6-1-2009

Power Capping Via Forced Idleness

Anshul Gandhi
Carnegie Mellon University

Mor Harchol-Balter
Carnegie Mellon University, harchol@cs.cmu.edu

Rajarshi Das
IBM Research

Jeffrey O. Kephart
IBM Research

Charles Lefurgy
IBM Research

Power Capping Via Forced Idleness

Anshul Gandhi
Carnegie Mellon University
anshulg@cs.cmu.edu

Mor Harchol-Balter*
Carnegie Mellon University
harchol@cs.cmu.edu

Rajarshi Das
IBM Research
rajarshi@us.ibm.com

Jeffrey O. Kephart
IBM Research
kephart@us.ibm.com

Charles Lefurgy
IBM Research
lefurgy@us.ibm.com

Abstract

We introduce a novel power capping technique, *IdleCap*, that achieves higher effective server frequency for a given power constraint than existing techniques. IdleCap works by repeatedly alternating between the highest performance state and a low-power idle state, maintaining a fixed average power budget, while significantly increasing the average processor frequency. In experiments conducted on an IBM BladeCenter HS21 server across three representative workloads, IdleCap reduces the mean response time by up to a factor of 3 when compared to power capping using clock-throttling. Furthermore, we argue how IdleCap applies to next-generation servers using DVFS and advanced idle states.

1. INTRODUCTION

Many data center operators are facing severe power constraints as they attempt to fit new high-density servers within their existing data center power and cooling infrastructure [2]. One method for operating a server under a power constraint is *power capping*, which limits the average power consumption of the server over a time interval to stay within a budget [3, 5]. In recent years, IBM, HP, and Intel have deployed power capping in their products [4, 6, 12]. One way to achieve power capping is to run the server at a *fixed* frequency, where this frequency is the highest performance state or throttle state such that the server's power consumption in that state is below the required power budget. Another way to achieve power capping is to lower the processor frequency whenever the server's power consumption over unit time crosses the given power budget.

In this paper we not only aim to achieve the power capping goal, but also seek to simultaneously minimize *response time* of the workload. Our solution is to introduce a new power capping approach, *IdleCap*, that provides a higher time-averaged processor frequency for a given power consumption budget or *power cap*, thereby leading to lower response times. Like existing power capping techniques based on feedback control [13, 14, 20, 21] and soft-scaling [18], IdleCap dithers between available processor states to cap power consumption. What distinguishes IdleCap is that it alternates between *extreme* states (the highest performance state and a low-power idle state), as compared to existing power-capping techniques which dither between *adjacent* clock-throttling states (t-states) or DVFS states (p-states) [1]. In the implementation of IdleCap presented in this paper, the low-power idle state is the C1E state [9] available on modern Intel processors.

Importantly, IdleCap does *not* switch to the idle state simply when there is no work to do (as is common nowadays [16] to save power and energy). Rather, IdleCap *purposely* alternates between the highest performance state and the C1E state. Since the time involved in switching between C1E and the highest performance state is on the order of microseconds, IdleCap can easily achieve the required average power constraint on a 1 second (and possibly finer) timescale. The intuition and analysis behind IdleCap are presented in Section 2.

To verify our idea, we test IdleCap on a Woodcrest-based server [8] (Dual-Core Intel Xeon Processor 5160). We compare IdleCap with power capping via clock-throttling since clock-throttling is available on all server processors and offers a wide choice of power cap settings. By comparison, DVFS usually offers a reduced choice of power cap settings and is not available on all server processors. We observe a factor of 3 improvement in mean response time for CPU-bound (DAXPY [19] and LINPACK [10]) workloads and a factor 2 improvement for memory-bound (STREAM [15]) workloads. These improvements hold over a range of *alternation rates*, where the time between switches can vary from a fraction of a second to tens of seconds. A detailed discussion of the experimental setup and results is presented in Section 3.

