Impact of DVFS on n-Tier Application Performance

Qingyang Wang [1], Yasuhiko Kanemasa [2], Jack Li [1], Chien An Lai [1]
Masazumi Matsubara [2], Calton Pu [1]

[1] *College of Computing, Georgia Institute of Technology*
[2] *System Software Laboratories, FUJITSU LABORATORIES LTD.*

Abstract

Dynamic Voltage and Frequency Scaling (DVFS) has been widely deployed and proven to reduce energy consumption at low CPU utilization levels; however, our measurements of the n-tier application benchmark (RUBBoS) performance showed significant performance degradation at high utilization levels, with response time several times higher and throughput loss of up to 20%, when DVFS is turned on. Using a combination of benchmark measurements and simulation, we found two kinds of problems: large response time fluctuations due to push-back wave queuing in n-tier systems and throughput loss due to rapidly alternating bottlenecks. These problems arise from anti-synchrony between DVFS adjustment period and workload burst cycles (similar cycle length but out of phase). Simulation results (confirmed by extensive measurements) show the anti-synchrony happens routinely for a wide range of configurations. We show that a workload-sensitive DVFS adaptive control mechanism can disrupt the anti-synchrony and reduce the performance impact of DVFS at high utilization levels to 25% or less of the original.

1 Introduction

Dynamic Voltage and Frequency Scaling (DVFS) technologies (e.g., SpeedStep for Intel, Cool'n'Quiet and PowerNow for AMD) have been widely deployed in current generation of processors used in data centers. Their support for variable CPU clock rates and power consumption levels enables power savings for low CPU utilization levels. DVFS has a simple control algorithm: reduce the CPU clock rate when utilization is low, and raise the clock rate when utilization becomes high. Control of DVFS can be done in the operating system (OS) kernel or firmware (e.g., BIOS), and it works well for stable workloads for which the need for adjustments arises infrequently [13, 20, 31].

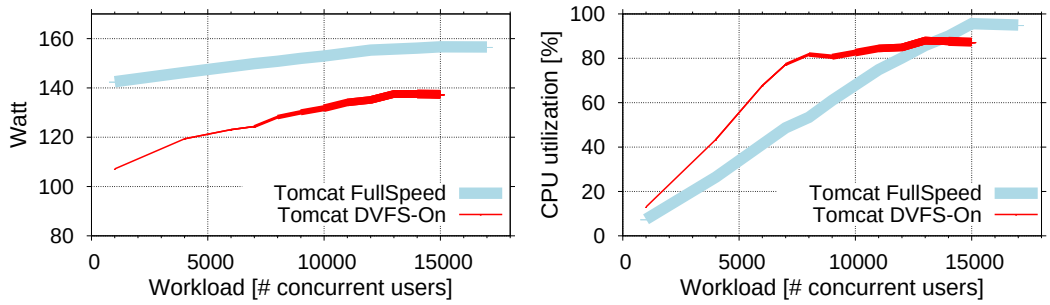
However, there are important classes of applications that have bursty workloads and computing infrastructures that should run at high utilization levels. Concretely, web-facing applications (e.g., e-commerce and news) are known to have bursty workloads.[1] Bursty workloads oscillate between high and low CPU requirements, causing potential mismatches with respect to DVFS adjustments. Furthermore, data centers need to run at high utilization levels to achieve high return on investment. Our study shows that DVFS indeed has significant impact on n-tier application performance at high CPU utilization levels, a result that have high potential impact on data centers.

The first contribution of the paper is an experimental measurement of DVFS impact on n-tier application benchmark (RUBBoS) performance at high utilization. We found two kinds of problems that cause the significant performance losses. First, large response time fluctuations happen, due to push-back wave queuing for database (MySQL) and application servers (Tomcat) in the n-tier system. Second, overall throughput decreases by up to 20% due to rapidly alternating bottlenecks (a form of multi-bottlenecks) between MySQL and Tomcat servers. These measurement results are refined by a detailed simulation study of both problems.

The second contribution of the paper is a pair of simu-

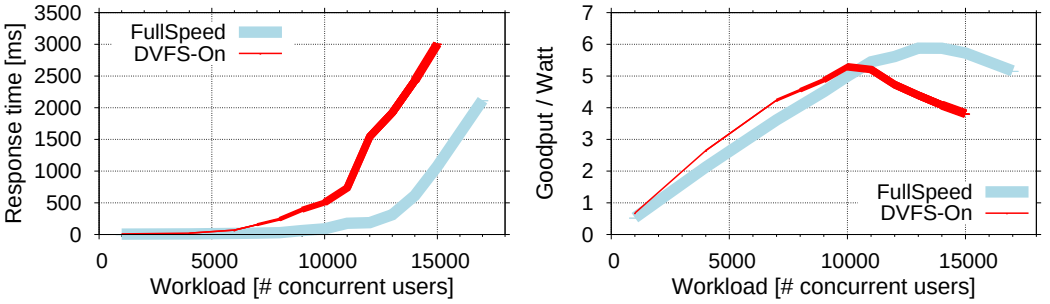
TRIOS'13, November 03 2013, Farmington, PA, USA.
 ACM 978-1-4503-2463-2/13/11...$15.00.
http://dx.doi.org/10.1145/2524211.2524220

[1] The popular term *Slashdot effect* describes a phenomenon where a web page linked by a popular blog or media site suddenly experiences a huge increase in web traffic [3].



(a) Significant power savings with DVFS-On in Tomcat under all workloads

(b) Tomcat CPU utilzation; the variable widths of the DVFS-On line represent different CPU clock rates.



(c) System response time increases more due to DVFS-On

(d) Power and SLA-sensitive throughput comparison (goodput(2s)/power consumption).

Figure 1: Advantage and disadvantage of DVFS turned on. Figure 1(a) shows DVFS can save power at all workloads in the range while Figure 1(c) shows that DVFS leads to a significant response time increase after workload 5,000. Figure 1(d) shows DVFS power efficiency, which indicates that DVFS wins at low workload range (before workload 10,000) while loses at high workload range (after workload 10,000). Goodput here means the throughput with profitable response time (e.g., $< 2s$).

lation studies to determine an upper bound of DVFS impact and methods to reduce it. The first study determined an upper bound of DVFS impact for representative n-tier application workloads, which happens when the workload burst cycles and DVFS adjustment periods are in anti-synchrony. The second study evaluated the effectiveness of two OS-level solutions to reduce the DVFS impact. The first solution is increasing the frequency of DVFS adjustments to eliminate the anti-synchronous cycles. The second solution is a workload-sensitive adaptive control mechanism that changes the DVFS adjustment period to disrupt anti-synchrony.

The rest of the paper is organized as follows. Section 2 introduces the problem of DVFS impact on n-tier application performance. Section 3 describes the two kinds of problems that cause DVFS-related performance loss push-back wave queuing and rapidly alternating bottlenecks. Section 4 presents the calculation and simulation-based determination of an upper bound of the DVFS impact. Section 5 describes two methods to reduce the DVFS impact by disrupting the anti-synchrony: increasing DVFS adjustment frequency and workload-sensitive adaptive control. Section 6 summarizes related work. Section 7 concludes the paper.

2 Problem Description

We started our experimental study of DVFS to confirm its power savings for n-tier application workloads such as the RUBBoS [1] benchmark based on Slashdot. A sample 4-tier configuration (1/2/1/2) (Figure 19(c) in Appendix A with other technical details) denotes 1 web server (Apache), 2 application servers (Tomcat), 1 database clustering middleware (C-JDBC), and 2 database servers (MySQL). The measurement results (Figure 1(a)) show DVFS achieving expected power savings. The thick line in the figure represents DVFS off (called FullSpeed) and the variable-width line represents DVFS on (managed by Dell Active Power Controller at BIOS level). The width of the DVFS-On line is proportional to the actual CPU clock rate (thinnest = slowest

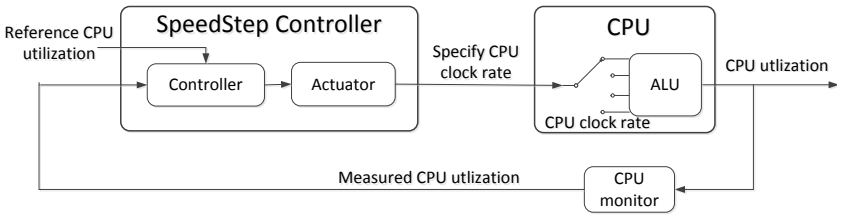


Figure 2: DVFS CPU clock rate control infrastructure

clock, P8, and thickest = FullSpeed, P0). Figure 1(a) and other subfigures in Figure 1) show the average CPU clock rate of Tomcat in the DVFS-On case gradually increasing as workload increases.

