Lightning in the Cloud: A Study of Very Short Bottlenecks on n-Tier Web Application Performance

Qingyang Wang*†, Yasuhiko Kanemasa‡, Jack Li*, Chien-An Lai*,
Chien-An Cho*,Yuji Nomura‡, Calton Pu*

*College of Computing, Georgia Institute of Technology
†Computer Science and Engineering, Louisiana State University
‡System Software Laboratories, FUJITSU LABORATORIES LTD.

Abstract

In this paper, we describe an experimental study of very long response time (VLRT) requests in the latency long tail problem. Applying micro-level event analysis on fine-grained measurement data from n-tier application benchmarks, we show that very short bottlenecks (from tens to hundreds of milliseconds) can cause queue overflows that propagate through an n-tier system, resulting in dropped messages and VLRT requests due to timeout and retransmissions. Our study shows that even at moderate CPU utilization levels, very short bottlenecks arise from several system layers, including Java garbage collection, anti-synchrony between workload bursts and DVFS clock rate adjustments, and statistical workload interferences among co-located VMs.

As a simple model for a variety of causes of VLRT requests, very short bottlenecks form the basis for a discussion of general remedies for VLRT requests, regardless of their origin. For example, methods that reduce or avoid queue amplification in an n-tier system result in non-trivial trade-offs among system components and their configurations. Our results show interesting challenges remain in both causes and effective remedies of very short bottlenecks.

1 Introduction

Wide response time fluctuations (latency long tail problem) of large scale distributed applications at moderate system utilization levels have been reported both in industry [18] and academia [24, 26, 27, 38, 43]. Occasionally and without warning, some requests that usually return within a few milliseconds would take several seconds. These very long response time (VLRT) requests are difficult to study for two major reasons. First, the VLRT requests only take milliseconds when running by themselves, so the problem is not with the VLRT requests, but emerges from the interactions among system components. Second, the statistical average behavior of system components (e.g., average CPU utilization over typical measurement intervals such as minutes) shows all system components to be far from saturation.

Although our understanding of the VLRT requests has been limited, practical solutions to bypass the VLRT request problem have been described [18]. For example, applications with read-only semantics (e.g., web search) can use duplicate requests sent to independent servers and reduce perceived response time by choosing the earliest answer. These bypass techniques are effective in specific domains, contributing to an increasingly acute need to improve our understanding of the general *causes* for the VLRT requests. On the practical side, our lack of a detailed understanding of VLRT requests is consistent with the low average overall data center utilization [37] at around 18%, which is a more general way to avoid VLRT requests (see Section 4). The current situation shows that VLRT requests certainly merit further investigation and better understanding, both as an intellectual challenge and their potential practical impact (e.g., to increase the overall utilization and return on investment in data centers).

Using fine-grained monitoring tools (a combination of microsecond resolution message timestamping and millisecond system resource sampling), we have collected detailed measurement data on an n-tier benchmark (RUBBoS [6]) running in several environments. Micro-level event analyses show that VLRT requests can have very different causes, including CPU dynamic voltage and frequency scaling (DVFS) control at the architecture layer, Java garbage collection (GC) at the system software layer, and virtual machine (VM) consolidation at the VM layer. In addition to the variety of causes, the non-deterministic nature of VLRT requests makes the events dissimilar at the micro level.

Despite the wide variety of causes for VLRT requests, we show that they can be understood through the concept

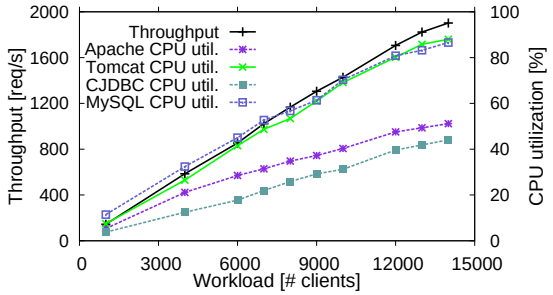


Figure 1: System throughput increases linearly with the CPU utilization of representative servers at increasing workload.

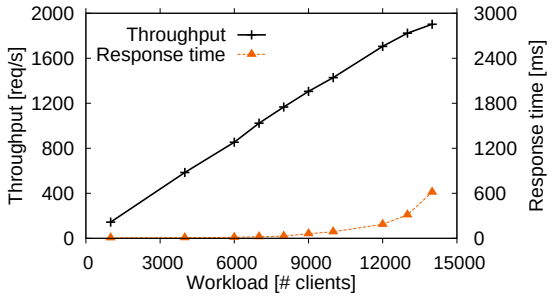


Figure 2: System throughput and average response time at increasing workload. The wide response time fluctuations are not apparent since the average response time is low (<200ms) before 12000 clients.

of *very short bottleneck*, defined as a very short period of time (on the order of tens of milliseconds), during which the CPU is saturated. When considered at the abstraction level of very short bottlenecks, the phenomenon of VLRT requests becomes reproducible: Even though the actual VLRT requests may not be literally the same ones, a similar number of VLRT requests arise reliably according to the timeline in experiments. Consequently, we are able to show concrete evidence that ties convincingly the various causes mentioned above to VLRT requests. We compare very short bottlenecks to lightning, since they are very short in duration (tens of milliseconds), but have a long impact on the VLRT requests (several seconds).

The main contribution of the paper is a set of micro-level event analyses of fine-grained experimental data of the RUBBoS benchmark in several environments. The initial steps of the micro-level event analyses are similar: (1) VLRT requests are detected of an n-tier system with moderate utilization; (2) at the same time, long request queues form in the Apache overflowing TCP buffer, that causing dropped packets and retransmission after 3 seconds; (3) the long queues in Apache formed because the downstream servers (Tomcat) have completely full queues during that time; (4) long queues in Tomcat servers are created by very short bottlenecks, in which

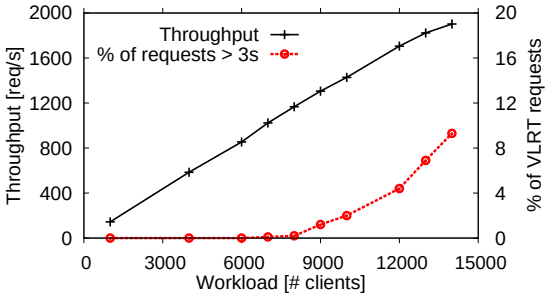


Figure 3: The percentage of VLRT requests starts to grow rapidly starting from 9000 clients.

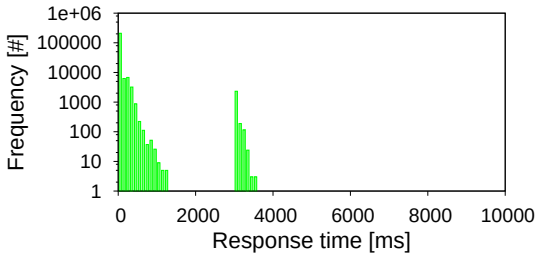
the server CPU becomes saturated for a very short period of time. We note that even though the bottlenecks are very short, the arrival rate of requests (thousands per second) quickly overwhelm the queues in the servers. The final step (5) of each micro-level event analysis identifies a specific cause associated with the very short bottlenecks: Java GC, DVFS, and VM consolidation.