IdleCap can also lead to higher effective frequency in processors with DVFS provided it has access to its advanced idle-states (e.g., Nehalem-based servers [7]). However, since such servers are not available in the market yet, we experiment with clock-throttling and the C1E idle-state in Woodcrest-based servers. In Section 4 we discuss how IdleCap can be used in future processors that support DVFS and more

*Research supported by NSF SMA/PDOS Grant CCR-0615262 and a 2009 IBM Faculty Award.

power-efficient C-states [9].

2. THE IDLECAP SCHEME

In this section, we introduce the IdleCap power capping scheme and the intuition behind it. Then, we analyze IdleCap for both CPU-bound and non CPU-bound workloads and establish that IdleCap will outperform power capping approaches based on clock-throttling for all power cap values in each case.

2.1 High-level algorithm

Consider the solid line in Fig. 1 (labeled *clock-throttling*), which displays the power consumed by the server used in our experiments as a function of the server frequency established by clock-throttling, when the server is 100% utilized by the DAXPY workload. We refer to this as the *power-to-frequency* relationship at 100% CPU utilization. Observe that, within the allowed frequency range of .375 GHz to 3 GHz, the power-to-frequency relationship is very close to linear. However, the power consumption in the C1E state, P_C, is 126 watts, which is 23 watts less than the value that the linear relationship would suggest, $P_0 = 149$ watts. Because of this non-linear drop in power below .375 GHz, the power-to-frequency relationship is actually *concave downwards*. This is the key fact that we exploit.

Suppose that one is given a power budget of 170 Watts. According to the clock-throttling graph, this allows us to run the server at a steady frequency of 1 GHz. Now suppose we purposely choose to use our 170 Watts instead to alternate between extreme power levels, alternating each second between approximately 220 Watts and 120 Watts, achieving an average power usage of 170 Watts. When running at 220 Watts, the server frequency is (about) 3 GHz. When running at 120 Watts, the server frequency is (about) 0 GHz. Hence, the average server frequency achievable by alternating between 220 Watts and 120 Watts is 1.5GHz. This fact is expressed by the dashed line connecting P_C and P_{max}.

Thus in the above example we have gone from a frequency of 1 GHz to 1.5 GHz without any additional power! This is the idea behind IdleCap. While the rest of our discussion focuses on clock-throttling, the same intuition is equally valid in processors with DVFS where $P_0 > P_C$.

2.2 More detailed algorithm

In Section 2.1, we maintained a fixed power budget of $\mathcal{P} =$ 170 Watts by alternating between 120 Watts and 220 Watts spending $r = \frac{1}{2}$ fraction of time in the highest performance state. More generally, IdleCap can maintain any fixed power budget, $\mathcal{P}$, by spending some r fraction of time in the highest performance state and $1-r$ fraction of time in the C1E state.

The relationship between power consumption and effective frequency for IdleCap is represented by the the dashed line in Fig. 1 (labeled IdleCap). Observe that the IdleCap curve is always below the *concave-downwards* clock-throttling curve in Fig. 1. Graphically, the horizontal *gap* between the two curves shows that for any desired power budget, $\mathcal{P}$, IdleCap's effective frequency exceeds that of clock-throttling.

For example, suppose we require that the system maintain a power cap of 157 watts while running the 100% CPU-bound

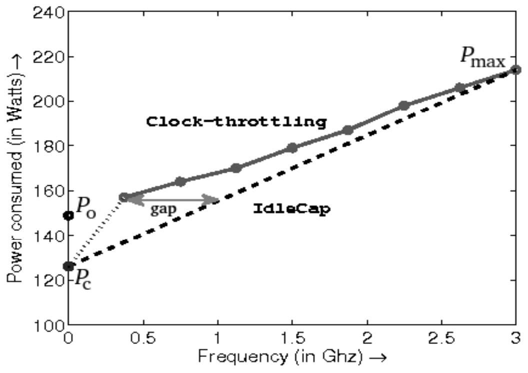


Figure 1: ***Solid line represents power consumed vs frequency for a 100% CPU-bound workload DAXPY, and P_C represents the C1E state. Dashed line represents IdleCap, which affords higher effective CPU frequency than clock-throttling for any power budget.***