The performance degradation at high utilization levels can be seen clearly in Figure 1(c). Starting from workload 5,000, DVFS-On response time becomes longer, e.g., it is 126ms higher than FullSpeed at workload 8,000. This is an important practical problem since in a data center (where the aggregated power consumption is most conspicuous) we need high utilization levels in order to achieve high return on investment (ROI). A more careful calculation of cost-sensitive performance metrics showed that the performance loss problem is serious even when we take into account the power savings.

Figure 1(d) shows an analysis of throughput that is sensitive to power consumption and SLA (service level agreement) specifying a revenue model [25] that limits the profitable response time to 2 seconds. The lines represent goodput divided by power consumption (profitable jobs per watt). The constant line (FullSpeed) shows that peak profit is achieved at 14,000 workload (corresponding to about 90% CPU utilization) and the variable line (DVFS-On) with peak profit at 10,000 workload (about 80% CPU at predominantly less than full clock rate). This analysis shows that the power savings in Figure 1(a) actually carries a non-trivial cost in performance degradation where it matters the most in a data center (high utilization levels).

Figure 1(b) shows a direct comparison of the CPU utilization levels of the Tomcat server as the workload increases to 15,000 clients. Other than a small gap at the very high end (DVFS-On tapering lower than FullSpeed at 15,000 workload), the graph contains no obvious indications of problems with DVFS. Taken together, the graphs in Figure 1 show that the performance degradation due to DVFS at high utilization levels has clearly visible symptoms, yet its causes are non-trivial.

Our investigation traced the performance degradation to three intertwined problem factors. First, although DVFS works well for steady workloads, the natural fluctuations in web-facing application workloads cause mismatches between the DVFS setting and workload intensity. Second, natural workload fluctuations (default setting of RUBBoS) may cause large response time fluctuations when in anti-synchrony (out of phase) with respect to the DVFS periodic adjustments. Third, dependencies among the n-tier servers (e.g., between the application server and database server) may cause rapidly alternating bottlenecks and limit performance. These factors are studied in the following sections.

3 Measurement Results: Two Performance Problems

Our experiments measured the performance of the RUBBoS benchmark on Dell servers with Intel Xeon processors (details in Appendix A), with SpeedStep (Intel's DVFS) turned on and off. The main goal was to evaluate the impact of DVFS with respect to the interesting phenomena reported previously at the application level: large response time fluctuations [32] and rapidly alternating bottlenecks [33]. We used the RUBBoS default bursty workload generator (with a few bug fixes) to make our experimental results comparable to previous work. At first glance, it seems rather unlikely that DVFS would be a factor, since its time scale (tens to few hundreds of milliseconds) was apparently too short for application level measurements (often done at periods of multiple seconds). We decided to proceed with the study for two reasons. First, DVFS has not been ruled out completely as a potential cause for those phenomena. Second, the web interactions in RUBBoS and production applications have become short (e.g., milliseconds) due to continual advances of processor and memory technologies. To study large response time fluctuations (Section 3.1) and rapidly alternating bottlenecks (Section 3.2), we monitor system events at fine time granularity (microseconds). Detailed simulation-based studies complement the measurements by enabling the exploration of configurations and settings that escape OS-level control.

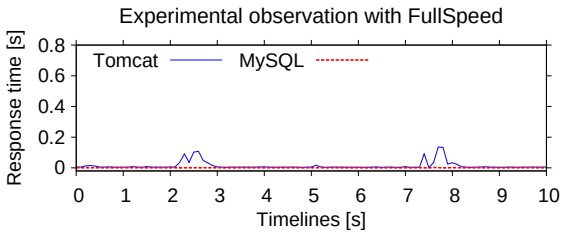
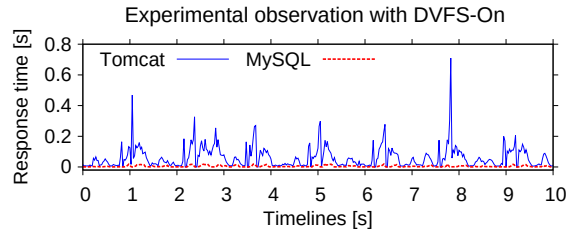
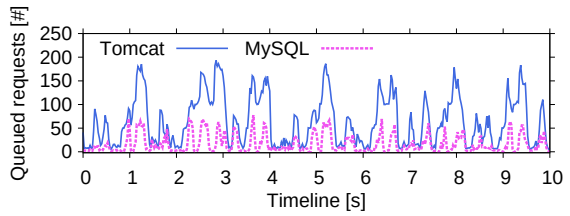


Figure 3: Experimental observation of small response time fluctuations of the system with FullSpeed at workload 8,000.

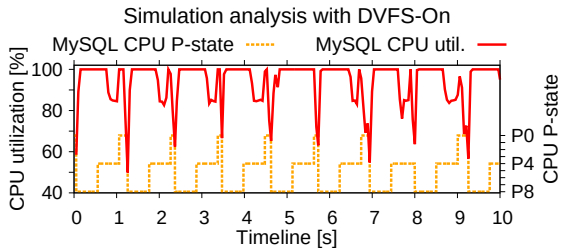


(a) Fine-grained response time measurements in each tier

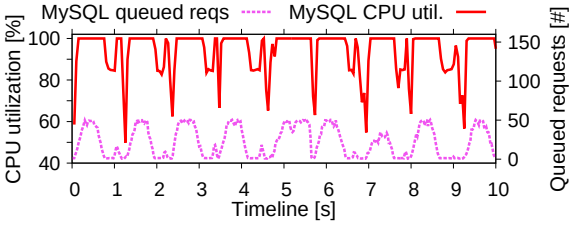


(b) Fine-grained queued request length measurements in each tier

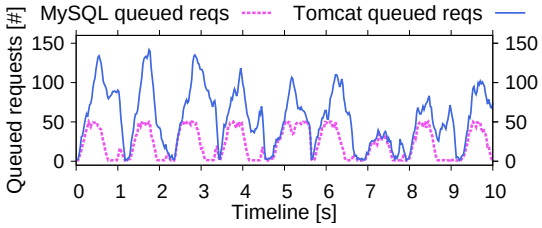
Figure 4: Experimental observation of large response time fluctuations of the system with DVFS-On at WL 8,000. The response time peaks in Figure 4(a) correspond to the peaks of queued requests in Figure 4(b).



(a) MySQL CPU utilization and MySQL CPU P-state; "wrong" P-state frequently causes short-term CPU saturations.



(b) MySQL CPU utilization and MySQL queued requests; short-term CPU saturations cause queued requests in MySQL.



(c) Queued requests in MySQL and Tomcat; once queued requests in MySQL reach a threshold (e.g., due to limited DB connections), requests start to queue in Tomcat tier (pushed back).

Figure 5: Simulation analysis of large response time fluctuations of the system with DVFS at WL 8,000. "Wrong" P-state in MySQL (Figure 5(a)) not only causes queued requests in MySQL (Figure 5(b)), but also in Tomcat due to push-back waves (Figure 5(c)).

3.1 Large Response Time Fluctuations Due to Push-Back Wave Queuing

One of the recent interesting discoveries that distinguish n-tier applications from embarrassingly parallel applications such as MapReduce/Hadoop is the existence of inter-tier resource dependencies [32], which can amplify load fluctuations across tiers. Specifically, small response time fluctuations in the database tier can be amplified and propagated to the front tiers, particularly at high utilization levels, a phenomenon we call *push-back wave*, since the increased response time propagates from source (database server) towards the client. The first question that we want to answer with our experiments is whether DVFS adjustments will have any effect in n-tier application response time, caused by push-back waves. This is a long shot, since the DVFS adjustments happen at short time intervals (sub-seconds or less) compared to observed push-back waves (seconds).