We further provide a systematic discussion of remedies for VLRT requests. Although some causes of VLRT requests can be "fixed" (e.g., Java GC was streamlined from JVM 1.5 to 1.6), other VLRT requests arise from statistical coincidences such as VM consolidation (a kind of noisy neighbor problem) and cannot be easily "fixed". Using very short bottlenecks as a simple, but general model, we discuss the limitations of some potential solutions (e.g., making queues deeper through additional threads causes bufferbloat) and describe generic remedies to reduce or bypass the queue amplification process (e.g., through priority-based job scheduling to reduce queuing of short requests), regardless of the origin of very short bottlenecks.

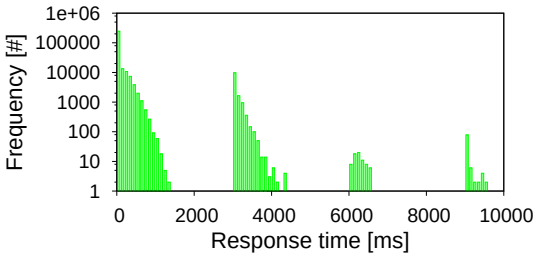
The rest of the paper is organized as follows. Section 2 shows the emergence of VLRT requests at increasing workload and utilization using the Java GC experiments. Section 3 describes the micro-level event analyses that link the varied causes to VLRT requests. Section 4 discusses the remedies for reducing or avoiding VLRT requests using very short bottlenecks as a general model. Section 5 summarizes the related work and Section 6 concludes the paper.

2 VLRT Requests at Moderate Utilization

Large response time fluctuations (also known as the latency long tail problem) of large scale distributed applications happen when very long response time (VLRT) requests arise. VLRT requests have been reported by industry practitioners [18] and academic researchers [24, 27, 38, 43]. These requests are difficult to study, since



(a) 9000 clients; the system throughput is 1306 req/s and the highest average CPU usage among component servers is 61%.



(b) 12000 clients; the system throughput is 1706 req/s and the highest average CPU usage among component servers is 81%.

Figure 4: Frequency of requests by their response times at two representative workloads. The system is at moderate utilization, but the latency long tail problem can be clearly seen.

they happen occasionally and without warning, often at moderate CPU utilization levels. When running by themselves, the VLRT requests change back to normal and return within a few milliseconds. Consequently, the problem does not reside within the VLRT requests, but in the interactions among the system components.

Since VLRT requests arise from system interactions, usually they are not exactly reproducible at the request level. Instead, they appear when performance data are statistically aggregated, as their name "latency long tail" indicates. We start our study by showing one set of such aggregated graphs, using RUBBoS [6], a representative web-facing n-tier system benchmark modeled after Slashdot. Our experiments use a typical 4-tier configuration, with 1 Apache web server, 2 Tomcat Application Servers, 1 C-JDBC clustering middleware, and 2 MySQL database servers (details in Appendix A).

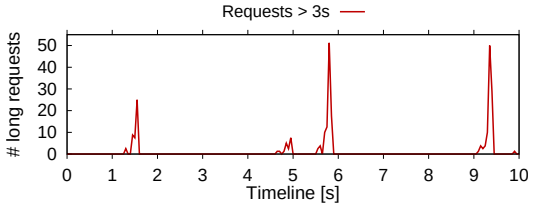
When looking at statistical average metrics such as throughput, VLRT requests may not become apparent immediately. As illustration, Figure 1 shows the throughput and CPU utilization of RUBBoS experiments for workloads from 1000 to 14000 concurrent clients. The average CPU utilization of Tomcat and MySQL rise gradually, as expected. The system throughput grows linearly, since all the system components have yet to reach saturation. Similarly, the aggregate response time graph (Figure 2) show little change up to 12000 clients. Without looking into the distribution of request response time, one might overlook the VLRT problems that start at moderate CPU utilization levels.

Although not apparent from Figure 1, the percentage of VLRT requests (defined as requests that take more than 3 seconds to return in this paper) increases significantly starting from 9000 clients as shown in Figure 3. At the workload of 12000 clients, more than 4% of all requests become VLRT requests, even though the CPU utilization of all servers is only 80% (Tomcat and MySQL) or much lower (Apache and C-JDBC). The latency long tail problem can be seen more clearly when we plot the frequency of requests by their response times in Figure 4 for two representative workloads: 9000 and 12000. At moderate CPU utilization (about 61% at 9000 clients, Figure 4(a)), VLRT requests appear as a second cluster after 3 seconds. At moderately high CPU utilization (about 81% at 12000 clients, Figure 4(b)), we see 3 clusters of VLRT requests after 3, 6, and 9 seconds, respectively. These VLRT requests add up to 4% as shown in Figure 3.

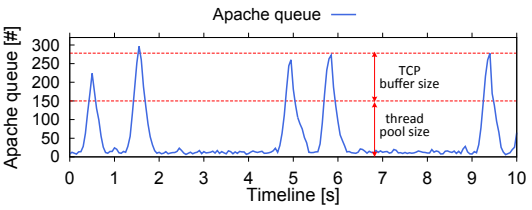
One of the intriguing (and troublesome) aspects of wide response time fluctuations is that they start to happen at moderate CPU utilization level (e.g., 61% at 9000 clients). This observation suggests that the CPU (the critical resource) may be saturated only part of the time, which is consistent with previous work [38, 40] on very short bottlenecks as potential causes for the VLRT requests. Complementing a technical problem-oriented description of very short bottlenecks (Java garbage collection [38] and anti-synchrony from DVFS [40]), we also show that VLRT requests are associated with a more fundamental phenomenon (namely, *very short bottleneck*) that can be described, understood, and remedied in a more general way than each technical problem.

3 VLRT Requests Caused by Very Short Bottlenecks

We use a micro-level event analysis to link the causes of very short bottlenecks to VLRT requests. The micro-level event analysis exploits the fine-grained measurement data collected in RUBBoS experiments. Specifically, all messages exchanged between servers are timestamped at microsecond resolution. In addition, system resource utilization (e.g., CPU) is monitored at short time intervals (e.g., 50ms). The events are shown in a timeline graph, where the X-axis represents the time elapsed during the experiment at fine-granularity (50ms units in this section).



(a) Number of VLRT requests counted at every 50ms time window. Such VLRT requests contribute to bi-modal response time distribution as shown in Figure 4(a).



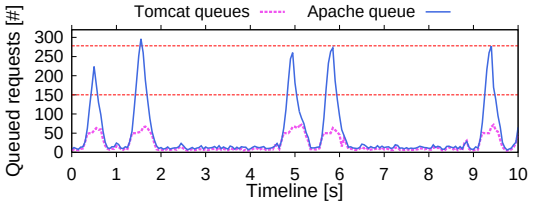
(b) Frequent queue peaks in Apache during the same 10-second timeframe as in (a). The queue peaks match well with the occurrence of the VLRT requests in (a). This arises because Apache drops new incoming packets when the queued requests exceed the upper limit of the queue, which is imposed by the server thread pool size (150) and the operating system TCP stack buffer size (128 by default). Dropped packets lead to TCP retransmissions (>3s).

Figure 5: VLRT requests (see (a)) caused by queue peaks in Apache (see (b)) when the system is at workload 9000 clients.

