Evaluating Latency-Sensitive Applications' Performance Degradation in Datacenters with Restricted Power Budget

Song Wu*, Chuxiong Yan*†, Haibao Chen*‡, Hai Jin*, Wei Guo†, Zhen Wang†, Deqing Zou*
*Services Computing Technology and System Lab
Cluster and Grid Computing Lab
Huazhong University of Science and Technology, Wuhan, China
†Tencent Inc, Shenzhen, China
‡Corresponding author, chenhaibao@hust.edu.cn

***Abstract*—For datacenters with limited power supply, restricting the servers' power budget (i.e., the maximal power provided to servers) is an efficient approach to increase the server density (the server quantity per rack), which can effectively improve the cost-effectiveness of the datacenters. However, this approach may also affect the performance of applications in servers. Hence, the prerequisite of adopting the approach in datacenters is to precisely evaluate the application performance degradation caused by restricting the servers' power budget. Unfortunately, existing evaluation methods are inaccurate because they are either improper or coarse-grained, especially for the latency-sensitive applications widely deployed in datacenters.**

In this paper, we analyze the reasons why state-of-the-art methods are not appropriate for evaluating the performance degradation of latency-sensitive applications in case of power restriction, and we propose a new evaluation method which can provide a fine-grained way to precisely describe and evaluate such degradation. We verify our proposed method by a real-world application and the traces from Tencent's datecenter with 25328 servers. The experimental results show that our method is much more accurate compared with the state of the art, and we can significantly increase datacenter efficiency by saving servers' power budget while maintaining the applications' performance degradation within controllable and acceptable range.

I. Introduction

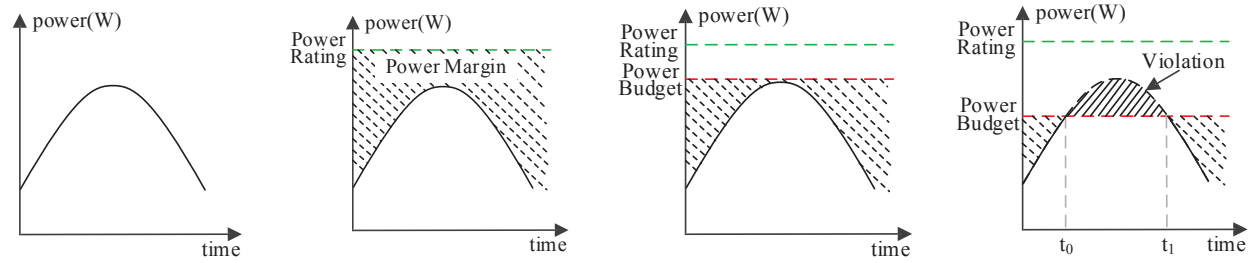
Datacenters have become the first option for *ISPs* (Internet Service Providers) to deploy their Internet services, especially leading ones like Google, Amazon, and Tencent [1]. Studies [2, 3] have found that the power used by servers is always far below their power rating (i.e., servers' maximum power). Assuming the power demand of a server over time is as shown in Fig.1(a), if we set its power budget at the power rating as shown in Fig.1(b), there will be a huge power margin in the area shaded by dashed line. Specifically, the power margin is the part of the power budget that is not consumed by the servers, which directly decreases the datacenter efficiency.

To eliminate the power margin in servers, the server consolidation approach [4–6] unites as many applications as possible into fewer servers to make full use of the power budget. However, there are some problems with this method in practical datacenters [7]. First, due to the higher density of applications in a server, a single point of failure will affect more applications. Meanwhile, server consolidation is typically based on virtualization, which could bring some overheads and unfair contentions among applications. Finally, there is inevitable cost of application migration when consolidating applications.

Another approach to eliminate the power margin is to restrict the servers' power budget [2, 8–10], that is, to set a lower power budget by using power capping technology. Since the power supply of a rack is typically fixed, restricting the power budget will allow more servers to be deployed in a rack.

However, restricting the power budget may result in performance degradation when the power demand of a server exceeds its budget. Taking Fig. 1 as an example, when we restrict the power budget below the power rating as shown in Fig. 1(c), there is no performance degradation, and a significant part of the power margin is alleviated compared with Fig. 1(b). When we further restrict the power budget to alleviate more of the power margin as shown in Fig. 1(d), power budget violation occurs during the time from t_0 to t_1, which will degrade applications' performances. Hence, there exists a tradeoff between the applications' performances and the server density of the datacenter. It is a challenge to appropriately restrict the power budget to increase the number of servers per rack while keeping applications' performance degradation in a tolerable range. Targeting this challenge, in this paper we focus on how to evaluate applications' performance degradation under a restricted power budget for servers, which provides guidelines for power budget management in datacenters.

Latency-sensitive applications (e.g., Web-based applications, online games, streaming media) are prevalent in datacenters [11]. Unfortunately, on one hand, existing evaluation approaches [2, 8, 9] for applications' performance degradation under a restricted power budget are mainly designed for batch applications and are not suitable for latency-sensitive ones due to their defective or coarse-grained methods; on the other hand, because of concerns about affecting users'



(a) Server's power demand (b) Case 1 (c) Case 2 (d) Case 3

Figure 1. The power margin (shown in shaded area) varies with different power budget settings. Typically, the lower the power budget assigned to the server, the more power margin can be eliminated. When the power budget is set to below power rating, it may cause a power budget violation.

experience, datacenters are very cautious in dealing with the performance degradation of latency-sensitive applications [2]. In this paper, we propose a fine-grained evaluation approach based on calculus, which is designed for the performance degradation evaluation of latency-sensitive applications.

The main contributions of this paper are two-fold:

- We elaborately investigate the state-of-the-art evaluation methods and analyze the reasons why they are inappropriate for latency-sensitive applications. Then we propose an evaluation method based on calculus, which introduces two important metrics, *PDW* (percentage of delayed workload) and *AD* (average delay), to precisely evaluate what percentage of the workload (e.g., request-response operations) is delayed and how long this workload is delayed, respectively.
- We verify our proposed method by a real-world latency-sensitive application and a trace of Tencent's production datacenter with more than 25000 servers, and provide guidelines for power budget restriction in datacenters which can help administrators decide the proper power budget of each server so as to increase datacenter efficiency while maintaining the application performance degradation within controllable and acceptable range. The experiment results present that our proposed method is much more accurate than the state-of-art, and the analysis based on the Tencent's trace shows that at least one third of the power budget of the servers in Tencent's production datacenter can be saved.