Our measurements with FullSpeed (DVFS off) confirm this first impression. Figure 3 contains a 10-second representative segment of an experiment (workload of 8,000 with the graph drawn at 50ms time granularity), which shows only small occasional response time fluctuations. As a baseline, the graph shows that the default workload is not overly bursty and the configuration settings of the experiment are reasonable.

When we ran the same experiments with DVFS turned on (Figure 4(a)), we were surprised by significant response time fluctuations at fine-grain time intervals (each "wave" lasting less than a second). Since the hardware configuration, software configuration, and benchmark workload for these two sets of experiments are ex-

actly the same, and the only difference is DVFS being on or off, we can conclude that DVFS is causing the large response time fluctuations in Figure 4(a).

We confirmed that the large response time fluctuations in Figure 4(a) are caused by the push-back wave phenomenon through a detailed analysis of the waiting queue length at each tier. Concretely, Figure 4(b) shows the number of queued requests in each tier of the system in the DVFS-On case. This figure shows that many requests are frequently queued in each tier (especially Tomcat) of the system. The peaks in response time in Figure 4(a) correlate strongly with high numbers of queued requests in Figure 4(b). Concretely, once the queued requests in MySQL reach $\tilde{5}0$, the queued requests in Tomcat also increase significantly, since the DB connection pool in Tomcat is fully used.

Due to the lack of on-chip hardware sensors, we use a detailed simulator (Apendix B) to study the CPU clock rate adjustments made by the DVFS mechanism as implemented at the BIOS level in the Dell server. Figure 5(a) shows the CPU utilization of MySQL (near the top) and CPU clock rate at the bottom. The graph shows that a workload increase at the low clock rate (P8 state) causes the CPU to become 100% utilized, leading to clock rate adjustments to P4 and then P0 (fastest clock rate). A combination of the fast CPU and workload decrease then reduces the CPU utilization, which triggers the DVFS adjustment from P0 to P8. The cycle repeats itself, with varied time periods in each state, but it remains a stable cycle.

We also use the simulator to confirm our hypothesis that push-back waves are indeed related to the large response time fluctuations. Figure 5(b) shows MySQL CPU utilization and the number of queued requests in MySQL. The simulation data shows that when MySQL CPU is 100% utilized (regardless of CPU clock rate), requests start to queue in MySQL until it exhausts the DB thread pool to handle DB connections from Tomcat (set at 50 to match the measurements described above). The push-back waves can be seen explicitly in Figure 5(c), which superimposes the number of queued requests in both MySQL and Tomcat. The simulation data confirms that once the queued requests in MySQL reaches 50, additional requests are queued in Tomcat, which confirms the experimental data in Figure 4(b).

An astute reader might have noticed two kinds of waves in Figure 5(c). The first kind is the superimposed (single peak) wave described above, e.g., waves with peaks at markers 2 and 5 on the X-axis. The second kind of waves has a small notch in addition to the main peak, e.g., waves with peaks at 1 and 4. In contrast to the main peak (when both queues increase and decrease at the same time), the small notch indicates that the Tomcat queue length increased while the MySQL queue length decreased. For the second kind of wave, the first peak shows the push-back and the small notch indicates a new phenomenon: the rapidly alternating bottleneck that we describe in the following section (3.2).

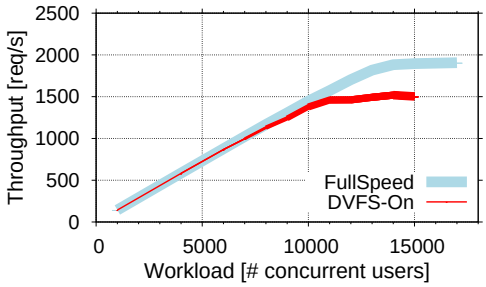


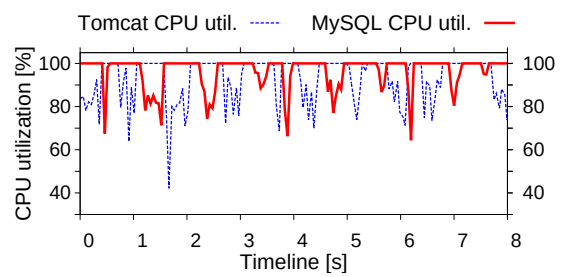
Figure 6: Experimental observation of significant throughput loss of the system with DVFS-On.

Readers may also wonder why can't we just increase the number of DB connections in Tomcat to avoid the push-back waves from MySQL to Tomcat. As Wang et al. [34] pointed out that purely increasing the allocation of soft resources (including DB connections in Tomcat) in an n-tier system can cause significant overhead for the bottleneck resources of the system and degrade the system performance.

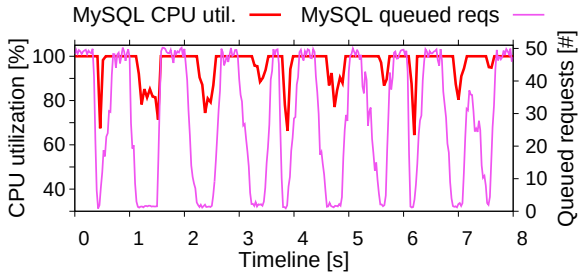
3.2 Throughput Loss Due to Rapidly Alternating Bottlenecks

Rapidly alternating bottlenecks are a kind of multi-bottleneck [33] that arises in n-tier systems due to the inter-dependencies among the tiers. Multi-bottlenecks are difficult to diagnose, since the overall throughput becomes limited, but there is no identifiable single saturated resource in the system. Although systematic methods to find multi-bottlenecks in general remain an active research challenge, we have found concrete examples of rapidly alternating bottlenecks.

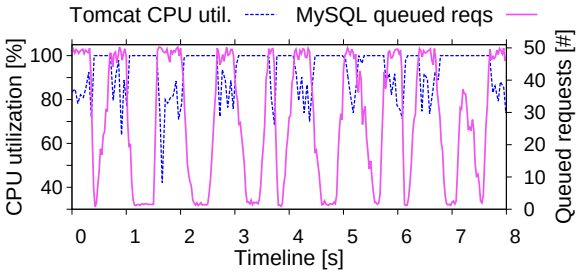
Figure 6 compares the overall RUBBoS throughput between the FullSpeed (DVFS off) and the DVFS-On cases. The two curves overlap (despite response time differences analyzed earlier) from 1,000 up to 9,000 workload. From 10,000, the throughput of FullSpeed starts to visibly outpace DVFS-On. The throughput advantage of FullSpeed reaches about 20% at workload 14,000 and stabilizes. This large difference would not have appeared, had the DVFS control algorithm been able to maintain the CPU clock rate at the maximum (P0 state). The appearance of rapidly alternating bottlenecks



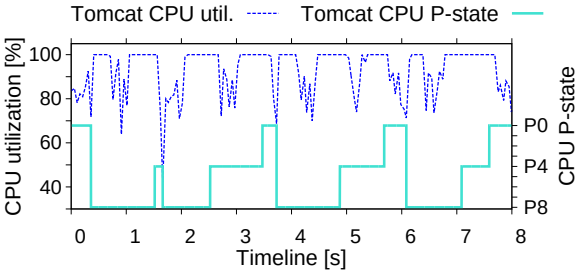
(a) CPU saturation alternatives rapidly between Tomcat and MySQL (the pearson correlation coefficient is -0.97), which indicates an alternating bottleneck in the system.



(b) Short-term saturations of CPU in MySQL (caused by "wrong" P-state) make requests queued in MySQL.



(c) Once queued requests in MySQL reach a threshold, requests are pushed back to Tomcat, which decelerates Tomcat's request processing thus the Tomcat CPU util. decreases.