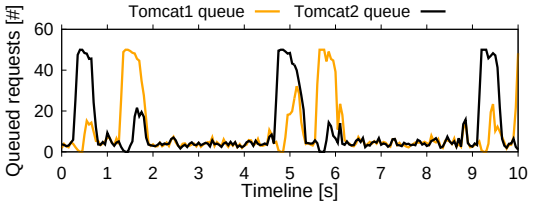
3.1 VLRT Requests Caused by Java GC

In our first illustrative case study of very short bottlenecks, we will establish the link between VLRT requests shown in Figure 4 to the Java garbage collector (GC) in the Tomcat application server tier of the n-tier system. We have chosen Java GC as the first case because it is deterministic and easier to explain. Although Java GC has been suggested as a cause of transient events [38], the following explanation is the first detailed description of data flow and control flow that combine into queue amplification in an n-tier system. This description is a five-step micro-event timeline analysis of fine-grained monitoring based on a system tracing facility that timestamps all network packets at microsecond granularity [40]. By recording the precise arrival and departure timestamps of each client request for each server, we are able to determine precisely how much time each request spends in each tier of the system.

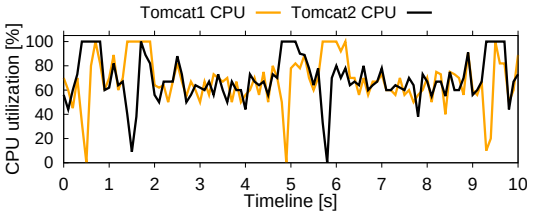
In the first step of micro-event analysis (transient events), we use fine-grained monitoring data to determine which client requests are taking seconds to finish instead of the normally expected milliseconds response time. Specifically, we know exactly at what time these VLRT requests occur. A non-negligible number (up to 50) of such VLRT requests appear reliably (even though



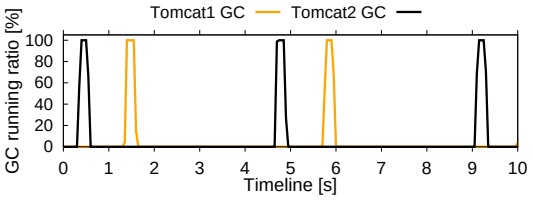
(a) Queue peaks in Apache coincide with the queue peaks in Tomcat, suggesting push-back wave from Tomcat to Apache.



(b) Request queue for each Tomcat server (1 and 2). The sum of the two is the queued requests in the Tomcat tier (see (a)).



(c) Transient CPU saturations of a Tomcat server coincide with the queue peaks in the corresponding Tomcat server (see (b)).



(d) Episodes of Java GC in a Tomcat server coincide with the transient CPU saturation of the Tomcat server (see (c)).

Figure 6: Queue peaks in Apache (a) due to very short bottlenecks caused by Java GC in Tomcat (d).

they may not be exactly the same set for different experiments) at approximately every four seconds as measured from the beginning of each experiment (Figure 5(a)). The X-axis of Figure 5(a) is a timeline at 50ms intervals, showing the clusters of VLRT requests are tightly grouped within a very short period of time. Figure 5(a) shows four peak/clusters of VLRT requests during a 10-second period of a RUBBoS experiment (workload 9000 clients). Outside of these peaks, all requests return within milliseconds, consistent with the average CPU utilization among servers being equal to or lower than 61%.

In the second step of micro-event analysis (retransmitted requests), we show that dropped message packets are likely the cause of VLRT requests. To make this

connection, we first determine which events are being queued in each server. In an n-tier system, we say that a request is waiting in a queue at a given tier when its request packet has arrived and a response has not been returned to an upstream server or client. This situation is the n-tier system equivalent of having a program counter entering that server but not yet exited. Using the same timeframe of Figure 5(a), we plot the request queue length in the Apache server in Figure 5(b). Figure 5(b) shows five peak/clusters, in which the number of queued requests in Apache is higher than 150 for that time interval. The upper limit of the queued requests is slightly less than 300, which is comparable to the sum of thread pool size (150 threads) plus TCP buffer size of 128. Although there is some data analysis noise due to the 50ms window size, the number of queued requests in Apache suggests strongly that some requests may have been dropped, when the thread pool is entirely consumed (using one thread per incoming request) and then the TCP buffer becomes full. Given the 3-second retransmission timeout for TCP (kernel 2.6.32), we believe the overlapping peaks of Figure 5(a) (VLRT requests) and Figure 5(b) (queued requests in Apache) make a convincing case for dropped TCP packets causing the VLRT requests. However, we still need to find the source that caused the requests to queue in the Apache server, since Apache itself is not a bottleneck (none of the Apache resources is a bottleneck).

In the third step of micro-event analysis (queue amplification), we continue the per-server queue analysis by integrating and comparing the requests queued in Apache Figure 5(b) with the requests queued in Tomcat. The five major peak/clusters in Figure 6(a) show the queued requests in both Apache (sharp/tall peaks near the 278 limit) and Tomcat (lower peaks within the sharp/tall peaks). This near-perfect coincidence of (very regular and very short) queuing episodes suggests that it is not by chance, but somehow Tomcat may have contributed to the queued requests in Apache.

Let us consider more generally the situation in n-tier systems where queuing in a downstream server (e.g., Tomcat) is associated with queuing in the upstream server (e.g., Apache). In client/server n-tier systems, a client request is sent downstream for processing, with a pending thread in the upstream server waiting for the response. If the downstream server encounters internal processing delays, two things happen. First, the downstream server's queue grows. Second, the number of matching and waiting threads in the upstream server also grows due to the lack of responses from downstream. This phenomenon, which we call *push-back wave*, appears in Figure 6(a). The result of the third step in micro-event analysis is the connection between long queue in Apache to queuing in Tomcat due to Tomcat saturation.

In the fourth step of micro-event analysis (very short bottlenecks), we will link the Tomcat queuing with very short bottlenecks in which CPU becomes saturated for a very short time (tens of milliseconds). The first part of this step is a more detailed analysis of Tomcat queuing. Specifically, the queued requests in the Tomcat tier (a little higher than 60 in Figure 6(a)), are the sum of two Tomcat servers. The sum is meaningful since a single Apache server uses the two Tomcat servers to process the client requests. To study the very short bottlenecks of CPU, we will consider the request queue for each Tomcat server (called 1 and 2) separately in Figure 6(b). At about 0.5 seconds in Figure 6(b), we can see Tomcat2 suddenly growing a queue that contains 50 requests, due to the concurrency limit of the communication channel between each Apache process and a Tomcat instance (AJP [1] connection pool size set to 50).

The second part of the fourth step is a fine-grained sampling (at 50ms intervals) of CPU utilization of Tomcat, shown in Figure 6(c). We can see that Tomcat2 enters a full (100%) CPU utilization state, even though it is for a very short period of about 300 milliseconds. This short period of CPU saturation is the very short bottleneck that caused the Tomcat2 queue in Figure 6(b) and through push-back wave, the Apache queue in Figure 6(a)). Similar to the Tomcat2 very short bottleneck at 0.5 seconds in Figure 6(b), we can see a similar Tomcat1 very short bottleneck at 1.5 seconds. Each of these very short bottlenecks is followed by similar bottlenecks every four seconds during the entire experiment.

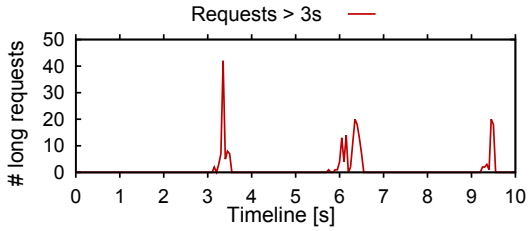
The fifth step of the micro-event analysis (root cause) is the linking of transient CPU bottlenecks to Java GC episodes. Figure 6(d) shows the timeline of Java GC, provided by the JVM GC logging. We can see that both Tomcat1 and Tomcat2 run Java GC at a regular time interval of about four seconds. The timeline of both figures shows that the very short bottlenecks in Figure 6(c) and Java GC episodes happen at the same time throughout the entire experiment. The experiments were run with JVM 1.5, which is known to consume significant CPU resources at high priority during GC. This step shows that the Java GC caused the transient CPU bottlenecks.