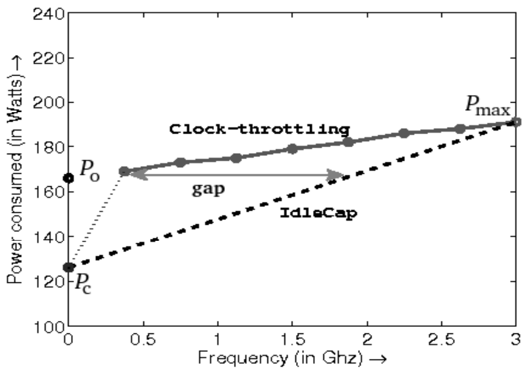


Figure 2: ***Solid line represents power consumed vs frequency for a memory-bound workload STREAM, and P_C represents the C1E state. Dashed line represents IdleCap, which affords higher effective CPU frequency than clock-throttling for any power budget.***

workload from Figure 1. If we were to use conventional power capping that utilizes active CPU states, our system would run at a fixed 0.375 GHz (see the datapoint to the left of the arrow). Drawing a *horizontal line* across to the dashed line, we find that the IdleCap scheme could achieve the same power consumption of 157 watts, but increase the effective frequency to over 1 GHz, nearly tripling the processor speed. To achieve this effective frequency, IdleCap would alternate between the C1E state and the 3 GHz state, spending about $r = \frac{1}{3}$ of the time in the 3 GHz state and $(1 - r) = \frac{2}{3}$ of the time in the C1E state. As is evident from Fig. 1, the improvement in effective frequency diminishes as the power cap increases, but is always non-negative.

In alternating between extreme frequency states, we define the *alternation period* to be the time between successive visits to the same frequency state. In the example above, if the alternation period is, say, 1 second, then we would repeatedly spend $\frac{1}{3}$ seconds in the 3 GHz state followed by $\frac{2}{3}$ seconds in the C1E state. As we will later see in Figs. 3(b), 4(b) and 5(b), the exact duration of the alternation period has little effect on the improvement in mean response time afforded by IdleCap.

To a degree that will depend on the specifics of the application, and the extent to which it is CPU-bound, this higher effective frequency will typically translate into lower response time. For example, if the application is 100% CPU-bound, one would expect the improvement in mean response time afforded by IdleCap over clock-throttling to be almost a factor 3 at lower power cap values. In fact, this is what we see for the DAXPY workload in Section 3.

2.3 Non CPU-bound workloads

Fig. 2 illustrates the power-to-frequency relationship for a memory-bound workload, STREAM. The solid line represents clock-throttling whereas the dashed line represents IdleCap. Both Figs. 1 and 2 have an almost linear power-to-frequency relationship for clock-throttling. However, the clock-throttling line in Fig. 2 is *flatter* than the clock-throttling line in Fig. 1. This increases the downwards concavity of clock-throttling for STREAM and in turn makes the horizontal gap between clock-throttling and IdleCap much higher for STREAM than for DAXPY. This flatness can be explained as follows: Since STREAM is a memory-bound workload, it spends a lesser fraction of time using the CPU than DAXPY. In fact STREAM is around 70% memory-bound and 30% CPU-bound. Hence, on increasing the server frequency, the power consumed by STREAM does not rise as quickly as for DAXPY.

Given the higher downward concavity of clock-throttling for STREAM as compared to DAXPY, one would expect to see a higher improvement in response time for STREAM over DAXPY. Specifically, the *horizontal gap* for STREAM is greater, and hence the improvement in effective frequency is also greater than for DAXPY. However, the improvement in mean response time for a memory-bound workload does not scale in the same way as does the improvement in effective server frequency. This is because the response time of a memory-bound workload is made up of time spent at the CPU as well as time spent executing memory operations. On increasing the effective server frequency, only the time spent at the CPU decreases, hence the overall improvement in mean response time is not as great as the improvement in effective server frequency. We verify this experimentally for STREAM in Section 3.