(d) Low CPU utilization leads to P8 state in Tomcat, which causes more short-term saturations in Tomcat.

Figure 7: Simulation analysis of the rapidly alternating bottleneck in the system at workload 14,000. Figure 7(b) 7(c), and 7(d) show that due to the dependency between Tomcat and MySQL through soft resources, DVFS can lead to the rapid alternation of CPU saturation between Tomcat and MySQL as shown in Figure 7(a).

are suggested by this fact and the small notch in peaks 1 and 4 of Figure 5(c), showing Tomcat queue increasing while MySQL queue decreasing.

Our simulation data shows that rapidly alternating bottlenecks develop between MySQL and Tomcat at high utilization levels, with Figure 7 showing a representative scenario at 14,000 workload. The graph shows a 8-second snippet of a benchmark run similar to previous graphs 3, 4, and 5. We can see that effectively one of the servers is saturated (at 100% CPU utilization) all the time, although the other one is not. The Pearson correlation coefficient between these two metrics is -0.97, which is a strong negative correlation. Since one of the mutually-dependent servers is bottlenecked at all times, the overall system throughput cannot increase although no single system resource is fully saturated.

Although Figure 7(a) is an interesting confirmation of multi-bottleneck phenomenon, the multi part is less important than the rapid part in the current context. For the DVFS study, the important problem is the rapid alternation. Figure 7(b) shows that MySQL CPU utilization is nearly fully utilized (at the top), but the MySQL queue (the thin line) grows and shrinks rapidly between maximum (50 in these experiments) and minimum (empty). Between markers 0 and 7 (7 seconds elapsed time), the queue goes from full to empty and back a total of 9 times. As explained in the previous section, the full MySQL queue causes Tomcat to queue requests due to push-back. The Tomcat queue growth is omitted here since it is similar to Figure 5(c). As a result of push-back from MySQL, the Tomcat CPU utilization decreases to less than full utilization, when MySQL is the bottleneck. This rapid alternation now causes Tomcat utilization to oscillate between full and 60% (Figure 7(c)). This oscillation in CPU utilization is sufficient to cause the DVFS control to switch the CPU states between P0, P4, and P8. More seriously, the oscillations cause the DVFS control to prefer low clock rate (P8), as shown in Figure 7(d). The low clock rate causes a transient saturation in Tomcat (Figure 7(a)), which slows down the sending of requests to MySQL, making MySQL less than fully utilized. These alternating bottlenecks (between Tomcat and MySQL) happen at a frequency of more than once a second, which coincides with both the RUBBoS workload generation cycle and DVFS adjustment period, making the problem persistent and often amplified.

We emphasize that the coincidence of workload generation cycle and DVFS adjustment period is not an artifact of any intentional tweaking of experimental configuration settings. We used the default RUBBoS workload generator settings, which were chosen more than a decade ago by application designers to model representative n-tier applications. Similarly, we used the Dell-provided BIOS setting of DVFS adjustment policy, which was carefully chosen by hardware designers for different reasons. Nevertheless, the coincidence led us to the following question: *Can we find an upper bound for DVFS impact on n-tier application performance?* This is an important question since DVFS control (as most of the hardware) is usually considered a transparent layer, not to be taken into consideration when measuring application level performance. Determining the amplitude of DVFS impact can improve significantly our understanding of the uncertainties in application benchmark performance measurements.

4 Study on An Upper Bound of DVFS Control Impact

Before we start the study of upper bound, we should observe that the lower bound of DVFS control impact is zero, when DVFS is turned off (the FullSpeed case). This does not mean that interesting phenomena (e.g., push-back and multi-bottlenecks) will disappear entirely. Our study shows that interesting phenomena arise because of DVFS control, but are not caused solely by DVFS. Further study of those phenomena is beyond the scope of this paper.

4.1 Algorithmic Calculation of DVFS Adjustment Errors

The study of upper bound is based on a DVFS control model using the detailed simulator (Appendix B). We use a terminology loosely based on control systems to refer to the components of DVFS control. For example, we use the term "error" to denote the difference between the predicted workload/utilization and the actual work done by the DVFS-controlled CPU. For each control period (also called a time_window):

1. $Error$ = $|Work\_done - Required\_CPU|$
2. $Work\_done$ = util * clock_rate * time_window
3. $Required\_CPU$ = $ready\_queue\_length$ * task_requirement
4. $ready\_queue\_length$ = $remaining\_ready\_req$ + $incoming\_req$

The above definitions contain approximations due to the non-zero time_window. For example, we need to take into account the arriving requests when calculating the $ready\_queue\_length$. Still, if the DVFS control had perfect knowledge of the workload, it would be able to reduce the $Error$ to zero by providing sufficient $Required\_CPU$ to satisfy $Work\_done$ for each time_window. In practice, DVFS can approach this ideal situation under two assumptions: (1) $Required\_CPU$ varies very little (steady workload), so the past is a good estimate of present and future; (2) $Required\_CPU$ is low (low workload) and the $Error$ remains low, too.

In our study, web-facing applications violate assumption (1), and data centers with high utilization requirements are incompatible with assumption (2). The objective of this section is to find an upper bound for $Error$ when running bursty workloads at high utilization levels. Intuitively, $Error$ is minimized if the CPU clock rate matches workload: both should be high (or low) simultaneously. Specifically, if a high workload encounters low clock rate, $Work\_done$ would be too low and $Error$ grows. In the simulator, we calculate $Error$ in each time_window by the following steps:

- Measure the CPU required for each kind of task. For each task being launched, add the $task\_requirement$ to $Total\_CPU\_required$
- For every task completed, add $task\_requirement$ to $Work\_done$.
- For every time_window, subtract the $Work\_done$ from $Total\_CPU\_required$
- $Total\_CPU\_required$ is the $Required\_CPU$ for the near future (including the current time_window).

This algorithm reflects the observation that $Error$ is accumulated when workload/clock-rate mismatches occur. The objective of our upper bound calculation then becomes a problem of maximizing the mismatches, which is the subject of Section 4.2. Conversely, the negative impact of DVFS can be decreased by reducing the mismatches, which is the subject of Section 5.

4.2 Anti-Synchrony between Workload and DVFS

Our study to find the maximum $Error$ for a given DVFS control escapes the classic assumptions of control systems. For example, instead of a fixed input workload model for which a control system can be designed with predictable maximum error, the workload can vary arbitrarily. Consequently, we use the detailed simulator to

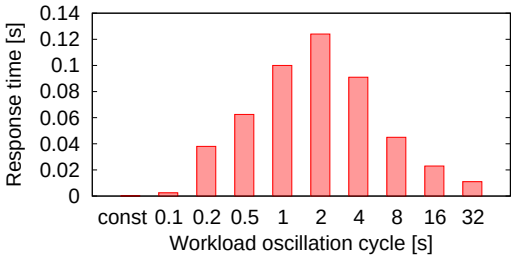


Figure 8: Simulation analysis of response time comparison among different workload oscillation cycles.

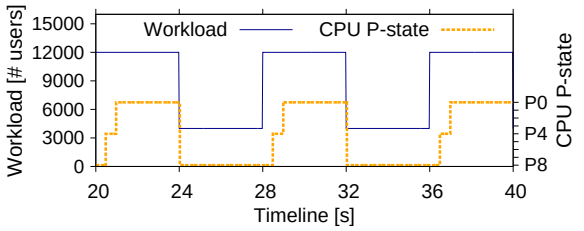
find the largest $Error$, which happens when the workload burst cycle and DVFS adjustment period are anti-synchronous (similar in length but out of phase).

The first step in our study is the implementation of an extended RUBBoS workload generator with fine-grain control over the period and intensity of bursty workloads [27]. The result is a bursty workload generator with two modes (high and low, e.g., 12,000 and 4,000 clients), plus controllable cycles between these two modes to simulate different burstiness levels in n-tier applications. A cycle consists of the generator running in one mode followed by a switch and running in the other mode. The generator then switches back to the original mode at the beginning of the next cycle. High burstiness is implemented as a short cycle and a steady workload can be implemented as an infinite cycle. In our experiments, we maintain the same average workload intensity level, e.g., 8,000 clients obtained by dividing the cycle evenly into half high (12,000 clients) and half low (4,000 clients). The cycle length is varied to generate different burstiness levels while maintaining the same workload intensity.