II. Motivation and Our Basic Idea

A. Background

The power supply is a common restriction in datacenter construction [12]. However, due to the over-provisioning of power budgets, a power margin exists widely in datacenters [2], which probably results in wastage of the power budget. Google studies the actual power usage of its servers and finds that the power usage is about 55% of the total power supply (i.e., the power margin is about 45%) [2]. An intuitive way to eliminate the power margin is by restricting the power budget by power capping, that is, setting a lower power threshold of a server as its power budget [2, 12, 13]. Since there is a strong correlation between servers' power consumption and CPU utilization, capping the CPU utilization has almost the same effect as resizing the power budget [14]. Therefore, in this paper, we use a server's CPU utilization threshold to represent its power budget for the simplicity of our analysis.

Latency-sensitive applications usually contain a large number of request-response operations (e.g., a request to the server and the server's response back to the client). However, power budget violation probably delays such operations. Taking a HTTP Web server as an example, the response time will increase significantly if power budget violation occurs [13, 15]. Therefore, the selection of the power budget requires a cautious tradeoff between the power margin alleviation and the performance degradation of latency-sensitive applications.

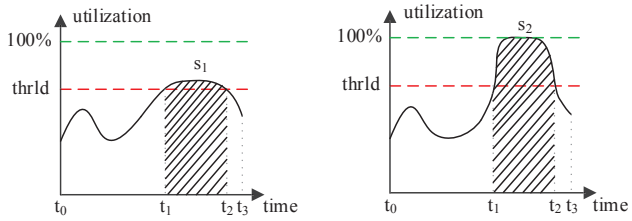
B. Motivation

Several methods have been used for evaluating the performance degradation caused by power budget violation, such as the approaches proposed in [2, 8, 9]. However, they all have weaknesses when evaluating the performance degradation of latency-sensitive applications. In this subsection, we analyze the problems of two typical state-of-the-art methods, which motivates us to devise a new one.

For example, [2] and [8] propose an evaluation method that measures the performance degradation by the time percentage of the power budget which is violated, which is denoted by *PBV* (percentage of budget violation). That is, the longer the violation of the power budget, the severer the performance degradation.

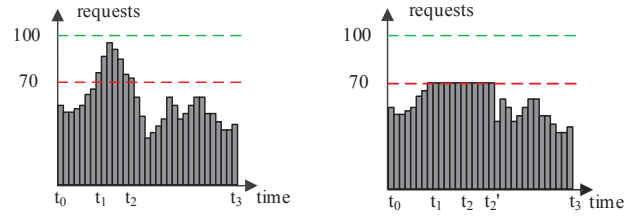
Fig. 2 shows two cases of power budget violations in a scenario where the two curves represent the CPU utilization demands of two servers over time and the two shaded areas represent the parts of jobs causing power budget violation, respectively. As shown in Figs. 2(a) and 2(b), the power budget is violated during the time from t_1 to t_2, which lies in the total time span from t_0 to t_3. In these two cases, the performance degradation values given by *PBV* are both $(t_2-t_1)/(t_3-t_0)$. However, the CPU utilization demand in Fig. 2(b) is much steeper than that in Fig. 2(a) during the time when the power budget is violated, and its shaded area

shown in s_2 is much larger than s_1 in Fig. 2(a). Intuitively, the performance degradation in Fig. 2(b) is severer, although its performance degradation value given by *PBV* is equal to that in Fig. 2(a). In summary, *PBV* cannot describe the performance degradation of latency-sensitive applications accurately, because it cannot reflect the degree of violation during the time when the power budget is violated.



(a) Case with slight violation (b) Case with serious violation

Figure 2. The performance degradations given by *PBV* in the above cases are the same, but the actual performance degradations are different.



(a) The incoming requests (b) The processed requests

Figure 3. The performance degradation given by *PPL* is 0 although budget violation exists in this case.

The method of [9] uses the percentage of work done during an observed time span to measure the performance loss of applications, which is denoted as *PPL* (percentage of performance loss). This method is useful for batch jobs which mainly concern on the completion time. However, it is too coarse-grained to evaluate the performance degradation of latency-sensitive applications, which usually contain a great deal of request-response operations. To demonstrate *PPL*'s drawback, here we take as an example a Web application which can process 100 requests per second at its best and 70 requests per second after restricting the power budget of the server. Now we introduce the scenario where a client sends requests at the rate in Fig. 3(a) and the number of total requests between t_0 and t_3 is 2000. The request-processing rate under this power budget is shown in Fig. 3(b), where the number of requests processed between t_0 and t_3 is also 2000; that is, no requests are lost. Therefore, the performance degradation given by *PPL* is 0. However, a performance degradation exists when power budget violation occurs, because some requests arriving between t_1 and t_2 cannot be processed immediately. This scenario shows that *PPL* is not suitable to evaluate the performance degradation of latency-sensitive applications because it fails to capture the performance loss of those tiny parts of these applications (e.g., the delay of a request-response operation of Web applications).

To address the drawbacks of existing methods, in this paper we aim to devise an appropriate method to evaluate the performance degradation of latency-sensitive applications.

C. Our Basic Idea

As we have analyzed above, both *PBV* and *PPL* have their shortcomings in evaluating the performance degradation of latency-sensitive applications. In this section, we introduce the basic idea of our method, which can address the problems of state-of-the-art methods.

In Fig. 2, we use the shaded area to reflect the degree of power budget violation, that is, the degree of applications' performance degradation. Typically, the larger the shaded area, the more severe the power budget violation. In order to measure the shaded area, we introduce the concept of CPU Workload (*Workload* for short), which is the integration of CPU utilization (i.e., $f_0(t)$) during a period of time (i.e., from t_0 to t_1), as shown in Equation 1.