In summary, the 5 steps of micro-event analysis show the VLRT requests in Figure 4 are due to very short bottlenecks caused by Java GC:

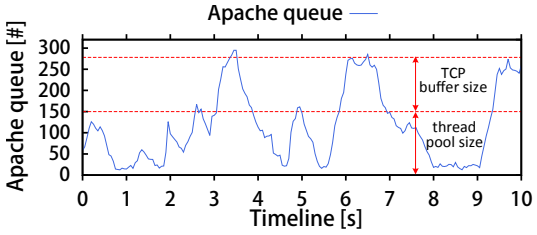
1. Transient events: VLRT requests are clustered within a very short period of time at about 4-second intervals throughout the experiment (Figure 5(a)).
2. Retransmitted requests: VLRT requests coincide with long request queues in the Apache server (Figure 5(b)) that causes dropped packets and TCP retransmission after 3 seconds.
3. Queue amplification: long queues in Apache are caused by push-back waves from Tomcat servers,

Workload	6000	8000	10000	12000
requests > 3s	0	0.3%	0.2%	0.7%
Tomcat CPU util.	31%	43%	50%	61%
MySQL CPU util.	44%	56%	65%	78%

Table 1: Percentage of VLRT requests and the resource utilization of representative servers as workload increases in the SpeedStep case.

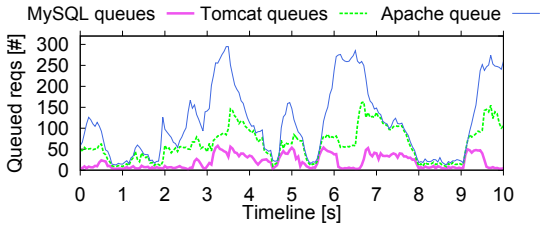


(a) Number of VLRT requests counted at every 50ms time window.

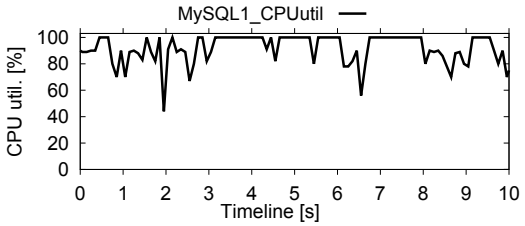


(b) Frequent queue peaks in Apache during the same 10-second time period as in (a). Once a queue spike exceeds the concurrency limit, new incoming packets are dropped and TCP retransmission occurs, causing the VLRT requests as shown in (a).

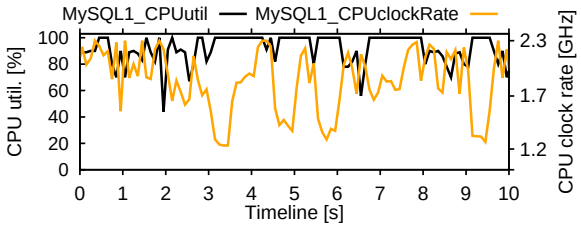
Figure 7: VLRT requests (see (a)) caused by queue peaks in Apache (see (b)) when the system is at workload 12000.



(a) Queue peaks in Apache coincide with the queue peaks in MySQL, suggesting the push-back waves from MySQL to Apache.



(b) Transient CPU saturation periods of MySQL1 coincide with the queue peaks in MySQL (see (a)).



(c) The low CPU clock rate of MySQL1 coincides with the transient CPU saturation periods, suggesting that the transient CPU saturation is caused by the delay of CPU adapting from a slow mode to a faster mode to handle a workload burst.

Figure 8: Queue peaks in Apache (see (a)) due to very short bottlenecks in MySQL caused by the anti-synchrony between workload bursts and DVFS CPU clock rate adjustments (see (c)).

where similar long queues form at the same time (Figure 6(a)).

4. Very short bottlenecks: long queues in Tomcat (Figure 6(b)) are created by very short bottlenecks (Figure 6(c)), in which the Tomcat CPU becomes saturated for a very short period of time (about 300 milliseconds).
5. Root cause: The very short bottlenecks coincide exactly with Java GC episodes (Figure 6(d)).

The discussions on the solutions for avoiding VLRT requests and very short bottlenecks are in Section 4.

3.2 VLRT Requests Caused by Anti-Synchrony from DVFS

The second case of very short bottlenecks was found to be associated with anti-synchrony between workloads bursts and CPU clock rate adjustments made by dynamic voltage and frequency scaling (DVFS). By anti-synchrony we mean opposing cycles, e.g., CPU clock rate changed from high to low after idling, but the slow CPU immediately meets a burst of new requests. Previous work [18, 39] have suggested power saving techniques such as DVFS being a potential source for VLRT requests. The following micro-event analysis will explain in detail the queue amplification process that links anti-synchrony to VLRT requests through very short bottlenecks.

In DVFS experiments, VLRT requests start to appear at 8000 clients (Table 1) and grow steadily with increasing workload and CPU utilization, up to 0.7% of all requests at 12000 clients with 78% CPU in MySQL. These experiments (similar to [39]) had the same setup as Java GC experiments in Section 3.1, with two modifications. First, the JVM in Tomcat was upgraded from 1.5 to 1.6 to reduce the Java GC demands on CPU [3], thus avoiding the very short bottlenecks described in Section 3.1

due to Java GC. Second, the DVFS control (default Dell BIOS level) in MySQL is turned on: Intel Xeon CPU (E5607) supporting nine CPU clock rates, with the slowest (P8, 1.12 GHz) nearly half the speed of the highest (P0, 2.26GHz).

In the first step of micro-event analysis (transient events) for DVFS experiments, we plot the occurrence of VLRT requests (Figure 7(a)) through the first 10-second of experiment with workload of 12000 clients. Three tight clusters of VLRT requests appear, showing the problem happened during a very short period of time. Outside of these tight clusters, all requests return within a few milliseconds.

In the second step of micro-event analysis (dropped requests), the request queue length in Apache over the same period of time shows a strong correlation between peaks of Apache queue (Figure 7(b)) and peaks in VLRT requests (Figure 7(a)). Furthermore, the three high Apache queue peaks rise to the sum of Apache thread pool size (150) and its TCP buffer size (128). This observation is consistent with the first illustrative case, suggesting dropped request packets during those peak periods, even though Apache is very far from saturation (46% utilization).

In the third step of micro-event analysis (queue amplification), we establish the link between the queuing in Apache with the queuing in downstream servers by comparing the queue lengths of Apache, Tomcat, and MySQL in Figure 8(a). We can see that peaks of Apache queue coincide with peaks of queue lengths in Tomcat and MySQL. A plausible hypothesis is queue amplification that starts in MySQL, propagating to Tomcat, and ending in Apache. Supporting this hypothesis is the height of queue peaks for each server. MySQL has 50-request peaks, which is the maximum number of requests sent by Tomcat, with database connection pool size of 50. Similarly, a Tomcat queue is limited by the AJP connection pool size in Apache. As MySQL reaches full queue, a push-back wave starts to fill Tomcat's queues, which propagates to fill Apache's queue. When Apache's queue becomes full, dropped request messages create VLRT requests.