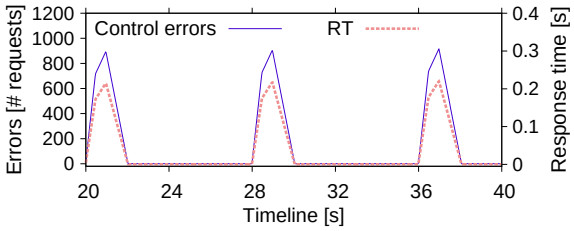
The high/low workload generator implementation enables a sensitivity study of response time as a function of workload burstiness. Figure 8 shows the average response time with DVFS-On with the same average workload intensity (8,000 clients) and high/low of 12,000/4,000 clients, but different workload burstiness cycles. The simulation data shows the system response time degraded the most from cycle time of 0.5sec to 4sec. From the control point of view, it is straightforward to explain the good response time for long cycles. The DVFS adjustments at 0.5sec intervals seem to be sufficient in handling the workload bursts that are longer than 4sec. Conversely, the very frequent cycles are also best handled by relatively slow control adjustments, since their behavior becomes increasingly similar to the average behavior at very high frequencies.

These conceptual explanations are confirmed by simulation data. Figure 9(a) shows the workload with oscillation cycle 8sec and the CPU clock rate (in P-state). When the workload intensity switches between high and low, the server takes two adjustment periods (about 1sec) to change between the lowest rate (P8) and the highest rate (P0). The adaptation time is relatively short and the system has a good match (high workload with high clock rate, and low workload with low clock rate) most of the time (7 out of 8sec). The small error due to adaptation delay is shown in Figure 9(b). At the beginning of each cycle (e.g., the origin of the graph on the left), workload goes up to 12,000, causing a transient saturation of CPU and increase in response time. The temporary $Error$ is shown in the graph and strongly correlated with the server response time increase. The saturation pushes DVFS to increase CPU clock rate, fixing the problem for the remainder $7/8$ of the cycle.

On the other end of the spectrum, when the workload oscillation cycle is much smaller, the simulation confirms that the DVFS control period of 0.5sec works quite well, too. Figure 11(a) shows workload cycles and response time when the workload oscillates at 200ms cycles. The very fast workload oscillations actually reduce the $Error$ since the workload cycles back to a "correct" rate before the CPU reacts. Each DVFS adjustment (e.g.,

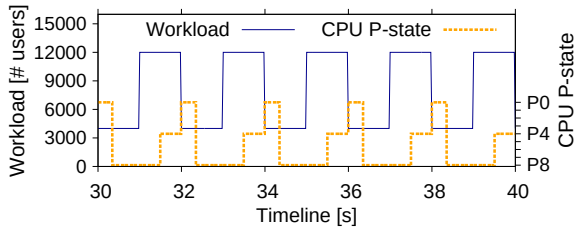


(a) Workload with oscillation cycle 8s and server CPU P-state. The adaptation delay of CPU clock rate happens when the workload switches from low to high.

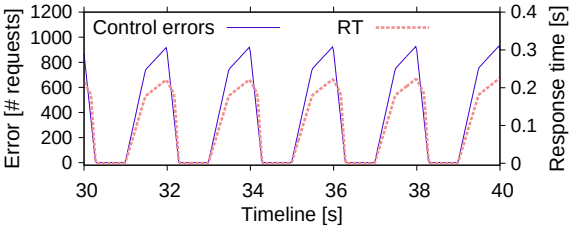


(b) Strong correlation between response time with control errors. The adaptation delay of CPU P-state (see Figure 9(a)) causes large control errors and also high peaks of the response time in the server.

Figure 9: DVFS works well when the workload oscillation cycle is much longer than the DVFS adaptation period. Figure 9(a) and 9(b) show that though the control error caused by each adaptation delay of CPU clock rate is high, the frequency of adaptions is low.

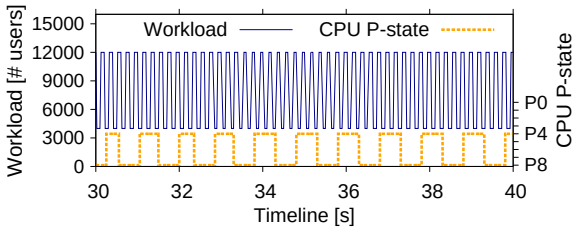


(a) Workload with oscillation cycle 2s and server CPU P-state. The adaptation delay of P-state covers the entire high workload period.

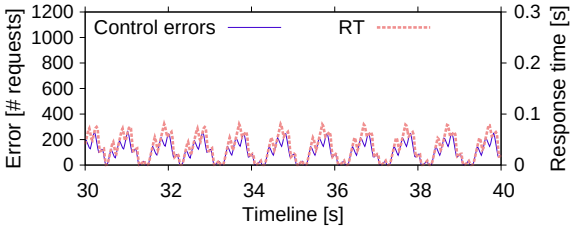


(b) Frequent high peaks of control errors and server response time due to the adaptation delay of CPU P-state (see Figure 9(b))

Figure 10: DVFS causes the largest errors when the workload oscillation cycle is close to the DVFS adaptation period. Figure 10(a) shows that the server CPU always stays in the "wrong" P-state when the workload is high, thus the overall control errors reach the maximum as shown in Figure 9(b).



(a) Workload with oscillation cycle 200ms and server CPU P-state. The adaptation of CPU P-state is less sensitive to the rapid workload oscillation.



(b) No high peaks of control errors compared to Figure 9(b) and 10(b), which leads to more stable response time.

Figure 11: DVFS works well when the workload oscillation cycle is much shorter than the DVFS adaptation period. Figure 11(a) shows that the DVFS controller is more robust to the rapidly changing workload, causing small errors and response time as shown in Figure 11(b).

high clock rate) can match several periods of high workload rate before it switches to a lower clock rate. This is reflected in the error graph (Figure 11(b)).

For control system experts, it is perhaps unsurprising that the largest $Errors$ appear when the workload oscillation cycles have lengths similar to the DVFS control cycle. This happens in the middle of spectrum. Figure 10(b) shows the response time and $Error$ calculations for workload cycles at 2sec (the highest impact in Figure 8). The mismatch between workload burst cycles and DVFS adjustment periods is called anti-synchrony in analogy to anti-synchronous oscillatory systems. A concrete example shown in Figure 10(a) is at timeline 31 (and 33) when the workload cycle goes high just as the CPU clock rate has been slowed down.

We observe that by upper bound we do not mean the design of an adversarial workload that would achieve the theoretical maximum $Error$ (i.e., exact anti-synchronous cycles against the DVFS adjustment periods). In this study, we aim to find the upper bound for bursty workloads that are reasonably regular and likely to happen in real world applications. We believe that alternative bursty workload models would yield substantially similar experimental results due to the necessary matching with the same DVFS adjustment periods.

We validated our simulation results using the real experiments in our testbed. We used the same bursty workload generator as we used in simulation to generate the high/low bursty workload with different oscillation cycles and kept the system configuration the same as before (see Figure 19(c)). Figure 12 shows the experimental results of the system response time under different workload oscillation cycles while maintaining the same average workload intensity (8,000 clients). This figure shows that the system response time reaches the highest when the workload oscillation cycle is 2sec and decreases with either shorter or longer oscillation cycles. Such observations match well with our simulations results above (see Figure 8).