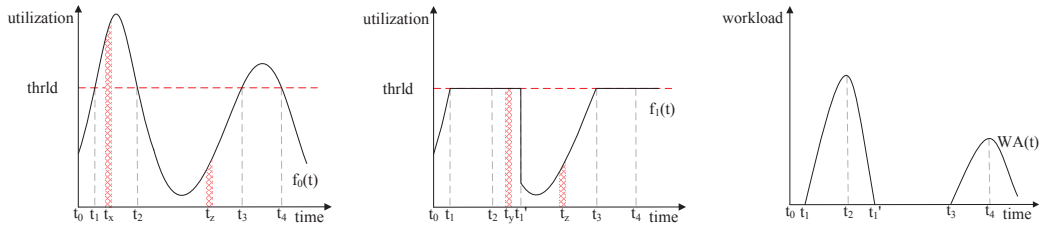
$$Workload = \int_{t_0}^{t_1} f_0(t)dt \tag{1}$$

The meaning of *Workload* is the CPU cycles serving an application's requirement during the time from t_0 to t_1. Because each request-response operation of the latency-sensitive application will consume a certain number of CPU cycles, we can use *Workload* to refer to a set of request-response operations.

Through the use of *Workload*, the problem of *PBV* presented in Fig. 2 can be solved, because more *Workload* is affected by the power budget violation in Fig. 2(b) , which implies more serious performance degradation. The weakness of *PPL* is that it is probably too coarse-grained to evaluate latency-sensitive applications (it neglects the violation occurring from t_1 to t_2 as shown in Fig. 3(a)). On the contrary, in this paper, we consider each *differential Workload* in a very narrow time span, for example, the red shaded areas at t_x, t_y, and t_z in Figs. 4(a) and 4(b), and the integration of *differential Workload* during a long time span composes the overall *Workload*. With *differential Workload*, the power budget violation from t_1 to t_2 in Fig. 3 can be easily captured, which is crucial for evaluating the performance degradation of latency-sensitive applications. Hence, the problem of *PPL* presented in Fig. 3 can also be solved.

III. Our Approach

In this section, we first analyze how power budget violation affects the performance of latency-sensitive applications. Second, we propose two metrics based on the concept of *differential Workload* in order to effectively evaluate the performance degradation of latency-sensitive applications caused by power budget violation. Third, we present how to



(a) CPU utilization demand (b) Actual CPU utilization (c) Workload accumulation and impact

Figure 4. This is a scenario showing a server's variation of CPU utilization and *WA(t)*. When the server's CPU utilization demand exceeds the threshold, its actual CPU utilization will be capped below the $thrld$ and *WA(t)* will increase.

apply our evaluation method to set a proper power budget for the servers in the datacenters.

A. Impact of Power Budget Violation

We analyze the impact of power budget violation on *Workload* with the scenario as shown in Fig. 4. Specifically, Fig. 4(a) presents the CPU demand over time (denoted as f_0) of a server. If setting a CPU utilization threshold at the value of $thrld$, the power budget violation will occur during the time from t_1 to t_2 and from t_3 to t_4, and its actual CPU utilization will be the curve (denoted as f_1) as shown in Fig. 4(b). A tiny part of *Workload* during the time when power budget violation occurs (e.g., *differential Workload* denoted by the red bar at t_x in Fig. 4(a)) cannot be served immediately, and it will accumulate until there are abundant CPU resources (e.g., the *differential Workload* at t_x in Fig. 4(a) is delayed and is finally served at t_y in Fig. 4(b)).

In order to evaluate the performance degradation of latency-sensitive applications, we introduce several key functions as follows.

First, any *differential Workload* could be delayed for a period of time, and the time period could be different. To express the delay of *differential Workload* at time t, we propose the function $Delay(t)$ (e.g., $Delay(t_x) = t_y - t_x$, and $Delay(t_z) = 0$ in this scenario).

Second, the amount of accumulated *Workload* varies over time, which reflects the degree of lack of CPU resources. To express the accumulated *Workload* at time t, we propose the function $WA(t)$. As shown in Fig. 4(c), from the beginning of t_1 (when power budget violation happens), more and more *Workload* is accumulated, so $WA(t)$ keeps growing. From t_2, the CPU utilization demand as shown in Fig. 4(a) falls below $thrld$; that is, there are abundant CPU resources available. Thereby $WA(t)$ starts to fall from t_2 until it becomes 0 at t_1', when the value of actual CPU utilization is the same as the CPU demand again. Hence, $WA(t)$ can reflect the real-time relationship between the CPU utilization demand and available CPU resources. Whenever $WA(t) > 0$, it generates delay of *Workload* (performance degradation).

Third, as time goes on, both the amount of total *Workload* submitted to the server and the summation of delayed *differential Workload* increase. For example, the total *Workload* submitted to the server between t_0 and t_2 is the area covered by f_0 from time t_0 to t_2 in Fig. 4(a), and the summation of delayed *Workload* is the total area of all *differential Workload* whose delay is greater than 0 (e.g., the red bar at t_x). To express these two kinds of *Workload* at time t in our analysis, we use the functions $TotalWorkload(t)$ and $DelayedWorkload(t)$, respectively.

B. Evaluation of Performance Degradation

As discussed above, the performance degradation of latency-sensitive applications is generated when workload is accumulated ($WA(t) > 0$). In this subsection, we devise two metrics of our method based on the integral of *differential Workload* to accurately evaluate the performance degradation and then present an algorithm to obtain these two metrics.

In order to evaluate the performance degradation accurately, we need to answer two key questions as follows:

- What percentage of *Workload* is delayed?
- How long is *Workload* delayed?

To answer the first question, we use *PDW* (percentage of delayed *Workload*) as a metric. It represents the ratio of delayed *Workload* to all *Workload* during a time span. The higher *PDW* is, the more *Workload* is affected by power budget violation. For the second question, we use *AD* (average delay) as a metric to describe the average delay of all delayed *differential Workload*.

The expressions of a server's *PDW* and *AD*, with its CPU utilization data (i.e., $f_0(t)$) during the time from 0 to t, are shown in Equation 2.

$$\begin{cases} PDW = \frac{DelayedWorkload(t)}{TotalWorkload(t)} \\ AD = \frac{\int_0^t f_0(t)Delay(t)dt}{DelayedWorkload(t)} \end{cases} \quad (2)$$

For *PDW*, $DelayedWorkload(t)$ at time t refers to the part of *Workload* whose latency is greater than 0. By dividing delayed *Workload* by $TotalWorkload(t)$, we can obtain the percentage of *Workload* delayed. For *AD*, $\int_0^t f_0(t)Delay(t)dt$ refers to the summation of the product of every delayed *differential Workload* (i.e., $f_0(t)dt$) and its delay (i.e., $Delay(t)$). If we divide it by $DelayedWorkload(t)$, the result will be the average delay of every *differential Workload*.