2.4 Analysis

In this Section we present expressions for the effective frequencies given a fixed power cap, P_{cap}, under both clock-throttling and IdleCap. To do this, we invert the linear dependence of power upon frequency depicted in Figs. 1 and 2, yielding:

$$\begin{aligned} f_{\text{CT}}(P_{\text{cap}}) &= \frac{P_{\text{cap}} - P_0}{P_{\max} - P_0} f_{\max} \\ f_{\text{IC}}(P_{\text{cap}}) &= \frac{P_{\text{cap}} - P_C}{P_{\max} - P_C} f_{\max} \end{aligned} \quad (1)$$

where f_{CT} and f_{IC} represent the effective frequencies under clock throttling and IdleCap, P_0 and $P_{\max}$ are the application-dependent parameters as shown in the figures, $f_{\max}$ is the highest server frequency (3 GHz), and P_C is the idle-state power consumption. The fraction of time that should be spent in the high performance state under IdleCap to achieve power P_{cap} is:

$$r(P_{\text{cap}}) = f_{\text{IC}}(P_{\text{cap}})/f_{\max} = \frac{P_{\text{cap}} - P_C}{P_{\max} - P_C}. \quad (2)$$

To understand the impact of power capping upon *mean response time*, one needs to compose the power-to-frequency relationship of Eq. (1) with the frequency-to-response-time relationship. Interestingly, this latter relationship is different for IdleCap than it is for clock throttling.

We define T_H to be the natural duration of a job when the processor operates at $f_{\max}$. This definition applies to jobs of any type. Under *clock throttling*, the processor slows to $f < f_{\max}$, while memory continues to operate at normal speed. If the job is completely compute bound, under a closed-loop setting, one can expect the job completion time to increase to $T_H f_{\max}/f$; otherwise one can only say that it should exceed T_H.

Under *IdleCap*, however, the CPU and memory are treated equally: they are both either fully utilized or fully unutilized. Thus both are slowed by the same factor r. According to Eq. (2), we can expect the job completion time, *for any workload type*, to increase to:

$$E[T^{IdleCap}] \quad = \quad T_H \cdot f_{\max}/f \quad (3)$$

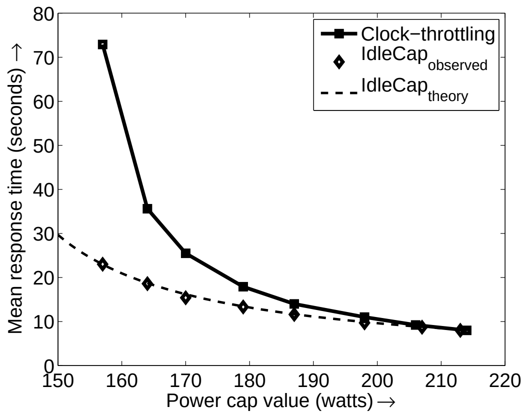
where T_H is obtained via measurements. In the next section, we shall see that our theoretical predictions for the mean response time under IdleCap (given by Eq. (3)) are in excellent agreement with the observed values.

The mean response time of jobs under existing power capping techniques is not entirely predictable, and is very application specific. Thus, a user cannot predict the power cap or processor frequency that she must operate at to achieve a given performance target (say mean response time target). By using IdleCap however, a user can easily predict the processor frequency value, f (using Eq. (3)), required to meet the given mean response time target. Thus, the user can set the value of $r = \frac{f}{f_{max}}$, which gives the fraction of time IdleCap should spend in the highest performance state.

