup;



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

Figure 14: Details of the experimental setup

locates the root cause of configuration errors by applying dynamic information flow analysis within process (mainly) in the runtime. Oliveira et al. (Oliveira et al., 2010) propose a mistake-aware management framework for protecting n-tier systems against operator mistakes by facilitating the previous correct operations. All these works differ from our work in that they focus more on faults or anomalous behavior of system components rather than the system performance problem.

Perhaps the work closest to ours is Aguilera et al.'s performance debugging based on Black boxes (Aguilera, Mogul, Wiener, Reynolds, & Muthitacharoen, 2003). The authors propose a statistical method that derives causal paths (the trace of a transaction) in a distributed system (not limited to n-tier system) from the communication messages between different nodes. By measuring the delay of request processing in each node, they detect the "bottlenecked" server as the node where a request has the longest delay in the system. Though this approach can be effective to detect the single bottleneck case, it may fail to detect rapidly alternating bottlenecks since requests can wait in multiple servers in an alternating pattern.

## 9. CONCLUSIONS

We observed that the performance of an n-tier system may degrade significantly due to rapidly alternating bottlenecks between multiple tiers. We found that rapidly alternating bottlenecks can be caused by various factors at different levels of an n-tier application; for instance, JVM GC at the software level (Section 3.2), VM collocation at the middleware level (Section 3.3). Solving those rapidly alternating bottlenecks leads to significant performance improvement (Section 4). We proposed a novel bottleneck detection method to detect these rapidly alternating bottlenecks (Section 5), where the effectiveness of our approach is validated through the two case studies in Section 3. More generally, our work is an important contribution towards scaling complex n-tier applications under elastic workloads in cloud environments.

## 10. REFERENCES

Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P., & Muthitacharoen, A. (2003). Performance debugging for distributed systems of black boxes. Paper presented at the *ACM SIGOPS Operating Systems Review*, 37(5), 74-89.

Attariyan, M., & Flinn, J. (2010). Automating Configuration Troubleshooting with Dynamic Information Flow Analysis. Paper presented at the *9th USENIX Symposium on Operating Systems Design and Implementation*. Vancouver, BC, Canada, Oct. 2010, 237-250

Balbo, G., & Serazzi, G. (1997). Asymptotic analysis of multiclass closed queueing networks: multiple bottlenecks. *Perform. Eval.*, 30(3), 115-152.

Barham, P., Donnelly, A., Isaacs, R., & Mortier, R. (2004). Using Magpie for Request Extraction and Workload Modelling. Paper presented at the *6th USENIX Symposium on Operating Systems Design and Implementation*. San Francisco, CA, Dec, 2004, 18-31

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., . . . Warfield, A. (2003). Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5), 164-177.

Casale, G., & Serazzi, G. (2004). Bottlenecks Identification in Multiclass Queueing Networks Using Convex Polytopes. Paper presented at the *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. Volendam, The Netherlands, Oct. 2004, 223-230

Denning, P. J., & Buzen, J. P. (1978). The operational analysis of queueing network models. *ACM Computing Surveys (CSUR)*, 10(3), 225-261.

Fonseca, R., Porter, G., Katz, R. H., Shenker, S., & Stoica, I. (2007). X-trace: A pervasive network tracing framework. Paper presented at the *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*. Cambridge, MA, April 2007, 20-30

Fujitsu SysViz: System Visualization. (2010). from "http://www.google.com/patents?id=0pGRAAAEBAJ&zoom=4&pg=PA1#v=onepage&q&f=false "

Govindan, S., Liu, J., Kansal, A., & Sivasubramaniam, A. (2011). Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines. Paper presented at the *Proceedings of the 2nd ACM Symposium on Cloud Computing, Cascais, Portugal, Oct. 2011*, 22-34

Jung, G., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., & Pu, C. (2009). A cost-sensitive adaptation engine for server consolidation of multitier applications *Middleware 2009* (pp. 163-183): Springer.

Kanemasa, Y., Wang, Q., Li, J., Matsubara, M., & Pu, C. (2013). Revisiting Performance Interference among Consolidated n-Tier Applications: Sharing is Better than Isolation. Paper presented at the *Service Computing (SCC), 2012 IEEE 10th International Conference on. Santa Clara, CA, USA, July 2013*.

Malkowski, S., Hedwig, M., & Pu, C. (2009). Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks. Paper presented at the *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*.

Malkowski, S., Kanemasa, Y., Chen, H., Yamamoto, M., Wang, Q., Jayasinghe, D., . . . Kawaba, M. (2012). Challenges and opportunities in consolidation at high resource utilization: non-monotonic response time variations in n-tier applications. Paper presented at the *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. Honolulu, Hawaii, USA, June 2012, 162-169

Mi, N., Casale, G., Cherkasova, L., & Smirni, E. (2008). Burstiness in multi-tier applications: symptoms, causes, and new models. Paper presented at the *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Leuven, Belgium. Dec. 2008, 265-286

Mi, N., Casale, G., Cherkasova, L., & Smirni, E. (2009). Injecting realistic burstiness to a traditional client-server benchmark. Paper presented at the *Proceedings of the 6th international conference on Autonomic computing*. Barcelona, Spain, June 2009 149-158

Oliveira, F., Tjang, A., Bianchini, R., Martin, R. P., & Nguyen, T. D. (2010). Barricade: Defending systems against operator mistakes. Paper presented at the *Proceedings of the 5th European conference on Computer systems*, Paris, France, April 2010, 83-96

RUBBoS: Bulletin board benchmark. (2004). from "http://jmob.ow2.org/rubbos.htm"

Sambasivan, R. R., Zheng, A. X., De Rosa, M., Krevat, E., Whitman, S., Stroucken, M., . . . Ganger, G. R. (2011). Diagnosing performance changes by comparing request flows. Paper presented at the *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, Boston, MA, March 2011, 43-56

Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., . . . Shanbhag, C. (2010). Dapper, a large-scale distributed systems tracing infrastructure. *Google research, 2010*.