Next, we demonstrate the expression of $DelayedWorkload(t)$ and $TotalWorkload(t)$.

$$TotalWorkload(t) = \int_0^t f_0(t)dt \tag{3}$$

Equation 3 is the expression of $TotalWorkload(t)$, which is the integral of CPU utilization ($f_0(t)$) between 0 and t.

$$DelayedWorkload(t) = \int_0^t DelayedWorkload(t)'dt \tag{4}$$

Equation 4 is the expression of $DelayedWorkload(t)$, which is the integral of $DelayedWorkload(t)'$ between 0 and t. The $DelayedWorkload(t)'$ can be understood as the growing speed of $DelayedWorkload(t)$, and is calculated by Equation 5.

$$DelayedWorkload(t)' = \begin{cases} 0 & WA(t) = 0 \\ f_0(t) & WA(t) > 0 \end{cases} \tag{5}$$

Equation 5 is the expression of $DelayedWorkload(t)'$, whose value is subjected to two conditions. When $WA(t) = 0$, there is no accumulated $Workload$, and thus there is no new delayed $Workload$ either. When $WA(t) > 0$, all *differential Workload* at time t will be delayed, and thus the value of $DelayedWorkload(t)'$ is $f_0(t)$.

In order to compute these two metrics of our evaluation method, we present an algorithm based on the calculus method. Generally, monitoring modules of a datacenter record the CPU utilization of each server. Here, the sample interval is denoted as T.

Specifically, this algorithm obtains the result by iterating all the sample intervals. We use the variables $delayedWorkload$, $totalWorkload$, wA, and $productSum$ to record the values of functions $DelayedWorkload(t)$, $TotalWorkload(t)$, $WA(t)$, and $\int_0^t f_0(t)Delay(t)dt$ in the iteration process, respectively.

In Algorithm 1, we want to obtain the performance degradation of a server under the desired CPU utilization threshold (denoted as $thld$) from 0 to 100%. Hence, in line 1, we iterate our evaluation operation 101 times. Meanwhile, as we treat the *Workload* in each sample interval as a piece of *differential Workload*, we iterate all n samples in line 2. Because $TotalWorkload(t)$ at time t is the integral on all *differential Workload* from 0 to t, we add the new *differential Workload* to $totalWorkload$ at each time in line 3. In lines 4 to 7, we refresh the value of wA by adding the integral of the difference between u_i and $thld$ at time t. That is, if $u_i > thld$, accumulated *Workload* increases (accumulated *Workload* decreases when $u_i < thld$). At the mean time, wA is guaranteed to be non-negative. In lines 8 to 11, we decide whether to add $delayedWorkload$ and $productSum$ by judging whether $wA > 0$. The $delayedWorkload$ and $productSum$ will increase only when $wA > 0$. In line 13, we compute and record the two metrics' results for the corresponding CPU utilization threshold by Equation 2. Using this algorithm, we can acquire the performance degradation (*PDW* and *AD*) of any server under a desired CPU utilization threshold.

Algorithm 1 Performance degradation algorithm

Input: The CPU utilization trace of a server with n samples, where u_i is the CPU utilization at time i, and T is the sampling interval.
Output: *PDW* and *AD*

```
1: for each thrld in[0, 100] do
2:     for each i in[0, n] do
3:         totalWorkload = totalWorkload + u_i × T
4:         wA = wA + (u_i − thld) × T
5:         if wA < 0 then
6:             wA = 0
7:         end if
8:         if wA > 0 then
9:             delayedWorkload = delayedWorkload + u_i × T
10:            productSum = productSum + wA × T
11:        end if
12:    end for
13:    compute PDW and AD at thld by using Equation 2 and record the results
14: end for
```

Cost Analysis: In our algorithm, each round of computation iterates all samples, and thus its computing complexity is $O(n)$, where n is the number of samples in the CPU utilization trace. On the other hand, the algorithm only uses a few variables in the whole computing procedures, and hence its space complexity is only $O(1)$.

C. Use Case in Datacenter

We have introduced our evaluation method comprising two metrics *PDW* and *AD*, which can reflect the performance degradation of latency-sensitive applications. Before adopting our method, datacenters should provide servers' CPU utilization traces, and thus our evaluation method can obtain the performance degradation of a server with each CPU utilization threshold by using Algorithm 1. Then, by combining the mapping relationship between CPU utilization and power, we can obtain servers' performance degradation under the restriction of any power budget. Finally, administrators choose a power budget for each server with acceptable performance degradation. In this subsection, we illustrate a possible way in which our evaluation method can be applied in a datacenter.

In a large-scale datacenter, every server is typically assigned a unique key denoted as SID. Taking a server whose SID is 10001 as an example, if we want to evaluate its performance degradation under different power budgets, we

Table I
THE INPUT AND OUTPUT OF A SERVER (SID=10001)

(a) Mapping table

utilization	*power*
100%	400*W*
90%	380*W*
80%	350*W*
70%	320*W*
60%	310*W*
...	...

(b) CPU trace

time	*utilization*
12:00:00	13%
12:00:01	10%
12:00:02	9%
12:00:03	11%
12:00:04	15%
...	...

(c) Performance degradation

budget	*thrld*	*PDW*	*AD*
...	...	...	...
213*W*	31%	0%	0 *sec*
210*W*	30%	1.4%	0.05 *sec*
208*W*	29%	3.1%	0.28 *sec*
207*W*	28%	5.1%	1.13 *sec*
...	...	...	...

need a table mapping its CPU utilization to power (as shown in Table I(a)) and its CPU utilization trace (as shown in Table I(b)). Typically, Table I(a) is provided by the manufacturer of this server. If this information is not provided, it is easy for administrators to obtain such information by measurements in the lab. To monitor servers, datacenters usually deploy agents to record the status data (e.g., CPU utilization) of the servers, which can be used to form Table I(b). By inputting Table I(a) and Table I(b) into the analysis module which has integrated our evaluation method, the module can output the performance degradation (*PDW* and *AD*) of this server under different performance budgets and CPU utilization thresholds as shown in Table I(c).