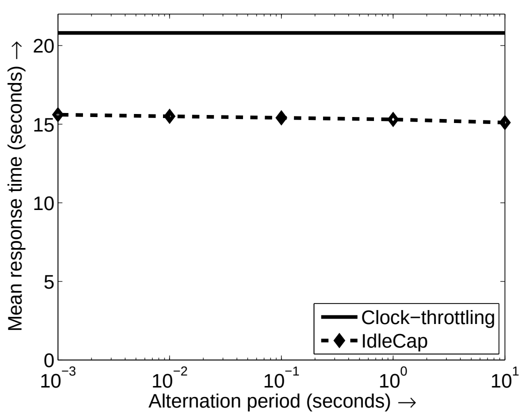
3. EXPERIMENTAL RESULTS

Our experimental setup consists of an IBM BladeCenter HS21 blade server featuring two 3.0 GHz dual-core Intel Woodcrest processors and 4 GB of memory per blade. All our experiments (including Figs. 1 and 2) are conducted on the Linux operating system, and we measure power consumption via IBM's Amester software. We experiment with three workloads: DAXPY and Intel's LINPACK benchmark are CPU-bound, while STREAM is memory-bound.

We use httperf [17] to generate jobs that run sequentially, one right after the other, so that there is always exactly one job in the system (closed-loop). Using httperf, we record the mean response time of the jobs, and use the AMEster software to set the clock-throttling state and measure the server's mean power consumption.



(a)



(b)

Figure 3: ***Figure (a) shows the improvement in mean response time of IdleCap over clock throttling for the DAXPY workload. IdleCap's improvement is as high as a factor of 3.5 at the lowest power caps. The solid line indicates experimental results for clock-throttling. The IdleCap curve indicates both experimental results for IdleCap (shown as diamonds) and the theoretical predictions for IdleCap from Eq. (3) in Section 2.4 (shown as dashed lines). Figure (b) shows the effect of the alternation period on the improvement in mean response time afforded by IdleCap over clock throttling for a power cap of 180 Watts. We see a slight increase in the improvement as we increase the alternation period.***

To compare IdleCap with conventional clock-throttling, we first run the workload under clock-throttling by setting the processor to each of the available throttle states, and measure the power consumption. We then use Eq. (2) to compute the value of r that would result in the same time-averaged power consumption obtained by clock-throttling. We implement IdleCap using a simple script that alternates between the highest performance state and the C1E halt state. We first run a version of IdleCap that alternates with *alternation period* one second, i.e., $r \times 1$ seconds are spent in the 3 GHz state and $(1 - r) \times 1$ seconds are spent in the idle state.

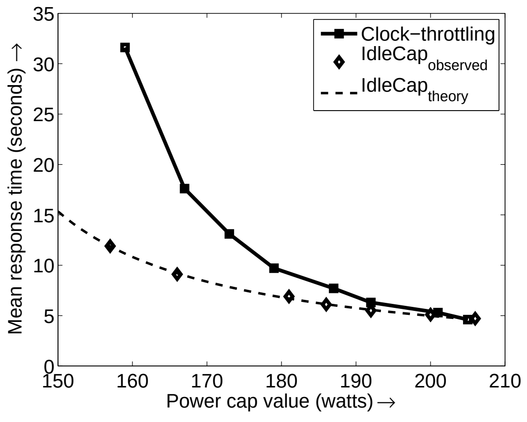
To study the effect of the alternation period on the improvement in mean response time afforded by IdleCap over clock-throttling, we fix an average power consumption value and run IdleCap with different alternation periods. We now discuss our results for the three workloads DAXPY, LINPACK and STREAM.

Fig. 3(a) illustrates the improvement in mean response time afforded by IdleCap over clock-throttling for the 100% CPU-bound workload DAXPY [19]. Observe that the ratio of clock-throttling's mean response time to that of IdleCap ranges from 1 at a power cap of 210 watts to as much as 3.5 for a power cap of 160 watts. The tremendous improvement at low average power levels results from exploiting the power savings of the C1E state for a longer fraction of time. Theoretical predictions for IdleCap are shown using the dashed line, which sits right on top of the experimental results.