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., & Tantawi, A. (2005). An analytical model for multi-tier internet services and its applications. Paper presented at the *ACM SIGMETRICS Performance Evaluation Review*.

Urgaonkar, B., Shenoy, P., Chandra, A., & Goyal, P. (2005). Dynamic provisioning of multi-tier internet applications. Paper presented at the *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second ACM International Conference on*, Seattle, WA, USA, June 2005.217-228

Wang, Q., Kanemasa, Y., Kawaba, M., & Pu, C. (2012). When average is not average: large response time fluctuations in n-tier systems. Paper presented at the *Proceedings of the 9th ACM international conference on Autonomic computing*, San Jose, CA, USA, September 2012. 33-42

Wang, Q., Kanemasa, Y., Li, J., Jayasinghe, D., Shimizu, T., Matsubara, M., . . . Pu, C. (2013). Detecting transient bottlenecks in n-tier applications through fine-grained analysis. Paper presented at the *Distributed Computing Systems (ICDCS), 2013 33rd IEEE International Conference on.. Philadelphia, USA, July 2013*,

Wang, Q., Kanemasa, Y., Li, J., Lai, C. A., Matsubara, M., & Pu, C. (2013). *Impact of DVFS on n-Tier Application Performance*. Paper presented at the *1st ACM international conference on Timely Results in Operating Systems*, Farmington, Pennsylvania, USA, November, 2013

Wang, Q., Malkowski, S., Kanemasa, Y., Jayasinghe, D., Xiong, P., Pu, C., Harada, L. (2011). *The impact of soft resource allocation on n-tier application scalability*. Paper presented at the *Parallel & Distributed Processing Symposium (IPDPS), 2011 25th IEEE International*. Anchorage, Alaska, 1034-1045

Xiong, P., Wang, Z., Malkowski, S., Wang, Q., Jayasinghe, D., & Pu, C. (2011). Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach. Paper presented at the *Distributed Computing*

*Systems (ICDCS), 2011 31st IEEE International Conference on*, Minneapolis, Minnesota, USA, June, 2011, 571-580

## APPENDIX A: EXPERIMENTAL SETUP

In our experiments we adopt the RUBBoS standard n-tier benchmark ("RUBBoS: Bulletin board benchmark," 2004), based on bulletin board applications such as Slashdot. RUBBoS has been widely used in numerous research projects due to its design, derived from a real web-facing application. RUBBoS can be configured as a three-tier (web server, application server, and database server) or four-tier (addition of clustering middleware such as C-JDBC(Cecchet, Marguerite, & Zwaenepole, 2004)) system. The benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes. We use browse-only workload in this paper. The closed-loop workload generator of this benchmark generates a request rate that follows a Poisson distribution parameterized by a number of emulated clients and a fixed user thinking time. Such workload generator has a similar design as other standard n-tier benchmarks such as RUBiS, TPC-W, Cloudstone etc.

We run the RUBBoS benchmark on our virtualized testbed. Figure 14 outlines the software components, ESXi host and virtual machine (VM) configuration, and a sample topology used in the experiments. 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. Each server runs on top of one VM. We have two types of VMs: "L" and "S", each of which represents a different size of processing power. Figure 14(c) shows a sample 1L/2S/1L/2S topology. Each ESXi host runs the VMs from the same tier of the application. The VMs from the same tier are pinned to separate CPU cores to minimize the interference between VMs. Hardware resource utilization measurements (e.g., CPU) are taken during the runtime period using Sysstat at one second granularity and VMware esxtop at two second granularity (the minimum granularity provided by the tool). The esxtop is used to monitor the CPU utilization of each VM in the VM collocation experiments (see Section 5).

## Authors



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 multitier enterprise systems



Yasuhiko Kanemasa received the B.Eng. degree in computer engineering from Tokyo Institute of Technology (Tokyo, Japan) in 1996, and the M.S. degree in computer science from Japan Advanced Institute of Science and Technology (Nomi, Japan) in 1998. He has been working for Fujitsu Laboratories Ltd. (Kawasaki, Japan) as a researcher since 1998. His research interests are in the areas of database systems, performance analysis of enterprise computer systems, and cloud computing. He is a member of IPSJ and DBSJ.

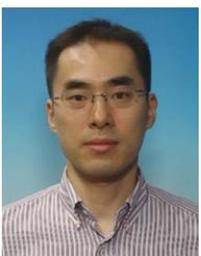


scalability.

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



Toshihiro Shimizu received his B.S. and M.S. in Computer Science from the University of Tokyo in 1994 and 1996, respectively. He has been working for Fujitsu Laboratories Ltd. since 1999. His current major research interests are in verification and testing of software systems.



is a member of IPSJ.

Masazumi Matsubara received the BE, ME and PhD in Engineering from University of Tsukuba in 1996, 1998 and 2001, respectively. He has been working for Fujitsu Laboratories Ltd. (Kawasaki, Japan) as a researcher since 2001. His research interests are in the areas of parallel computing, system management, and cloud computing. He



Motoyuki Kawaba received the B.E degree in Electronic Engineering from the University of Tokyo in 1989, and the M.E. degree in Information Engineering from the University of Tokyo in 1991. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1991 and has been engaged in research

and developments of computer systems and architectures. His research interests include storage systems, data management systems, and performance analysis of commercial workloads.



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.