In the fourth step of micro-event analysis (very short bottlenecks), we will link the MySQL queue to very short bottlenecks with a fine-grained CPU utilization plot of MySQL server (Figure 8(b)). A careful comparative examination of Figure 8(b) and Figure 8(a) shows that short periods of full (100%) utilization of MySQL coincide with the same periods where MySQL reaches peak queue length (the MySQL curve in Figure 8(a)). For simplicity, Figure 8(b) shows the utilization of one MySQL server, since the other MySQL shows the same correlation.

The fifth step of the micro-event analysis (root cause) is the linking of transient CPU bottlenecks to the anti-synchrony between workload bursts and CPU clock rate adjustments. The plot of CPU utilization and clock rate of MySQL server shows that CPU saturation leads to a rise of clock rate and non-saturation makes the clock rate slow down (Figure 8(c)). While this is the expected and appropriate behavior of DVFS, a comparison of Figure 8(a), Figure 8(b), and Figure 8(c) shows that the MySQL queue tends to grow while clock rate is slow (full utilization), and fast clock rates tend to empty the queue and lower utilization. Anti-synchrony becomes a measurable issue when the DVFS adjustment periods (500ms in Dell BIOS) and workload bursts (default setting of RUBBoS) have similar cycles, causing the CPU to be in the mismatched state (e.g., low CPU clock rate with high request rate) for a significant fraction of time.

In summary, the 5 steps of micro-event analysis show the VLRT requests in Figure 7(a) are due to very short bottlenecks caused by the anti-synchrony between workload bursts and DVFS CPU clock rate adjustments:

1. Transient events: VLRT requests are clustered within a very short period of time (three times in Figure 7(a)).
2. Retransmitted requests: VLRT requests coincide with periods of long request queues that form in the Apache server (Figure 7(b)) causing dropped packets and TCP retransmission.
3. Queue amplification: The long queues in Apache are caused by push-back waves from MySQL and Tomcat, where similar long queues form at the same time (Figure 8(a)).
4. Very short bottlenecks: The long queue in MySQL (Figure 8(a)) is created by very short bottlenecks (Figure 8(b)), in which the MySQL CPU becomes saturated for a short period of time (ranging from 300 milliseconds to slightly over 1 second).
5. Root cause: The very short bottlenecks are caused by the anti-synchrony between workload bursts and DVFS CPU clock rate adjustments (Figure 8(c)).

3.3 VLRT Requests Caused by Interferences among Consolidated VMs

The third case of very short bottlenecks was found to be associated with the interferences among consolidated VMs. VM consolidation is an important strategy for cloud service providers to share infrastructure costs and increase profit [12, 21]. An illustrative win-win scenario of consolidation is to co-locate two independent VMs with bursty workloads [28] that do not overlap, so the shared physical node can serve each one well and increase overall infrastructure utilization. However, statistically independent workloads tend to have somewhat random bursts, so the bursts from the two VMs sometimes alternate, and sometimes overlap. The interfer-

ences among co-located VMs is also known as the "noisy neighbors" problem. The following micro-event analysis will explain in detail the queue amplification process that links the interferences among consolidated VMs to VLRT requests through very short bottlenecks.

The experiments that study the interferences between two consolidated VMs consist of two RUBBoS n-tier applications, called SysLowBurst and SysHighBurst (Figure 9). SysLowBurst is very similar to the 1/2/1/2 configuration of previous experiments on Java VM and DVFS (Sections 3.1 and 3.2), while SysHighBurst is a simplified 1/1/1 configuration (one Apache, one Tomcat, and one MySQL). The only shared node runs VMware ESXi, with the Tomcat in SysLowBurst co-located with MySQL in SysHighBurst on the same CPU core. All other servers run on dedicated nodes. The experiments use JVM 1.6 and CPUs with disabled DVFS, to eliminate those two known causes of very short bottlenecks.

The experiments evaluate the influence of bursty workloads by using the default RUBBoS workload generator (requests generated following a Poisson distribution parameterized by the number of clients) in SysLowBurst, and observing the influence of increasingly bursty workload injected by SysHighBurst. The workload generator of SysHighBurst is enhanced with an additional burstiness control [29], called *index of dispersion* (abbreviated as I). The workload burstiness $I = 1$ is calibrated to be the same as the default RUBBoS setting, and a larger I generates a burstier workload (for each time window, wider variations of requests created).

The baseline experiment runs SysLowBurst by itself at a workload of 14000 clients (no consolidation), with the result of zero VLRT requests (Table 2, line #1). The consolidation is introduced by SysHighBurst, which has a very modest workload of 400 clients, which is about 3% of SysLowBurst. However, the modest workload of SysHighBurst has an increasing burstiness from $I = 1$ to $I = 400$, when most of SysHighBurst workload become batched into short bursts. The lines #2 through #5 of Table 2 shows the increasing number of VLRT requests as I increases. We now apply the micro-event timeline analysis to confirm our hypothesis that the VLRT requests are caused by the interferences between the Tomcat2 in SysLowBurst and MySQL in SysHighBurst.

In the first step of micro-event analysis (transient events), we plot the occurrence of the VLRT requests of SysLowBurst (Figure 10(a)) during a 15-second of experiment when the consolidated SysHighBurst has $I = 100$ bursty workload. We can see three tight clusters (at 2, 5, and 12 seconds) and 1 broader cluster (around 9 seconds) of VLRT requests appear, showing the problem happened during a relatively short period of time. Outside of these tight clusters, all requests return within a few milliseconds.

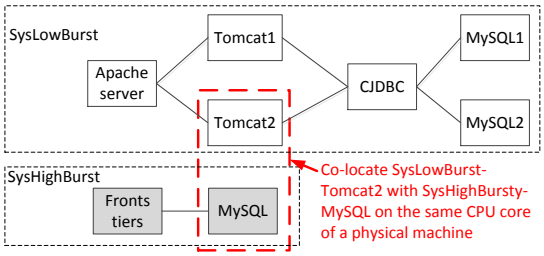


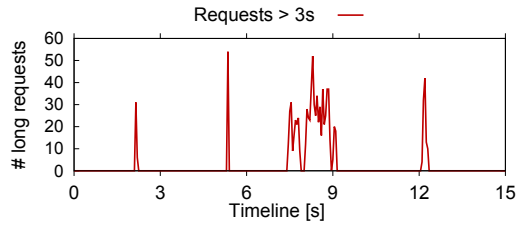
Figure 9: Consolidation strategy between SysLowBurst and SysHighBurst; the Tomcat2 in SysLowBurst is co-located with MySQL in SysHighBurst.

#	SysLowBurst WL	SysLowBurst requests > 3s	SysLowBurst Tomcat2-CPU (%)	SysHighBurst WL	SysHighBurst Burstiness level	SysHighBurst MySQL-CPU (%)
1	14000	**0**	74.1	0	Null	0
2	14000	**0.1%**	74.9	400	I=1	10.2
3	14000	**2.7%**	74.7	400	I=100	10.6
4	14000	**5.0%**	75.5	400	I=200	10.5
5	14000	**7.5%**	75.2	400	I=400	10.8

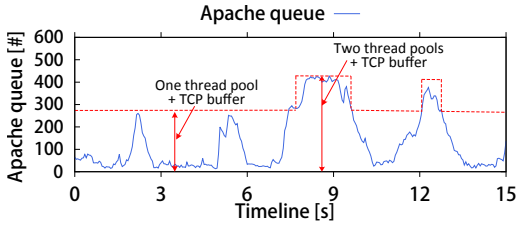
Table 2: Workload of SysLowBurst and SysHighBurst during consolidation. SysLowBurst is serving 14000 clients with burstiness $I = 1$ and SysHighBurst is serving 400 clients but with increasing burstiness levels. As the burstiness of SysHighBurst's workload increases, the percentage of VLRT requests in SysLowBurst increases.