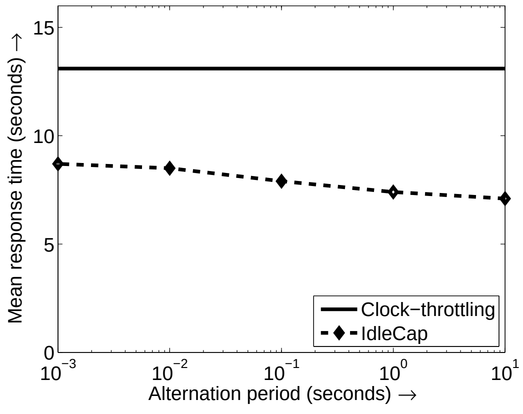
Fig. 3(b) illustrates the effect of alternation period on the improvement in mean response time afforded by IdleCap over clock-throttling for a power budget of $\mathcal{P} = 180$ Watts. We see that the alternation period has no significant effect on the improvement in mean response time afforded by IdleCap. This result holds true in our experiments for other values of $\mathcal{P}$ as well.

Figs. 4 and 5 present comparable results for the LINPACK [10] workload, which runs at about 70% CPU utilization (because the initialization phase runs on 1 core and the solving phase runs on all cores), and the memory-bound STREAM workload [15] respectively. The response time improvements for LINPACK are reduced slightly below those for DAXPY, and those for STREAM are reduced even further, as predicted in Section 2.3. Note the effect of alternation period on the improvement in mean response time afforded by IdleCap over clock-throttling in Figs. 4(b) and 5(b). We see that the percentage improvement drops slightly as we shorten the alternation period. This drop in improvement can be explained as follows: When we shorten the alternation period, there will be more alternations per second between the 3 GHz state and the C1E state. For larger alternation periods, the overhead due to alternating between these states is negligible. However, at alternation periods as low as 0.001 seconds, there are a thousand alternations happening in the system every second. These alternations affect the performance of the workload significantly. However, even at an alternation period of 0.001 seconds, we see that IdleCap provides a 30% improvement in mean response time over clock-throttling for both LINPACK and STREAM. Thus, we can implement IdleCap at granularities as fine as a millisecond, and possibly lower.

Finally, note that the IdleCap performance is entirely predictable across all workloads as discussed in Section 2.4. This is seen by the fact that the dashed lines (the theoretical predictions) in Figs. 3, 4 and 5 are always on top of the IdleCap experimental values (shown as diamonds).



(a)



(b)

Figure 4: ***Figure (a) shows the improvement in mean response time of IdleCap over clock throttling for the LINPACK workload. IdleCap's improvement is as high as a factor of 3.0 at the lowest power caps. The solid line indicates experimental results for clock-throttling. The IdleCap curve indicates both experimental results for IdleCap (shown as diamonds) and the theoretical predictions for IdleCap from Eq. (3) in Section 2.4 (shown as dashed lines). Figure (b) shows the effect of the alternation period on the improvement in mean response time afforded by IdleCap over clock throttling for a power cap of 180 Watts. We see an increase in the improvement as we increase the alternation period.***

4. FINAL REMARKS

In this paper, we have shown that IdleCap is superior to power capping techniques that rely on clock throttling (the t-states) for both CPU and memory-bound workloads. We have also shown that the behavior of IdleCap is entirely predictable via analysis. This is in contrast to existing power capping techniques, where the performance is application dependent. IdleCap is a very versatile algorithm, applying to a wide range of workloads across all processor frequencies and a broad range of alternation periods. Since IdleCap works even for alternation periods as small as 1 millisecond, it can respond well to spikes in power supply or workload variations, by changing its effective server frequency to ensure that the desired power cap is maintained.