There are two metrics expressing the performance degradation in our evaluation method: *PDW* and *AD*. When selecting an acceptable performance degradation, administrators can refer to the metric which is more important to them. For example, if the administrators care more about what percentage of an application's workload is delayed in the server (SID is 10001) and their maximum acceptable *PDW* is 4%, then they can choose 208 W as the power budget of the server (208 W is the smallest power budget whose associated *PDW* is less than or equal to 4%). If we use function $f_{PDW}(PDW_a)$ to express the power budget associated with acceptable *PDW*, then $f_{PDW}(0.04) = 208$. On the other hand, if the administrators care more about how long the application is delayed, and their maximum acceptable *AD* is 0.03 seconds, then they can choose 210 W as the power budget of the server (210 W is the smallest power budget whose associated *AD* is less than or equal to 0.03 seconds). If we use function $f_{AD}(AD_a)$ to express the power budget associated with acceptable *AD*, then $f_{AD}(0.03) = 210$.

However, there are also some cases in which administrators want to select a power budget which can satisfy the demanded *PDW* and *AD* simultaneously. A simple solution is to select the smaller power budget of the two values that satisfies acceptable values of *PDW* and *AD*. We use the function $f_{PDW\_AD}(PDW_a, AD_a)$ to express the power budget satisfying acceptable values of *PDW* and *AD* simultaneously, as shown in Equation 6. For example, if administrators select the power budget at which the acceptable *PDW* is 4% and *AD* is 0.03 seconds, the power budget will be 210 W.

$$f_{PDW\_AD}(PDW_a, AD_a) = max(f_{PDW}(PDW_a), f_{AD}(AD_a)) \quad (6)$$

Table II
SUGGESTIONS ABOUT POWER BUDGET

SID	PDW	AD	$budget$	$CPU\ thrld$
10001	2%	$0.1sec$	$210W$	30%
10002	3%	$0.5sec$	$350W$	78%
10003	1%	$0.1sec$	$200W$	21%
10004	5%	$1.1sec$	$300W$	60%
10005	4%	$0.4sec$	$210W$	32%
...	...	...	...	...

After the administrators have selected their acceptable values of *PDW* and *AD* (e.g., *PDW* = 2%, *AD* = 0.1 second) for the server whose SID is 10001, its power budget can be determined (210 W). When the administrators apply the evaluations on all servers in their datacenter, they will finally obtain a table as shown in Table II. We call this table a power budget decision list. Finally, the administrators can allocate the power budget to all servers by referring to this list.

IV. EXPERIMENTS

The experiments consist of three parts. In the first part, taking the measured results as the baseline, we will verify the accuracy of our evaluation method compared with existing approaches. In the second part, we apply our evaluation method on the traces of some representative servers in Tencent's datacenter and show the possible performance degradation of servers under different power budgets. In the third part, we carry out our evaluation on the traces of Tencent's datacenter with 25328 physical nodes hosting latency-sensitive applications to see how much the power budget can be saved in a production datacenter with acceptable performance degradation.

A. Verification of Our Method

In this experiment, we apply our evaluation method and state-of-the-art methods (*PBV* [2, 8] and *PPL* [9]) on the experimental server under different power budget thresholds and obtain the performance degradation calculated by these methods. We measure the practical performance degradation of the server as the baseline so as to verify the accuracy of our method and the state of the art by comparing the difference between the calculated and measured results.

We use two physical nodes to act as a Web server and a client, respectively. The Web server hosts an Apache HTTP Server application and constantly serves the HTTP requests from the client. All requests to the server access the same PHP page and thus consume fixed CPU resources

(because latency-sensitive applications consume a very tiny amount of CPU resources, the assumption of consuming fixed CPU resources does not affect the generality). At the client node, we devise an HTTP requests generator which can send a specified amount of requests per second as needed. Therefore, by controlling the sending rate of requests from the client, we can control the CPU utilization in the server node.

WorldCup98 [16] is a dataset consisting of the requests made to the World Cup website during the match period in 1998, where the number of requests to the Web server in each second is recorded. We transfer the trace of WorldCup98 into a CPU utilization trace as shown in Fig. 5.

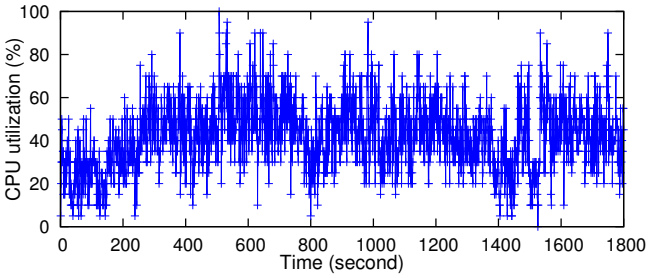
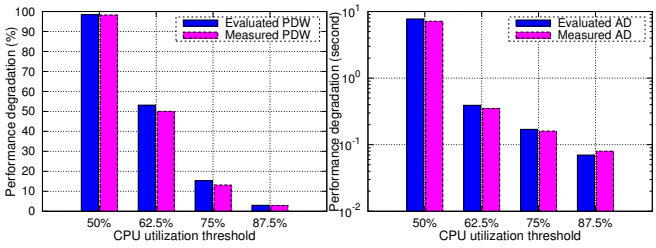


Figure 5. The CPU utilization of a server in WorldCup98

At the client, we record the response time of every request on the log. We take the response time of a request not affected by power budget violation as a datum. Based on this datum line, we can measure two kinds of data with the response time log: What Percentage of Requests Are Delayed, and How Long These Requests Are Delayed on Average. Hence, they can represent the measured results of metrics *PDW* and *AD* (denoted as Measured *PDW* and Measured *AD*).



(a) PDW (b) AD

Figure 6. The comparison of measured performance degradation and the evaluated results calculated by our evaluation method

To explore the performance degradation results under different CPU utilization thresholds, we set the threshold at 50%, 62.5%, 75%, and 87.5%, respectively (realized by disabling cores). The experimental results are shown in Fig. 6, where the evaluated results (denoted as Evaluated *PDW* and Evaluated *AD*) are the two metrics calculated by our evaluation method, and the measured results (denoted as Measured *PDW* and Measured *AD*) are the measured data calculated by the response time recorded in the log.