In the second step of micro-event analysis (dropped requests), we found the request queue in the Apache server of SysLowBurst has grown (Figure 10(b)) at the same time as the VLRT requests' peak times (Figure 10(a)). We will consider the two earlier peaks (at 2 and 5 seconds) first. These peaks (about 278, sum of thread pool size and TCP buffer size) are similar to the corresponding previous figures (Figure 5(b) and 7(b)), where requests are dropped due to Apache thread pool being consumed, followed by TCP buffer overflow. The two later peaks (centered around 9 and 12 seconds) are higher (more than 400), reflecting the creation of a second Apache process with another set of thread pools (150). The second process is spawned only when the first thread pool is fully used for some time. We found that packets get dropped during the higher peak periods for two reasons: during the initiation period of the second process (using non-trivial CPU resources, although for a very short time) and after the entire second thread pool has been consumed in a situation similar to earlier peaks.

In the third step of micro-event analysis (queue amplification), we establish the link between queues in Apache with queues in downstream servers by comparing the queue lengths of Apache and Tomcat in Figure 11(a). We can see that the four peaks in Tomcat coincide with the

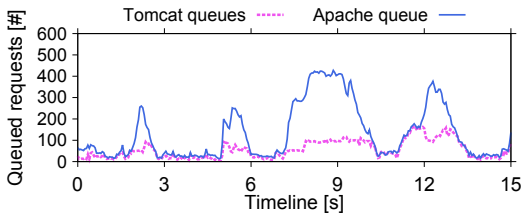


(a) Number of VLRT requests counted at every 50ms time window.

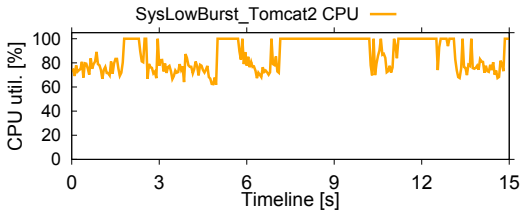


(b) Queue peaks in Apache coincide with the occurrence of the clustered VLRT requests (see (a)), suggesting those VLRT requests are caused by the queue peaks in Apache. Different from Figure 5(b) and 7(b), the Apache server here is configured to have two processes, each of which has its own thread pool. The second process is spawned only when the first thread pool is fully used. However, requests still get dropped when the first thread pool and the TCP buffer are full (at time marker 2 and 5).

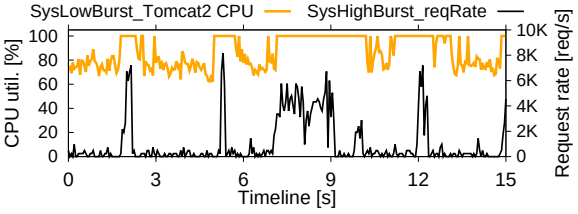
Figure 10: VLRT requests (see (a)) caused by queue peaks in Apache (see (b)) in SysLowBurst when the collocated SysHighBurst is at $I = 100$ bursty workload.



(a) Queue peaks in Apache coincide with those in Tomcat, suggesting the pushback waves from Tomcat to Apache in SysLowBurst.



(b) Transient saturation periods of SysLowBurst-Tomcat2 CPU coincide with the same periods where Tomcat has queue peaks (see (a)).



(c) The workload bursts for SysHighBurst coincide with the transient CPU saturation periods of SysLowBurst-Tomcat2, indicating severe performance interferences between consolidated VMs.

Figure 11: Very short bottlenecks caused by the interferences among consolidated VMs lead to queue peaks in *SysLowBurst-Apache*. The VM interferences is shown in (b) and (c).

queue peaks in Apache (reproduced from the previous figure), suggesting that queues in Tomcat servers have contributed to the growth of queued requests in Apache, since the response delays would prevent Apache to continue. Specifically, the maximum number of requests between each Apache process and each Tomcat is the AJP connection pool size (50 in our experiments). As each Apache process reaches its AJP connection pool size and TCP buffer filled, newly arrived packets are dropped and retransmitted, creating VLRT requests.

In the fourth step of micro-event analysis (very short bottlenecks), we will link the Tomcat queues with the very short bottlenecks in which CPU becomes saturated for a very short period (Figure 11(b)). We can see that the periods of CPU saturation in Tomcat of SysLowBurst coincide with the Tomcat queue peaks (the Tomcat curve in Figure 11(a)), suggesting that the queue peaks in Tomcat are caused by the transient CPU bottlenecks.

The fifth step of the micro-event analysis (root cause) is the linking of transient CPU bottlenecks to the performance interferences between consolidated VMs. This is illustrated in Figure 11(c), which shows the Tomcat2 CPU utilization in SysLowBurst (reproduced from Figure 11(b)) and the MySQL request rate generated by SysHighBurst. We can see a clear overlap between the Tomcat CPU saturation periods (at 2, 5, 7-9, and 12 seconds) and the MySQL request rate jumps due to high workload bursts. The overlap indicates the very short bottlenecks in Tomcat are indeed associated with workload bursts in SysHighBurst, which created a competition for CPU in the shared node, leading to CPU saturation and queue amplification.

In summary, the 5 steps of micro-event analysis show the VLRT requests in Figure 10(a) are due to very short bottlenecks caused by the interferences among consolidated VMs:

1. Transient events: VLRT requests appear within a very short period of time (4 times in Figure 10(a)).
2. Retransmitted requests: The VLRT requests correspond to periods of similar short duration, in which long request queues form in Apache server (Figure 10(b)), causing dropped packets and TCP retransmission after 3 seconds.
3. Queue amplification: The long queues in Apache are caused by push-back waves from Tomcat, where

similar long queues form at the same time (Figure 11(a)).

4. Very short bottlenecks: The long queues in Tomcat (Figure 11(a)) are created by very short bottlenecks (Figure 11(b)), in which the Tomcat CPU becomes saturated for a short period of time.
5. Root cause: The very short bottlenecks are caused by the interferences among consolidated VMs (Figure 11(c)).

4 Remedies for VLRT Requests and Very Short Bottlenecks

4.1 Specific Solutions for Each Cause of VLRT Requests

When Java GC was identified as a source of VLRT requests [38], one of the first questions asked was whether we could apply a "bug fix" by changing the JVM 1.5 GC algorithm or implementation. Indeed this happened when JVM 1.6 replaced JVM 1.5. The new GC implementation was about an order of magnitude less demanding of CPU resources, and its impact became less noticeable at workloads studied in Section 3.1. A similar situation arose when DVFS [39] was confirmed as another source of VLRT requests due to anti-synchrony between workload bursts and DVFS power/speed adjustments. Anti-synchrony could be avoided by changing (reducing) the control loop to adjust CPU clock rate more often, and thus disrupt the anti-synchrony for the default RUBBoS workload bursts. Finally, interferences among consolidated VMs may be prevented by specifying complete isolation among the VMs, disallowing the sharing of CPU resources. Unfortunately, the complete isolation policy also defeats the purpose of sharing, which is to improve overall CPU utilization through sharing [23].