One might ask how these results extend to DVFS. Under current DVFS technology, the IdleCap technique is not applicable because the C1E idle state power is not sufficiently low to make the power-to-frequency relationship concave downwards. However, in future processors with even deeper idle processor states (e.g. C3 and C6 [7]), the power consumption for these states is expected to fall below the extrapolated value (P_0) obtained from the power-to-frequency relationship for DVFS, making the power-to-frequency relationship concave downwards. Figure 6 shows the processor power-to-frequency relationship for current-generation Intel mobile processors [11] that have these advanced idle states, assuming worst-case workload. Tomorrow's servers are expected to have these same idle states, resulting in the downwards concave behavior that we've been observing. As shown in Figure 6, IdleCap should again offer improvements over existing power capping throughout the entire range of power caps, with up to a factor of 1.6 improvement in effective frequency. While Figure 6 suggests a linear power-to-frequency curve, one can also imagine a non-linear power-to-frequency curve, especially towards the high frequency end of the spectrum. In such cases, IdleCap's improvement over DVFS would be limited to the lower frequency range. Our future work will evaluate the effectiveness of these deeper (more energy-efficient) C-states to maintain a power cap and also assess how their attendant longer entry and exit times affect the alternation rate in IdleCap.

Finally we note that DVFS does not enable us to achieve the full range of frequencies obtainable under clock-throttling, namely frequencies below 0.75 GHz (see Fig. 6). Hence, clock-throttling is the only mechanism for frequency scaling in the < 0.75 GHz frequency regime. Hence, even in systems that deploy DVFS in the higher frequency range, IdleCap can still be used to improve upon clock-throttling in the lower frequency range; the only difference is that IdleCap would now alternate between the idle state and the lowest DVFS state.

5. REFERENCES

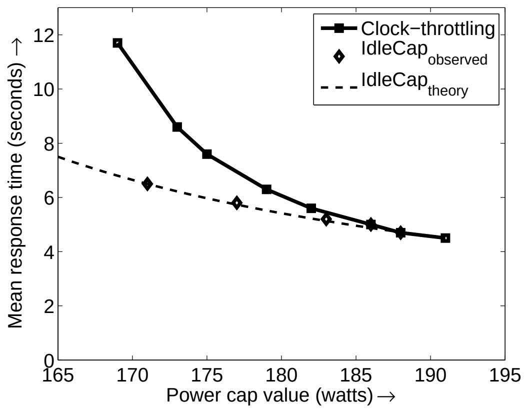
[1] L. Brown, A. Keshavamurthy, D. Li, R. Moore, V. Pallipadi, and L. Yu. ACPI in Linux: architectures, advances and challenges. In *Proceedings of the Linux Symposium*, 2005.

[2] Gartner. Gartner says 50 percent of data centers will have insufficient power and cooling capacity by 2008. `http://www.gartner.com/it/page.jsp?id=499090`, November 2006.

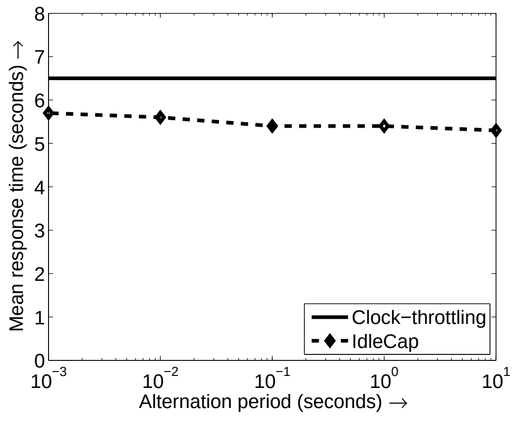
[3] HP. Study: HP data center management solution reduces costs by 34 percent. `http://www.hp.com/hpinfo/newsroom/press/2007/070625xa.html`, June 2007.

[4] HP. HP insight power manager. `http://h18013.www1.hp.com/products/servers/management/ipm/index.html`, Novemeber 2008.