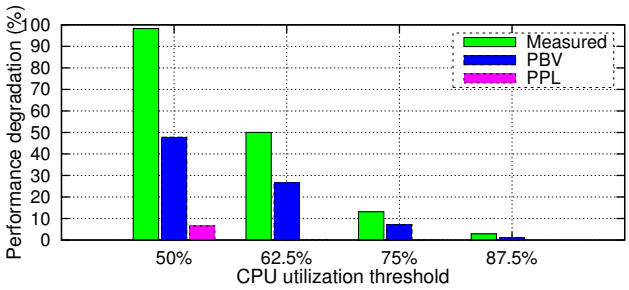


Figure 7. The comparison of measured performance degradation and evaluated results calculated by state-of-the-art methods

As shown in Fig. 6(a), when the CPU utilization threshold is 50%, 62.5%, 75%, and 87.5%, the Evaluated *PDW* is 98.4%, 53.2%, 15.4%, and 3%, which is very close to the Measured *PDW* of 98.3%, 50%, 12.3%, and 3%, respectively. By comparing the results of Evaluated *PDW* and Measured *PDW*, we can find that Measured *PDW* is very close to Evaluated *PDW* (the average difference is 3.3%). In other words, the first metric *PDW* in our evaluation method can effectively evaluate the performance degradation.

The results of *AD* are shown in Fig. 6(b). When the CPU utilization threshold is set to 50%, 62.5%, 75%, and 87.5%, respectively, Evaluated *AD* is 7.74, 0.39, 0.17, and 0.07 seconds, while Measured *AD* is 7.15, 0.35, 0.16, and 0.08 seconds. Comparing the results of Evaluated *AD* with Measured *AD*, we find they are very close to each other too (the average difference is 7.5%). This demonstrates that the second metric *AD* in our evaluation method can also accurately evaluate the performance degradation.

We then further compare the results calculated by the state-of-the-art methods *PBV* and *PPL* with the measured performance degradation. Considering that *PBV* and *PPL* are both evaluation methods based on the percentage of applications affected, we choose What Percentage of Requests Are Delayed as the referential data (denoted as Measured). The performance degradations given by these two evaluation methods *PBV* and *PPL* are denoted as *PBV* and *PPL*. The results are shown in Fig. 7. With a CPU utilization threshold of 50%, 62.5%, 75%, and 87.5%, *PBV* is 47.8%, 26.7%, 7.2%, and 1.1%, while *PPL* is 6.5%, 0.24%, 0.1%, and 0%, respectively. We find that both *PBV* and *PPL* are far lower than the measured performance degradation (the average difference in *PBV* is 49.6%, and that of *PPL* is 95.8%). In particular, *PPL* is much more inaccurate than *PBV*.

In the above experiments, we adjust the CPU utilization threshold to multiple possible levels (from 50% to 87.5%) to verify the accuracy of our method and existing approaches. The experimental results demonstrate that our method can evaluate the performance degradation of latency-sensitive applications accurately (the average difference in *PDW* and *AD* is 3.3% and 7.5%). These results also demonstrate that the performance degradation calculated by existing approaches is far lower than the measured performance degradation, which means they are both over-optimistic

in their evaluation. *PBV* has ignored the effect of the accumulation of the former *Workload* (a delay in *Workload* will delay the next *Workload*), which aggravates the performance degradation. Thus *PBV* has overlooked a part of the performance degradation. *PPL* also neglects a part of the performance degradation because it only considers the time taken to accomplish an application, instead of considering every *differential Workload*'s delay, which is important for latency-sensitive applications. In particular, *PPL* is much more optimistic compared with *PBV* (the average difference in *PPL* is much higher than that in *PBV*), which demonstrates that *PPL*, which only considers the time taken to accomplish an application, will miss more performance degradation of latency-sensitive applications.

The above experiments have already shown that *PBV* and *PPL* are not appropriate to evaluate the performance degradation of latency-sensitive applications, and thus in our next experiments performing evaluations on production servers we will not use these two methods.

B. Evaluation on Typical Tencent's Servers

We have already verified the accuracy of our evaluation method in previous experiments; now we want to apply our evaluation method to production servers and see how the performance degradation varies under different restricted power budgets. For generality, we will select some representative servers based on application types and average load. Then, we evaluate the performance degradation under different CPU utilization thresholds and show how to set a CPU utilization threshold with acceptable performance degradation.

Typical latency-sensitive applications include Web pages, online games, and streaming services [17]. To evaluate the performance degradation of these kinds of applications, traces from Tencent's production datacenter are a great choice because it comprises 10166, 7824, and 7338 nodes deploying Web pages, games, and videos, respectively. We will denote them as web, game, and video for short. Tencent's trace is collected by each agent deployed on running servers and stored in the database in MySQL records. The monitoring data comprise fields including the CPU, memory, disk, network, and alert information. We take only the CPU utilization of all these fields during ten days for the performance degradation evaluation.

To transfer CPU utilization to server power usage, we adopt the model ($Power = Power_{idle} + (Power_{busy} - Power_{idle}) * CPU\ Utilization$) described in [2]. $Power_{idle}$ and $Power_{busy}$ mean the power consumption when the CPU utilization is 0 and 100%. For simplicity, we assume that $Power_{busy}$ is 300 W and $Power_{idle}$ is 150 W for all servers. Thus, if the CPU utilization of a server is 50%, then its power usage is 225 W. This model is useful when we calculate how much the power budget can be restricted after determining the CPU utilization threshold.