As new sources of VLRT requests such as VM consolidation (Section 3.3) continue to be discovered, and suggested by previous work [18, 30], the "bug fix" approach may be useful for solving specific problems, but it probably would not scale, since it is a temporary remedy for each particular set of configurations with their matching set of workloads. As workloads and system components (both hardware and software) evolve, VLRT requests may arise again under a different set of configuration settings. It will be better to find a more general approach to resolve entire classes of problems that cause VLRT requests.

4.2 Solutions for Very Short Bottlenecks

We will discuss potential and general remedies using very short bottlenecks as a simple model, regardless of what caused the VLRT requests (three very different causes of very short bottlenecks were described in Section 3). For this discussion, a very short bottleneck is a very short period of time (from tens to hundreds of milliseconds) during which the CPU remains busy and thus continuously unavailable for lower priority threads and processes at kernel, system, and user levels. The usefulness of the very short bottleneck model in the identification of causes of VLRT requests has been demonstrated in Section 3, where VLRT requests were associated with very short bottlenecks in three different system layers.

In contrast to the effect-to-cause analysis in Section 3, the following discussion of general remedies will follow the chronological order of events, where very short bottlenecks happen first, causing queue amplification, and finally retransmitted VLRT requests. For concreteness, we will use the RUBBoS n-tier application scenario; the discussion applies equally well to other mutually-dependent distributed systems.

First, we will consider the disruption of very short bottleneck formation. From the description in Section 3, there are several very different sources of very short bottlenecks, including system software daemon processes (e.g., Java GC), predictable control system interferences (e.g., DVFS), and unpredictable statistical interferences (e.g., VM co-location). A general solution that is independent of any causes would have to wait for a very short bottleneck to start, detect it, and then take remedial action to disrupt it. Given the short lifespan of a very short bottleneck, its reliable detection becomes a significant challenge. Using a control system terminology, if we trigger the detection too soon ((e.g., a few milliseconds)), we have fast but unstable response. Similarly, if we wait too long in the control loop (e.g., serveral seconds), we may have more stable response but the damage caused by very short bottleneck may have already been done. This argument does not prove that the cause-agnostic detection and disruption of a very short bottleneck is impossible, but it is a serious research challenge.

Second, we will consider the disruption of the queue amplification process. A frequently asked question is whether lengthening the queues in servers (e.g., increasing TCP buffer size or thread pool size in Apache and Tomcat) can disrupt the queue amplification process. There are several reasons for large distributed systems to limit the depth of queues in components. At the network level (e.g., TCP), large network buffer size causes problems such as bufferbloat [20], leading to long latency and poor system performance. At the software systems level, over allocation of threads in web servers can cause significant overhead [41, 42], consuming critical bottleneck resources such as CPU and memory and degrade system performance. Therefore, the queue lengths in servers should remain limited.

On the other hand, the necessity for limitation in server queues does not mean that queue amplification is inevitable. An implicit assumption in queue amplification is the synchronous request/response communication style in current n-tier system implementations (e.g., with Apache and Tomcat). It is possible that asynchronous servers (e.g., nginx [4]) may behave differently, since it does not use threads to wait for responses and therefore it may not propagate the queuing effect further upstream. This interesting area (changing the architecture of n-tier systems to reduce mutual dependencies) is the subject of ongoing active research.

Another set of alternative techniques have been suggested [18] to reduce or bypass queue-related blocking. An example is the creation of multiple classes of requests [38], with a differentiated service scheduler to speed up the processing of short requests so they do not have to wait for VLRT requests. Some applications allow semantics-dependent approaches to reduce the latency long tail problem. For example, (read-only) web search queries can be sent to redundant servers so VLRT requests would not affect all of the replicated queries. These alternative techniques are also an area of active research.

Third, we will consider the disruption of retransmitted requests due to full queues in servers. Of course, once a packet has been lost it is necessary to recover the information through retransmission. Therefore, the question is about preventing packet loss. The various approaches to disrupt queue amplification, if successful, can also prevent packet loss and retransmission. Therefore, we consider the discussion on disruption of queue amplification to subsume the packet loss prevention problem. A related and positive development is the change of the default TCP timeout period from 3 seconds to 1 second in the Linux kernel [22].

Fourth, we return to the Gartner report on average data center utilization of 18% [37]. An empirically observed condition for the rise of very short bottlenecks is a moderate or higher average CPU utilization. In our experiments, very short bottlenecks start to happen at around 40% average CPU utilization. Therefore, we consider the reported low average utilization as a practical (and expensive) method to avoid the very short bottleneck problem. Although more research is needed to confirm this conjecture, low CPU utilization levels probably help prevent very short bottleneck formation as well as queue formation and amplification.

5 Related Work

Latency has received increasing attention in the evaluation of quality of service provided by computing clouds and data centers [10, 27, 31, 35, 38, 40]. Specifically, the long-tail latency is of particular concern for mission-critical web-facing applications [8, 9, 18, 26, 43]. On the solution side, many previous research [26,27] focuses on a single server/platform, not on multi-tier systems which have more complicated dependencies among component servers. Dean et al. [18] described their efforts to bypass/mitigate tail latency in Google's interactive applications. These bypass techniques are effective in specific applications or domains, contributing to an increasingly acute need to improve our understanding of the general causes for the VLRT requests.

Aggregated statistical analyses over fine-grained monitored data have been used to infer the appearance and causes of long-tail latency [17, 25, 27, 40]. Li et al. [27] measure and compare the changes of latency distributions to study hardware, OS, and concurrency-model induced causes of tail latency in typical web servers executing on multi-core machines. Wang et al. [40] propose a statistical correlation analysis between a server's fine-grained throughput and concurrent jobs in the server to infer the server's real-time performance state. Cohen [17] use a class of probabilistic models to correlate system-level metrics and threshold values with high-level performance states. Our work leverages the fine-grain data, but we go further in using micro-level timeline event analysis to link the various causes to VLRT requests.

Our work makes heavy use of data from fine-grained monitoring and profiling tools to help identify causes associated with the performance problem [2, 5, 7, 13, 14, 25, 32, 34]. For example, Chopstix [13] continuously collects profiles of low-level OS events (e.g., scheduling, L2 cache misses, page allocation, locking) at the granularity of executables, procedures and instruction. Collectl [2] provides the ability to monitor a broad set of system level metrics such as CPU and I/O operations at millisecond-level granularity. We use these tools when applicable.

Techniques based on end-to-end request-flow tracing have been proposed for performance anomaly diagnosis [7, 11, 16, 19, 25, 33, 36], but usually for systems with low utilization levels. X-ray [11] instruments binaries as applications execute and uses dynamic information flow tracking to estimate the likelihood that a block was executed due to each potential root cause for the performance anomaly. Fay [19] provides dynamic tracing through use of runtime instrumentation and distributed aggregation within machines and across clusters for windows platform. Aguilera et al. [7] propose a statistical method to infer request trace between black boxes in a distributed system and attribute delays to specific nodes. BorderPatrol [25] obtains request traces more precisely using active observation which carefully modifies the event stream observed by component servers.

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

(a) Software setup

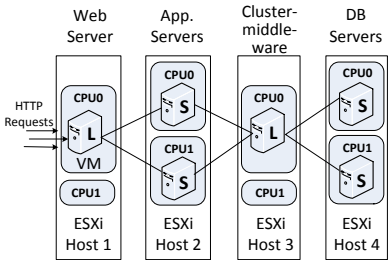
ESXi Host Configuration

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

(b) ESXi host and VM setup



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

Figure 12: Details of the experimental setup.