5 Study of Two Solutions to Disrupt Anti-Synchrony

5.1 Solution 1: Increasing DVFS Adjustment Frequency

Since the performance problem caused by DVFS is due to anti-synchrony between workload burst cycles and

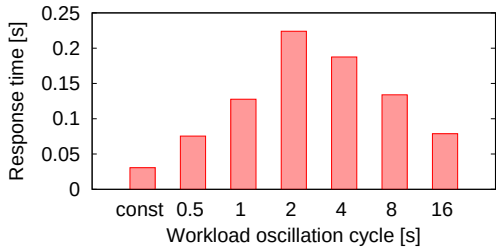


Figure 12: Experimental validation of the simulation analysis on workload oscillation cycles

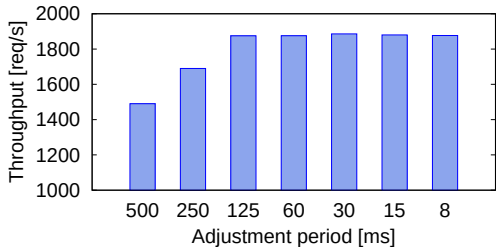


Figure 13: Adjustment period vs. system throughput

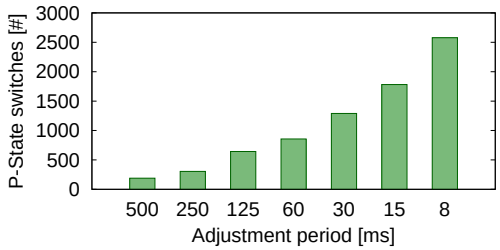


Figure 14: Small adjustment periods lead to increased P-state swithes.

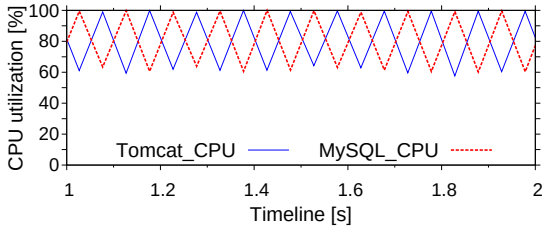


Figure 15: A rapidly alternating bottleneck occurs even when the adjustment period is as small as 50ms.

DVFS adjustment period, an obvious solution is to increase the frequency of DVFS adjustments to disrupt the anti-synchrony. Figure 13 shows the maximum throughput the system can achieve for several DVFS adjustment periods. We start from the 500ms Dell BIOS default period as the baseline. The simulation results show that the system throughput improves significantly when adjustment period is reduced from 500ms to 125ms (four times more frequently). The explanation is a reduction of rapidly alternating bottlenecks since the servers are getting more CPU more quickly when they need it. At 125ms DVFS adjustment period the Tomcat gets sufficient CPU to avoid rapidly alternating bottlenecks altogether and reaching the maximum throughput, unaffected by more frequent adjustments of CPU clock rate.

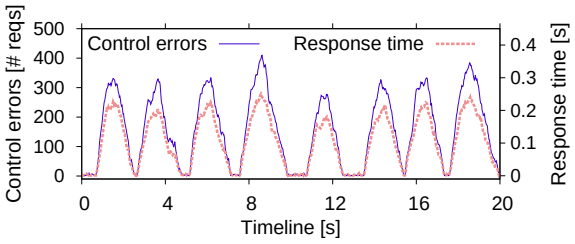
Increasing the DVFS adjustment frequency is an obvious solution that also has some obvious problems. First, DVFS adjustments have significant costs in terms of energy consumption and instruction execution delay [37]. This is one of the reasons the DVFS adjustments have been set to a relatively long period (500ms) in the first place. The increasing CPU clock adjustment overhead is illustrated in Figure 14 for adjustment periods from 500ms to 8ms.

The second problem is that increasing DVFS adjustment frequency only disrupts the anti-synchrony associated with the workload of similar burst cycle length. As workload burst cycle length decreases (a likely scenario for faster CPUs and more clients), anti-synchrony can happen to shorter DVFS adjustment periods. This is illustrated by Figure 15, which shows CPU utilization of Tomcat and MySQL when the workload burst cycle is 50ms. The simulation data shows that for cycles as short as 50ms, anti-synchrony can still happen due to rapidly alternating bottlenecks. This observation led to the design of Solution 2 described below.

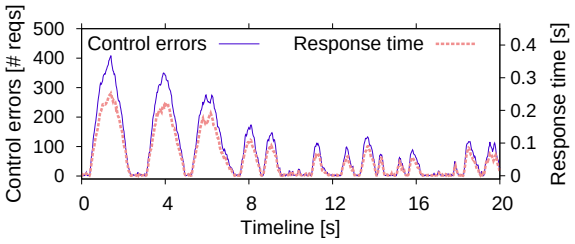
5.2 Solution 2: Workload-Sensitive Adaptive Control

Since workload burst cycles may vary, any fixed-length DVFS adjustment period will remain vulnerable to anti-synchrony. Consequently, a workload-sensitive adjustment method seems a better solution. Direct observations of workload intensity is challenging in n-tier systems, since there are significant and mutually-dependent variations at each tier. Instead, we choose to add a second level adaptive control on the DVFS itself.

Our design considers a classic (fixed-period) DVFS as the system to be controlled. The main observable of interest is the CPU P-state switch. We consider the change from P8 (slowest clock rate) to P0 (fastest clock rate) as an indication of confirmed workload burst. More generally, the interval between consecutive such switches can be considered a reasonable estimate of the workload burst cycle. For example, Figure 9(a) and Figure 10(a) shows that the CPU switches from P8 to P0 (P4 in between) frequently and the period of workload burst is every 8 seconds and 2 seconds, respectively. The second level controller uses the observed workload burst



(a) Server response time and control errors of the original controller



(b) Server response time and control errors of the adaptive controller

Figure 17: Comparison between the adaptive DVFS controller and the original DVFS controller. Figure 17(b) shows that the latter case is more effective to reduce the control errors and stablize the server response time.

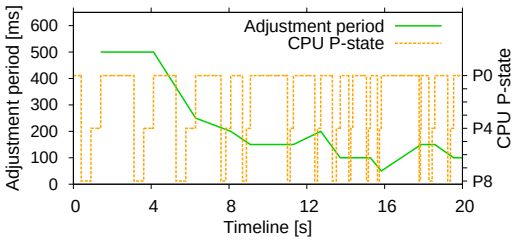


Figure 16: Adjustment period variation in the adaptive controller when server is at workload 11,000.

cycle to predict the onset of next workload burst. Although burst cycles may change over time, we assume that burst cycles can be estimated by a linear function in the neighborhood of an operating point. We adopt a simple moving-average (MA) model [38] to predict the workload burst cycle as shown in the Equestion 1:

$$B_{i+1} = \frac{1}{k} \sum_{j=i-k+1}^{i} B_j \quad (1)$$

The MA model assumes the short-term dependencies between successive burst cycles. B_{i+1} denotes the length of the next workload burst cycle. k refers to the previous k burst cycles remembered: the larger the k, the more past P-state switches will be taken into account. We estimated the model offline using least-squares based on methods in the Matlab System ID Toolbox to fit the input-output data collected from simulation. The model is evaluated using the r^2 metrics defined in Matlab as a goodness-of-fit measure. In general, the r^2 value indicates the percentage of variation in the output captured by the model. In our case, the r^2 is 0.912, which indicates a good fit of the model.

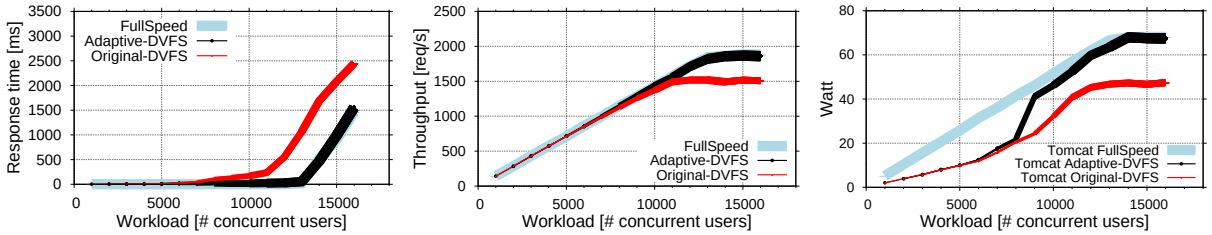
Once we obtain the predicted workload burst cycle, we adjust the length of adjustment period accordingly to reduce the DVFS negative impact. For example, we can set the length of adjustment period to be proportionally smaller than the workload burst cycle as shown in the following equation:

$$T_i = \begin{cases} T_{lb} & \text{if } (B_i/n) < T_{lb} \\ T_{ub} & \text{if } (B_i/n) > T_{ub} \\ B_i/n & \text{otherwise} \end{cases} \quad (2)$$

Both T_{lb} and T_{ub} are thresholds that prevent the new adjustment period to be either too small (significant P-state switching overhead) or too large (significant performance degradation caused by prediction error). n sets the distance between the workload burst cycle and the new adaptation period (which is 2 adjustment periods in our case). As shown in Figure 8, the larger distance is set between these two metrics, the better performance of the system. A setting of 4-8 times larger would typically leads to fairly good results. In our evaluation, we set T_{lb}, T_{ub}, and n to be 50ms, 500ms, and 10. Thus, the burst cycle is always 5 times larger than the adaptation period in our adaptive controller.