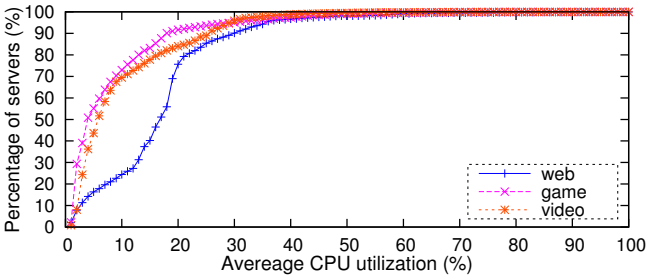
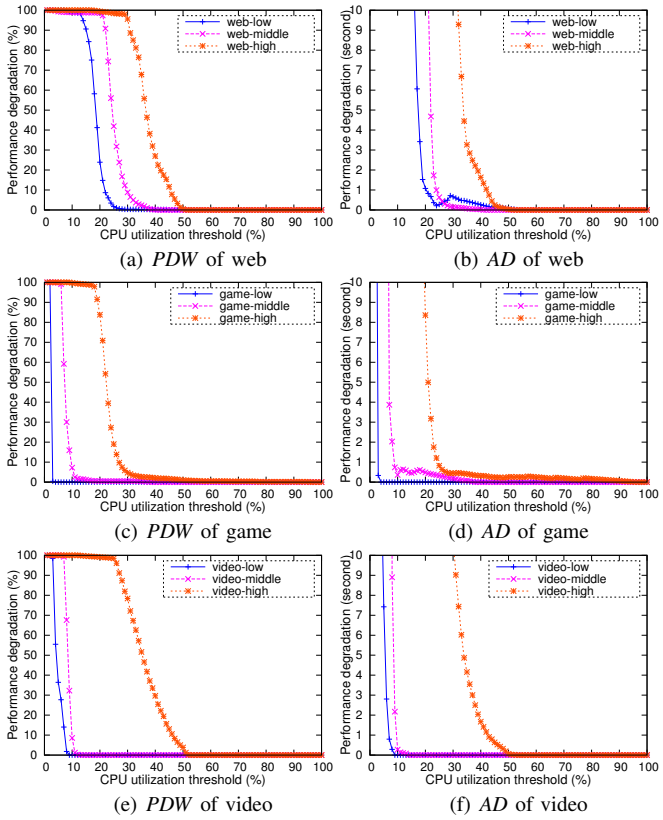


Figure 8. The percentage of servers whose average CPU utilization is lower than a specified value

To explore the CPU utilization feature of servers in Tencent's datacenter, we count how the average CPU utilization of these servers is distributed. Fig. 8 presents the percentage of servers whose average CPU utilization is below a specific value. As we can see, among the servers deploying Web pages, about 80% of servers have an average CPU utilization lower than 20%. The average CPU utilization of servers deploying games and videos is much lower (80% of servers have an average CPU utilization lower than 13%), and a possible reason for this is that games and videos are much less CPU-intensive. Without loss of generality, we choose nine representative servers based on application types and average load in our evaluation. We grade the average CPU utilization of servers into three classes based on their rankings among the servers of the respective application types, that is, high for top 10% ranking, middle for top 40% ranking, and low for 70% ranking. These nine representative servers are denoted as web-high, web-middle, web-low, game-high, game-middle, game-low, video-high, video-middle, and video-low, respectively.

The evaluation results of these nine representative servers are shown in Fig. 9. The most common feature reflected by these figures is that performance degradation (both *PDW* and *AD*) increases with decreasing CPU utilization threshold (e.g., the *PDW* of web-middle in Fig. 9(a) is 6.49% when the CPU utilization threshold is 30%, but grows to 31.78% when the threshold is 25%). This regulation tells us that if we want to obtain the revenue of a lower CPU utilization threshold, we need to bear a harsher performance degradation. However, there are also some exceptions to this regulation for *AD* (e.g., web-low in Fig. 9(b)), but the overall trend is not affected.

With deeper investigation of the results in Fig. 9, we can summarize two key points. First, we find that there is much room to reduce the CPU utilization threshold for all servers (e.g., web-high can reduce its CPU utilization threshold to 50% with almost no performance degradation, and the CPU utilization threshold can be lower with web-middle and web-low). This implies that we can potentially achieve great room for power budget restriction with servers deploying latency-sensitive applications. Second, in some cases there is a point when performance degradation starts to grow drastically as the CPU utilization threshold decreases



(a) *PDW* of web (b) *AD* of web

(c) *PDW* of game (d) *AD* of game

(e) *PDW* of video (f) *AD* of video

Figure 9. The performance degradation of Tencent's typical servers under different CPU utilization thresholds

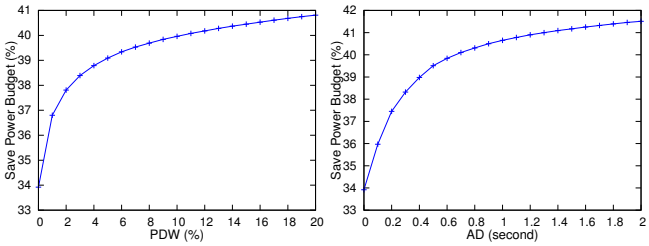
(e.g., *AD* starts to grow drastically at the CPU utilization threshold of 27% with web-low in Fig. 9(b)). We can call this point a maximum-benefit point, and we suggest that the CPU utilization threshold be set before this point, in order to obtain the benefit with relatively low performance degradation.

An important phenomenon in Fig. 9 is that servers deploying games and videos have more room to restrict the power budget compared with servers deploying Web pages. For example, if the acceptable *PDW* is 5%, the minimum CPU utilization threshold that can be set for web-middle, game-middle, and video-middle is 31%, 10%, and 10% (reductions of 103.5 W, 135 W, and 135 W, respectively). The reason why servers deploying games and videos can be restricted to a lower CPU utilization threshold is reflected in Fig. 8, where the average CPU utilization of most of these servers is far lower than that of servers deploying Web applications.

To summarize, if we accept more performance degradation (higher *PDW* and *AD*), we can reduce the power budget of servers further by setting a lower CPU utilization. Inspiringly, these representative servers present great room for the power budget restriction even though no performance degradation is allowed (i.e., *PDW* is 0% and *AD* is 0 sec). Hence, it is very meaningful to restrict the power budget of production servers with little performance degradation.

C. Evaluation on Tencent's Datacenter

In the previous section, we carried out performance degradation evaluations on some representative servers in Tencent's datacenter. Now, we aim to determine exactly how much room for power budget restriction we can achieve at the scale of the whole datacenter. We take the traces of all 25328 servers (deploying Web pages, games, and videos) in Tencent's datacenter, exploring what percentage of the power budget can be restricted.