6 Conclusion

Applying a micro-level event analysis on extensive experimental data collected from fine-grain monitoring of n-tier application benchmarks, we demonstrate that the latency long tail problem can have several causes at three system layers. Specifically, very long response time (VLRT) requests may arise from CPU DVFS control at the architecture layer (Section 3.2), Java garbage collection at the system software layer (Section 3.1), and interferences among virtual machines (VM) in VM consolidation at the VM layer (Section 3.3). Despite their different origins, these phenomena can be modeled and described as *very short bottlenecks* (tens to hundreds of milliseconds). The micro-level event analysis shows the VLRT requests are coincidental to very short bottlenecks in various servers, which in turn amplify queuing in upstream servers, quickly leading to TCP buffer overflow and request retransmission, causing VLRT requests of several seconds.

We discuss several approaches to remedy the emergence of VLRT requests, including cause-specific "bug-fixes" (Section 4.1) and more general solutions to reduce queuing based on the very short bottleneck model (Section 4.2) that will work regardless of the origin of VLRT requests. We believe that our study of very short bottlenecks uncovered only the "tip of iceberg". There are probably many other important causes of very short bottlenecks such as background daemon processes that cause "multi-millisecond hiccups" [18]. Our discussion in Section 4 suggests that the challenge to find effective remedies for very short bottlenecks has only just begun.

7 Acknowledgement

We thank the anonymous reviewers and our shepherd, Liuba Shrira, for their feedback on improving this paper. This research has been partially funded by National Science Foundation by CNS/SAVI (1250260, 1402266), IUCRC/FRP (1127904), CISE/CNS (1138666), NetSE (0905493) programs, and gifts, grants, or contracts from Fujitsu, Singapore Government, 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.

A Experimental Setup

We adopt the RUBBoS standard n-tier benchmark, based on bulletin board applications such as Slashdot [6]. 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 [15]) system. The workload consists of 24 different web interactions, each of which is a combination of all processing activities that deliver an entire web page requested by a client, i.e., generate the main HTML file as well as retrieve embedded objects and perform related database queries. These interactions aggregate into two kinds of workload modes: browse-only and read/write 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. 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 12 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 (Apache), application servers, clustering middleware servers (C-JDBC), and database servers. Figure 12(c) shows a sample 1/2/1/2 topology. Each server runs on top of one VM. Each ESXi host runs the VMs from the same tier of the application. Apache and C-JDBC are deployed in type "L" VMs to avoid bottlenecks in load-balance tiers.

References

[1] The AJP connector. "http://tomcat.apache.org/tomcat-7.0-doc/config/ajp.html".

[2] Collectl. "http://collectl.sourceforge.net/".

[3] Java SE 6 performance white paper. "http://java.sun.com/performance/reference/whitepapers/6_performance.html".

[4] NGINX. "http://nginx.org/".

[5] Oprofile. "http://oprofile.sourceforge.net/".

[6] RUBBoS: Bulletin board benchmark. "http://jmob.ow2.org/rubbos.html".

[7] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 74–89, 2003.

[8] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[9] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, pages 253–266, 2012.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[11] M. Attariyan, M. Chow, and J. Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, pages 307–320, 2012.

[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 164–177, 2003.

[13] S. Bhatia, A. Kumar, M. E. Fiuczynski, and L. Peterson. Lightweight, high-resolution monitoring for troubleshooting production systems. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, pages 103–116, 2008.

[14] B. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic instrumentation of production systems. In *Proceedings of the 2004 USENIX Annual Technical Conference*, pages 15–28, 2004.

[15] E. Cecchet, J. Marguerite, and W. Zwaenepole. C-JDBC: Flexible database clustering middleware. In *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track*, pages 9–18, 2004.

[16] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of the 32th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2002)*, pages 595–604, 2002.

[17] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)*, pages 231–244, 2004.

[18] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[19] Ú. Erlingsson, M. Peinado, S. Peter, M. Budiu, and G. Mainar-Ruiz. Fay: Extensible distributed tracing from kernels to clusters. *ACM Transactions on Computer Systems (TOCS)*, 30(4):13, 2012.

[20] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the internet. *Communications of the ACM*, 55(1):57–65, 2012.

[21] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam. Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC 2011)*, page 22, 2011.

[22] IETF. RFC 6298. "http://tools.ietf.org/html/rfc6298".

[23] Y. Kanemasa, Q. Wang, J. Li, M. Matsubara, and C. Pu. Revisiting performance interference among

consolidated n-tier applications: Sharing is better than isolation. In *Proceedings of the 10th IEEE International Conference on Services Computing (SCC 2013)*, pages 136–143, 2013.

[24] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat. Chronos: Predictable low latency for data center applications. In *Proceedings of the 3rd ACM Symposium on Cloud Computing (SoCC 2012)*, pages 9:1–9:14, 2012.

[25] E. Koskinen and J. Jannotti. Borderpatrol: Isolating events for black-box tracing. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 191–203, 2008.

[26] J. Leverich and C. Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 4:1–4:14, 2014.

[27] J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. Technical Report UW-CSE14-04-01, Department of Computer Science & Engineering, University of Washington, April 2014.

[28] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *Proceedings of the ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008)*, pages 265–286, 2008.

[29] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *Proceedings of the 6th International Conference on Autonomic computing (ICAC 2009)*, pages 149–158, 2009.

[30] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. DeepDive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the 2013 USENIX Annual Technical Conference*, pages 219–230, 2013.

[31] D. A. Patterson. Latency lags bandwith. *Communications of the ACM*, 47(10):71–75, 2004.

[32] V. Prasad, W. Cohen, F. Eigler, M. Hunt, J. Keniston, and B. Chen. Locating system problems using dynamic instrumentation. In *Proceedings of the 2005 Ottawa Linux Symposium*, pages 49–64, 2005.

[33] P. Reynolds, C. E. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting the unexpected in distributed systems. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI'06)*, pages 115–128, 2006.

[34] Y. Ruan and V. S. Pai. Making the" box" transparent: System call performance as a first-class result. In *Proceedings of the 2004 USENIX Annual Technical Conference*, pages 1–14, 2004.

[35] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout. It's time for low latency. In *Proceedings of the 13th USENIX Workshop on Hot Topics in Operating Systems (HotOS 13)*, pages 11–11, 2011.

[36] R. R. Sambasivan, A. X. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing performance changes by comparing request flows. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*, pages 43–56, 2011.

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

[38] Q. Wang, Y. Kanemasa, M. Kawaba, and C. Pu. When average is not average: large response time fluctuations in n-tier systems. In *Proceedings of the 9th International Conference on Autonomic computing (ICAC 2012)*, pages 33–42, 2012.

[39] Q. Wang, Y. Kanemasa, C.-A. Li, Jack Lai, M. Matsubara, and C. Pu. Impact of dvfs on n-tier application performance. In *Proceedings of ACM Conference on Timely Results in Operating Systems (TRIOS 2013)*, pages 33–42, 2013.

[40] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu. Detecting transient bottlenecks in n-tier applications through fine-grained analysis. In *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems (ICDCS 2013)*, pages 31–40, 2013.

[41] 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 *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2011)*, pages 1034–1045, 2011.

[42] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, pages 230–243, 2001.

[43] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey. Bobtail: Avoiding long tails in the cloud. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, pages 329–342, 2013.