Figure 17 shows the effectiveness of the adaptive DVFS controller compared to the original controller with fixed adjustment periods on a server at workload 11,000. Figure 16 shows that the length of the adaptation period varies over time due to the changes of the workload bursty cycle in the adaptive controller case. Figure 17(b) shows the $Error$ caused by anti-synchrony between DVFS adjustment period and workload burst cycle. The server response time (and $Error$) become smaller and more stable over time due to the workload-sensitive adaptive changes of the DVFS adjustment period. Figure 17(a) shows high $Error$ and response time of the original DVFS controller compared to our two-level adaptive DVFS controller.

Figure 18 further shows the effectiveness of the adaptive DVFS controller in n-tier applications. We apply the adaptive DVFS controller to the 4-tier system as shown in our motivation case (see Section 2) in simulation. Figure 18(a) and 18(b) show that the adaptive DVFS controller achieves the similar response time and throughput to the FullSpeed case on the entire workload



(a) System response time comparison (b) System throughput comparison (c) Tomcat power consumption comparison

Figure 18: Comparison among three different DVFS policies in the context of n-tier applications. The adaptive DVFS controller achieves better balance between performance and power usage than the other two.

range, which suggests that the adaptive DVFS controller doesn't cause significant performance loss. Since in simulation we are unable to directly measure the power consumption of each server, we adopt a simple model [16] to estimate the CPU power consumption as a function of CPU P-state distribution and CPU utilization:

$$P = C * R_{active} * V^2 * f + P_{static} \quad (3)$$

Here, C is a constant related to the capacitance of transistor gates, R_{active} is the CPU utilization, V is the operating voltage, and f is the CPU frequency. From the specification of the Xeon CPU model in our experiments we know the CPU frequency and voltage at each P-state [2]. To simplify the analysis, we denote C to be 1 and P_{static} to be 0 since we only consider the CPU dynamic power consumption. Thus given the measured CPU utilization and P-state of each server in simulation, we are able to estimate the power consumption of each server at each workload.

Figure 18(c) shows the adaptive DVFS controller is able to save the similar amount of power as the original DVFS controller does before workload 8,000 while it gradually merges to the FullSpeed case as workload continue to increase. The width of each of the three lines is proportional to the average CPU clock rate. Overall Figure 18 shows that the adaptive DVFS controller achieves better balance between performance and power consumption than the other two policies. Evaluating our solution in real implementation would naturally be the future work of this paper.

6 Related Work

Reducing power consumption has become a major research topic due to the increasing cost of power consumption in current data centers [4, 9, 19, 29].

[2]E.g., 1.197 GHz/0.750V at P8-state, 2.26GHz/1.350V at P0-state

Balancing power/performance using DVFS has been studied extensively at the microarchitectural level [6, 12, 21, 24] ever since Weiser et al.[35] proposed the idea of modulating the CPU clock rate based on CPU load. For example, Brooks et al.[6] proposed mechanisms to enforce thermal thresholds on processors while meet performance requirements. Power budgeting of SMP systems to minimize performance loss has also been implemented via DVFS [21, 24]. These previous works achieve good balance between power and performance in their specific domain, but not n-tier systems.

Workload characteristics have been considered as important contributors to the effectiveness of DVFS. Freeh et al. [14] and Choi et al. [10] have investigated how to efficiently choose a proper CPU mode based on the workload characteristics to minimize both the energy usage and performance loss. This premise further leads to hardware-based techniques [23, 28] and OS-level solutions [18, 30, 26] that set processor modes based on predicted workload characteristics. Some previous research on the effectiveness of DVFS uses micro-benchmark workloads [30, 36], especially the SPEC CPU benchmarks, which are not an accurate representative of real-world use. Consequently, the findings from those micro-benchmarks must be interpreted with caution [22].

Analytical models to estimate the impact of DVFS on system performance have been formulated in previous research. Curtis-Maury et al. [11] present an online performance model aiming for the integration of DVFS and dynamic concurrency throttling in performance-constrained systems. Weissel et al [36] and Miftakhutdinov et al. [28] use hardware performance counters to parameterize models on which to aid power management decisions and DVFS performance prediction. Several other papers consider the end-to-end performance impact when performing DVFS for multi-tier applications [8, 17]. Though these models can get good prediction accuracy when the system is lightly loaded, their accuracy is unclear for the system in high utilization.

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
Operating system	RHEL 6.2 (kernel 2.6.32)
Hypervisor	VMware ESXi v5.0
System monitor	esxtop 5.0, Sysstat 10.0.0
Power monitor	WATTS UP? .Net

(a) Software setup

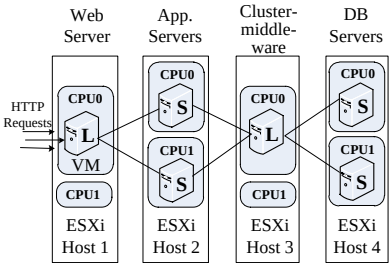
ESXi Host Configuration

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 19: Details of the experimental setup.

7 Conclusions

DVFS is a widely deployed power-saving technology that works well for stable workloads and/or systems with low utilization levels. However, web-facing applications running in data centers have bursty workloads and high utilization levels. Through measurements of n-tier application benchmark (RUBBoS) and a simulation study, we quantified the impact of DVFS on n-tier application performance. Experimental and simulation results show that significant degradations happen on response time and throughput because of two problems: (1) large response time fluctuations due to push-back wave queuing in n-tier systems, and (2) throughput loss due to rapidly alternating bottlenecks between the database and application servers.

Through a simulation study, we determined an upper bound of DVFS impact for representative bursty workloads by studying the anti-synchrony between workload burst cycles and DVFS adjustment periods. Using the detailed simulator, we also studied two OS-level solutions to reduce the DVFS impact by disrupting the anti-synchrony: (1) frequent DVFS adjustments, and (2) workload-sensitive adaptive control of DVFS. These solutions may have significant practical impact on the deployment of web-facing applications in data centers by preserving DVFS power savings, reducing application performance unpredictability, and increasing server utilization levels.

8 Acknowledgement

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

A Experimental Environment

We adopt the RUBBoS standard n-tier benchmark, based on bulletin board applications such as Slashdot [1]. 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 [7]) system. The workload consists of 24 different interactions. The benchmark includes two kinds of workload modes: browse-only and read/write mixes. We use browse-only CPU intensive 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. Since the request rate follows a Poisson distribution, it naturally fluctuates when observed at fine-granularity. In reality, the actual workloads for n-tier web-facing systems present more fluctuations and are more unpredictable [15, 27].

We run the RUBBoS benchmark on our virtualized testbed. Figure 19 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 19(c) shows a sample 1/2/1/2 topol-

Table 1: Notations for modeling.