(a) *PDW* (b) *AD*

Figure 10. The restricted percentage of the power budget of all servers with different levels of performance degradation

In order to show the room for power budget restriction of all 25328 servers directly in Tencent's datacenter, we transfer the servers' CPU utilization threshold into the power budget and explore what percentage of the power budget can be restricted in all of these servers. Fig. 10 presents the restricted percentage of the power budget that complies with the acceptable performance degradation (*PDW* and *AD*). Both Figs. 10(a) and 10(b) demonstrate that even if no performance degradation is accepted, we can restrict 33.92% of the power budget of these servers. With increases in *PDW* and *AD*, we can further restrict the power budget (e.g., we can restrict 39.96% of the power budget when *PDW* is 10%, and we can restrict 40.65% of the power budget when *AD* is 1 sec). Hence, by controlling the performance degradation, Tencent's servers are capable of restricting at least 33.92% of the power budget. That is to say, it is very helpful to use our evaluation method as guidance to restricting the power budget of servers in the production datacenter, which helps increase the datacenter efficiency.

V. Related Work

Recent researches [2, 8, 10, 18] have noted that power over-provisioning has greatly impeded the utilization of resources in datacenters. Reference [2] investigates the power consumption in the Google datacenter, showing the great power margin brought by power over-provisioning in servers and racks. Reference [18] also concentrates on the power margin and studies the way to make use of this part of the unutilized power budget.

A straightforward way to eliminate the power margin is by restricting the power budget by power capping [2, 8]. Intel Node Manager [19] facilitates power capping. However, performance degradation of applications may be caused by

power budget violation when the power budget is restricted. Reference [2] evaluates the performance degradation by measuring what percentage of time is spent in budget violation (*PBV*), and [8] evaluates it by measuring how much of the work done is lost (*PPL*). Although these two methods are suitable in some specialized scenarios (e.g., *PPL* is very effective in evaluating the performance degradation of latency-insensitive applications), they have drawbacks in describing the performance degradation of latency-sensitive applications accurately. Meanwhile, there are also some other methods that are used to evaluate the performance degradation of servers. Reference [5] carries out experiments to measure the performance by consolidating any two jobs. This method can be accurate, but it also carries a substantial experimental cost. Reference [20] assumes that the servers with similar CPU utilization characteristics bear similar performance degradations. This method is efficient when there are abundant jobs whose CPU utilization characteristics are similar. Reference [21] includes memory access time as another factor that affects latency other than the CPU time. It provides a new model for predicting job latency with multiple factors. Similar to the thought of *workload* accumulation in our work, [22] also assumes that *workload* not accomplished will be delayed to the next time interval, but it does not give an evaluation of the performance degradation when power budget violation occurs.

VI. Conclusion

In this paper, we analyze the reasons why the state-of-the-art methods are not appropriate to evaluate the impact of power budget violation on latency-sensitive applications' performance. Then we propose a new evaluation method which can provide a fine-grained way to evaluate such an impact precisely. We verify our proposed method by a real-world application and evaluate the performance degradation of 25328 servers in a production datacenter. Experimental results demonstrate that our evaluation method behaves more accurately than other state-of-the-art methods, and we can achieve substantial room for power budget restriction of servers while maintaining the applications' performance degradation within controllable and acceptable range.

Acknowledgement

The research is supported by National Science Foundation of China under grant No.61232008 and No.61472151, National 863 Hi-Tech Research and Development Program under grant No.2015AA011402 and No.2014AA01A302.

References

[1] Tencent corporation, http://www.tencent.com/.

[2] X. Fan, W. Weber, and L. Barroso. Power provisioning for a warehouse-sized computer. In *Proc. of ISCA*, pages 13–23. ACM, 2007.

[3] T. Imada, M. Sato, Y. Hotta, and H. Kimura. Power management of distributed web savers by controlling server power state and traffic prediction for QoS. In *Proc. of IPDPS*, pages 1–8. IEEE, 2008.

[4] Z. Zhou, F. Liu, H. Jin, B. Li, and H. Jiang. On arbitrating the power-performance tradeoff in SaaS clouds. In *Proc. of INFOCOM*, pages 872–880. IEEE, 2013.

[5] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proc. of MICRO*, pages 248–259. ACM, 2011.

[6] L. Tang, J. Mars, and M. Soffa. Compiling for niceness: Mitigating contention for QoS in warehouse scale computers. In *Proc. of CGO*, pages 1–12. ACM, 2012.

[7] D. Meisner, C. Sadler, L. Barroso, W. Weber, and T. Wenisch. Power management of online data-intensive services. In *Proc. of ISCA*, pages 319–330. IEEE, 2011.

[8] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *Proc. of ASPLOS*, pages 48–59. ACM, 2008.

[9] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *Proc. of HPCA*, pages 101–110. IEEE, 2008.

[10] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proc. of ISCA*, pages 66–77. IEEE, 2006.

[11] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: Comparing public cloud providers. In *Proc. of IMC*, pages 1–14. ACM, 2010.

[12] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proc. of MICRO*, pages 347–358. IEEE, 2006.

[13] Y. Wang, X. Wang, M. Chen, and X. Zhu. Partic: Power-aware response time control for virtualized web servers. *TPDS*, 22(2):323–336, 2011.

[14] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proc. of ICAC*, pages 4–4. IEEE, 2007.

[15] S. Wang, J. Chen, J. Liu, and X. Liu. Power saving design for servers under response time constraint. In *Proc. of ECRTS*, pages 123–132. IEEE, 2010.

[16] Worldcup98, ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[17] S. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proc. of MMSys*, pages 35–46. ACM, 2010.

[18] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *Proc. of SIGMETRICS*, pages 157–168. ACM, 2009.

[19] Intel node manager, http://www.intel.com/content/www/us/en/data-center/data-center-management/node-manager-general.html.

[20] C. Delimitrou and C. Kozyrakis. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proc. of ASPLOS*, pages 77–88. IEEE, 2013.

[21] B. Su, J. L. Greathouse, J. Gu, M. Boyer, L. Shen, and Z. Wang. Implementing a leading loads performance predictor on commodity processors. In *Proc. of ATC*, pages 205–210. USENIX, 2014.

[22] W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proc. of ICS*, pages 293–302. ACM, 2005.