[5] IBM. IBM helps clients "meter" datacenter power usage to help lower energy costs. `http://www-03.ibm.com/press/us/en/pressrelease/19695.wss`, May 2006.



(a)



(b)

Figure 5: ***Figure (a) shows the improvement in mean response time of IdleCap over clock throttling for the STREAM workload. IdleCap's improvement is as high as a factor of 2.0 at the lowest power caps. The solid line indicates experimental results for clock-throttling. The IdleCap curve indicates both experimental results for IdleCap (shown as diamonds) and the theoretical predictions for IdleCap from Eq. (3) in Section 2.4 (shown as dashed lines). Figure (b) shows the effect of the alternation period on the improvement in mean response time afforded by IdleCap over clock throttling for a power cap of 180 Watts. We see an increase in the improvement as we increase the alternation period.***

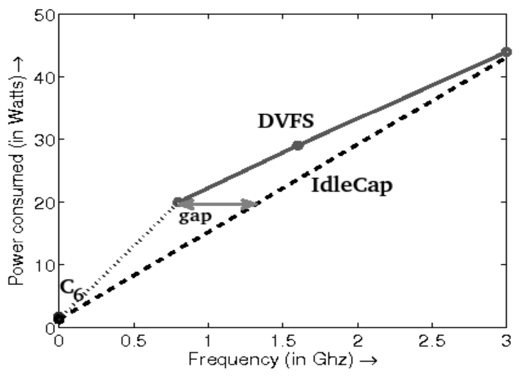


Figure 6: ***Solid line represents processor power vs frequency for DVFS Dual Core Extreme x9100, assuming worst-case workload. Dashed line represents IdleCap, which affords higher effective CPU frequency than DVFS, via the C6 idle state.***

[6] IBM. Going green with IBM systems director active energy manager. `http://www.redbooks.ibm.com/redpieces/pdfs/redp4361.pdf`, August 2008.

[7] Intel Corp. Intel: Nehalem. `http://intel.wingateweb.com/US08/published/sessions/NGMS001/SF08_NGMS001_100t.pdf`.

[8] Intel Corp. Intel news release: Woodcrest. `http://www.intel.com/pressroom/archive/releases/20060626comp.htm`.

[9] Intel Corp. Power and thermal management in the intel core duo processor. `http://download.intel.com/technology/itj/2006/volume10issue02/vol10_art03.pdf`, May 2006.

[10] Intel Corp. Intel Math Kernel Library 10.0 - LINPACK. `http://www.intel.com/cd/software/products/asmo-na/eng/266857.htm`, 2007.

[11] Intel Corp. Intel Core2 Duo Mobile Processor Datasheet: Table 20. `http://download.intel.com/design/mobile/datashts/32012001.pdf`, 2008.

[12] Intel Corp. Intel dynamic power datacenter manager. `http://softwarecommunity.intel.com/articles/eng/3931.htm`, August 2008.

[13] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, November 2007.

[14] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, Jacksonville, FL, USA, 2007. IEEE Computer Society.

[15] J.D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. `http://www.cs.virginia.edu/stream/`.

[16] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: eliminating server idle power. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, 2009.

[17] D. Mosberger and T. Jin. httperf—A Tool for Measuring Web Server Performance. *ACM Sigmetrics: Performance Evaluation Review*, 26(3):31–37, 1998.

[18] R. Nathuji and K. Schwan. Virtual power: Coordinated power management in virtualized enterprise systems. *SOSP*, October 2007.

[19] K. Rajamani, H. Hanson, J. C. Rubio, S. Ghiasi, and F. L. Rawson. Online power and performance estimation for dynamic power management. *IBM Research Report RC-24007*, July 2006.

[20] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. *HPCA*, 2008.

[21] Z. Wang, X. Zhu, C. McCarthy, P. Ranganathan, and V. Talwar. Feedback control algorithms for power management of servers. *Third International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, June 2008.