Parameter	Explanation	Value
TW	sampling time window	50ms
u_i	current CPU util in ith window	
u_{ref}	reference CPU util	80%
TP_{up}	scale-up threshold ($> u_{ref}$)	0
TP_{down}	scale-down threshold ($< u_{ref}$)	15%
T_u	consecutive times needed to trigger the scale-up action	10
T_d	consecutive times needed to trigger the scale-down action	1

ogy. Each server runs on top of one VM. 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 reduce the interference among VMs. We use two power management policies supported by Dell BIOS settings: Dell Active Power Controller and Maximum Performance as DVFS-On and FullSpeed. Note we always deploy Apache and C-JDBC in type "L" VMs and let them run at FullSpeed mode since we want to avoid bottlenecks in load-balance tiers. Thus only the application server tier and the database server tier run at the DVFS-On mode in the DVFS-On case. We use a WattsUp [2] power meter to log power draw each second for each ESXi host.

B Simulation Setup

We use simulation to reveal the mechanism of how DVFS degrades n-tier system performance due to the rich information we can get during the runtime simulation. Our simulation environment has two parts. The first part is a queueing network which simulates an n-tier system with arbitrary configurations. For this purpose we use an open source queueing network simulator JMT [5] which simulates our real experimental configurations. The second part is a service rate controller for each node of the queueing network. A service rate controller simulates a DVFS controller of a real server in our experiments and it changes the service rate of the controlled node based on the real-time workload.

Figure 2 shows our DVFS CPU clock rate control infrastructure in our simulation. The central part is the DVFS controller, which takes the measured CPU utilization of a server and the target reference CPU utilization as input signals; once input signals meet certain threshold, the Actuator specifies a new CPU clock rate for the controlled CPU.

The DVFS controller is essentially a discrete device

Table 2: Simulator validation through real experiments.

WL	Simulation TP		Experimental TP	
	FullSpeed (req/s)	DVFS-On (req/s)	FullSpeed (req/s)	DVFS-On (req/s)
1K	143	142	144	143
4K	565	564	585	582
8K	1120	1116	1167	1130
1K	1684	1524	1704	1462
14K	1865	1517	1883	1518
16K	1886	1530	1907	1503

Table 3: Maximum achievable throughput comparison in different configurations through simulation.

Config.	FullSpeed (req/s)	DVFS-On (req/s)	TP Loss (req/s)	Percent (%)
1/1/1	790	642	148	18.7
1/2/1/2	1886	1530	356	18.9
1/4/1/4	3753	3030	723	19.3
2/8/2/8	7613	5843	1769	23.2
4/16/4/16	14780	9916	4864	32.9
8/32/8/32	29370	20940	8430	28.7

with some adjustable time delays to respond to the incoming signals. Equation 4 shows a model to capture the aggregate effect of its adaptive operations.

$$f_{i+1} = f_i + Step \times I(u_i - u_{ref}) \tag{4}$$

Where

$$I(u_i - u_{ref}) = \begin{cases} 1 & \text{if } (u_i - u_{ref}) > TH_{up} \text{ for } T_u \\ -1 & \text{if } (u_i - u_{ref}) < \text{-}TH_{down} \text{ for } T_d \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Intuitively, the above equation means a CPU clock rate increment will occur if the signal is larger than the scale-up threshold for a consecutive T_u times (in sampling time windows), or decrement if the signal is smaller than the scaling-down threshold for a consecutive T_d times. Otherwise, the frequency stays unchanged. In this design, the discrete *step* was chosen to be relatively large to simplify the analysis, for instance, we only use P8 (slowest), P4-, and P0-state (fastest) of CPU in our simulation. The parameters of our model is in Table 1, which are derived from the real experimental obervations. We validate our model through real expeirmental results as shown in Table 2.

Table 3 shows the simulation results of system configurations ranging from 3 to 80 servers. This table shows that the maximum achievable throughput loss ranges from 18% to 32.9% under different confgiurations.

References

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

[2] *Watts up? .NET Power Meter*. "http://wattsupmeters.com".

[3] S. Adler. The slashdot effect: an analysis of three internet publications. *Linux Gazette*, 38:2, 1999.

[4] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A fast array of wimpy nodes. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP 2009)*, pages 1–14, 2009.

[5] M. Bertoli, G. Casale, and G. Serazzi. JMT: performance engineering tools for system modeling. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):10–15, 2009.

[6] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th IEEE International Symposium on High-Performance Computer Architecture (HPCA 2001)*, pages 171–182, 2001.

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

[8] S. Chen, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and W. H. Sanders. Blackbox prediction of the impact of dvfs on end-to-end performance of multitier systems. *ACM SIGMETRICS Performance Evaluation Review*, 37(4):59–63, 2010.

[9] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys 2012)*, pages 43–56, 2012.

[10] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *Proceedings of the 2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004)*, pages 4–9, 2004.

[11] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT 2008)*, pages 250–259, 2008.

[12] G. Dhiman and T. S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design (ISLPED 2007)*, pages 207–212, 2007.

[13] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for linux. In *Proceedings of the 5th USENIX Conference on Operating Systems Design and Implementation (OSDI'02)*, pages 105–116, 2002.

[14] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):835–848, 2007.

[15] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Trans. Comput. Syst.*, 30(4):14:1–14:26, Nov. 2012.

[16] V. Gupta, P. Brett, D. Koufaty, D. Reddy, S. Hahn, K. Schwan, and G. Srinivasa. The forgotten 'uncore': On the energy-efficiency of heterogeneous cores. In *Proceedings of the 2012 USENIX Annual Technical Conference*, 2012.

[17] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers*, 56(4):444–458, 2007.

[18] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)*, pages 359–370, 2006.

[19] M. Kazandjieva, B. Heller, O. Gnawali, P. Levis, and C. Kozyrakis. Green enterprise computing data: Assumptions and realities. In *Proceedings of the 3rd International Green Computing Conference (IGCC'12)*, pages 1–10, 2012.

[20] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture (HPCA 2008)*, pages 123–134, 2008.

[21] R. Kotla, S. Ghiasi, T. Keller, and F. Rawson. Scheduling processor voltage and frequency in server and cluster systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)*, 2005.

[22] E. Le Sueur and G. Heiser. Slow down or sleep, that is the question. In *Proceedings of the 2011 USENIX Annual Technical Conference*, 2011.

[23] H. Li, C.-Y. Cher, T. Vijaykumar, and K. Roy. Vsv: L2-miss-driven variable supply-voltage scaling for low power. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, pages 19–28, 2003.

[24] K. Ma, X. Li, M. Chen, and X. Wang. Scalable power control for many-core architectures running multi-threaded applications. In *Proceedings of the 38th ACM/IEEE International Symposium on Computer Architecture (ISCA 2011)*, pages 449–460, 2011.

[25] S. Malkowski, M. Hedwig, D. Jayasinghe, C. Pu, and D. Neumann. CloudXplor: a tool for configuration planning in clouds based on empirical data. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC 2010)*, pages 391–398, 2010.

[26] A. Merkel, J. Stoess, and F. Bellosa. Resource-conscious scheduling for energy efficiency on multicore processors. In *Proceedings of the 5th European conference on Computer systems (EuroSys 2010)*, pages 153–166, 2010.

[27] 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.

[28] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt. Predicting Performance Impact of DVFS for Realistic Memory Systems. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*, 2012.

[29] R. Nathuji and K. Schwan. VirtualPower: coordinated power management in virtualized enterprise systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*, pages 265–278, 2007.

[30] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for OS-level power management. In *Proceedings of the 4th ACM European conference on Computer systems (EuroSys 2009)*, pages 289–302, 2009.

[31] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Proceedings of the 35th ACM/IEEE International Symposium on Computer Architecture (ISCA 2008)*, pages 363–374, 2008.

[32] 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.

[33] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu. An experimental study of rapidly alternating bottlenecks in n-tier applications. In *Proceedings of the 6th International Conference on Cloud computing (Cloud 2013)*, 2013.

[34] 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 and Distributed Processing Symposium (IPDPS 2011)*, pages 1034–1045, 2011.

[35] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation (OSDI'94)*, pages 13–23, 1994.

[36] A. Weissel and F. Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2002)*, pages 238–246, 2002.

[37] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors. In *Proceedings of the 11th IEEE International Symposium on High-Performance Computer Architecture (HPCA 2005)*, pages 178–189, 2005.

[38] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu. Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach. In *Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS 2011)*, pages 571–580, 2011.