Constraints And Techniques For Software Power Management In Production Clusters

Arka Bhattacharya

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-110
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-110.html

May 17, 2013

Constraints And Techniques For Software Power Management In Production Clusters

by

Arka Bhattacharya

B.Tech. (IIT Kharagpur) 2010

A thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Engineering - Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor David Culler, Chair
Professor Randy Katz

Spring 2013

The thesis of Arka Bhattacharya is approved.

Chair Date

Date

University of California, Berkeley

Spring 2013

Constraints And Techniques For Software Power Management In Production Clusters

by

Arka Bhattacharya

Abstract

Constraints And Techniques For Software Power Management In Production Clusters

by

Arka Bhattacharya

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor David Culler, Chair

The proliferation of large clusters supporting online web workloads or large compute-intensive jobs has made cluster power management very important [1]. An analysis of utilization traces of production clusters reveal that a majority of them have a scope for (a) under-provisioning of electrical support infrastructure, leading to savings in **capital expenditure**, and (b) energy savings, leading to savings in **operational expenditure**; both with minimal impact on average job performance. Existing software techniques which tackle either of these problems have seen scant adoption because they do not address key problems and constraints relevant in production clusters.

In this thesis, we first investigate possible reductions in cluster power infrastructure provisioning. It is possible that the lower provisioned power level is exceeded due to software behaviors on rare occasions and could cause the entire cluster infrastructure to breach the safety limits. A mechanism to *cap* servers to stay within the provisioned budget is needed, and processor frequency scaling based power capping methods are readily available for this purpose. We show that existing methods, when applied across a large number of servers, are not fast enough to operate correctly under rapid power dynamics observed in data centers. We also show that existing methods when applied to an open system (where demand is independent of service rate) can cause cascading failures in the software service hosted, causing the service performance to fall uncontrollably even when power capping is applied for only a small reduction in power consumption. We discuss the causes

for both these short-comings and point out techniques that can yield a safe, fast, and stable power capping solution.

Next, we address wasteful energy consumption by idle servers in an under-utilized cluster. Despite many clusters having a low average utilization, existing energy management techniques have seen scant adoption because they require modifications to the existing cluster software and network stack, and do not address the reliability concerns that may arise during the course of power-cycling servers in a production cluster. We design, implement and evaluate a defensive energy management system *Hypnos*, which is unobtrusive, efficient, extensible and gracefully handles possible server software and hardware failures.

Professor David Culler
Thesis Committee Chair

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to thank my advisor Prof. David Culler. He always took time out of his busy schedule as the department chair to motivate and guide me through the various challenges involved in deploying power management in production clusters. His constant emphasis on building "stuff that works in the real world" encouraged me to interact with people who operate large clusters on a day-to-day basis, and find out their real concerns about energy management. I also want to thank Prof. Randy Katz for his input and encouragement during our group meetings and presentations.

Next, I want to thank Dr. Aman Kansal, Sriram Sankar, Dr. Sriram Govindan and Kushagra Vaid from Microsoft for giving me an opportunity to intern at Microsoft. At Microsoft, I got the opportunity to meet the teams that managed data centers, and play around with their power and performance data - an invaluable experience.

I also want to thank other members of my group - Andrew Krioukov, Prashanth Mohan and Stephen Dawson-Haggerty for their help in deploying and evaluating the power proportionality system on the Berkeley PSI cluster. I am greatly indebted to Jeff Anderson-Lee, the manager of the PSI cluster, for taking a huge risk by allowing me to deploy power proportionality on a cluster used regularly by about forty graduate students. He was extremely patient through various cluster outages I caused during the initial days of the deployment, and quite often shielded me from the wrath of angry students not being able to run their jobs. I also want to thank Albert Goto for assisting me through this process. Special thanks to Ganesh Ananthanarayanan, Matei Zaharia, Ali Ghodsi and Vyas Sekar for the innumerable passionate discussions on energy and research.

Finally, I want to thank my family, without whose constant support and encouragement I would have been completely lost.

Chapter 1

Introduction

The total cost of ownership of a data center can divided into its *provisioning* costs and its *consumption* costs. *Provisioning* costs include the cost of infrastructure for sourcing, distribution and backup for the peak power capacity (measured in $/kW). The cost of power provisioning and energy consumption in data centers is a very large fraction of the total cost of operating a data center [2]–[4] ranking just next to the cost of the servers themselves. The *consumption* cost is the money paid per unit of energy actually consumed (measured in $/kWh) over the life of a data center. These are significant as well, with the cost of powering a server (including cooling) slowly becoming more than the cost of the server hardware equipment itself [5]. In all, data centers account for 2% of the total U.S energy consumption [6], consuming an estimated 61 billion kilowatt-hours at a cost of $4.5 billion [7]. Thus, power provisioning as well as energy efficiency are important issues in data center management.

1.1 Lowering Provisioned Power Capacity

Provisioned capacity and related costs can be reduced by minimizing the peak power drawn by the data center. A lower capacity saves on expenses in utility connection charges, diesel generators, backup batteries, and power distribution infrastructure within the data center. Lowering capacity demands is also greener because from the power generation standpoint, the cost and environmental

impact for large scale power generation plants such as hydro-electric plants as well as green energy installations such as solar or wind farms, is dominated by the capacity of the plant rather than the actual energy produced. From the utility company perspective, providing peak capacity is expensive due to the operation of 'peaker power plants' which are significantly more expensive to operate and are less environmentally friendly than the base plants. Aside from costs, capacity is now is short supply in dense urban areas, and utilities have started refusing to issue connections to new data centers located in such regions. Reducing the peak power capacity required is hence extremely important.

The need to manage peak power is well understood and most servers ship with mechanisms for power capping [8], [9] that allow limiting the peak consumption to a set threshold. Further capacity waste can be avoided by coordinating the caps across multiple servers. For instance, when servers in one cluster or application are running at lower load, the power left unused could be used by other servers to operate at high power levels than would be allowed by their static cap. Rather than forcing a lower aggregate power level at all times, methods that coordinate the power caps dynamically across multiple servers and applications have been developed [10]–[14].

We identify two reasons why existing power capping methods do not adequately meet the challenge of power capping in data centers. The first is *speed*. We show through real world data center power traces that power demand can change at a rate that is too fast for the existing methods. The second is *stability*. We experimentally show that when hosting online applications, the system may become unstable if power capped. A small reduction in power achieved through existing power capping methods can cause the application latency to increase uncontrollably and may even reduce throughput to zero. We focus on the importance of the two necessary properties - *speed* and *stability*, and propose ways of achieving them and discuss the tradeoffs involved. Our observations are generic, and can be integrated into any power capping algorithm.

1.2 Lowering Energy Consumption

The other component of the cost of ownership of a data center is the operating cost. A way to reduce operating expenditure in a data center is to reduce the amount of non-sever related power

expenditure in a data center. Efforts in reducing the Power Utilization Efficiency (PUE) of large scale data centers[1], has resulted in the typical values reducing from 2 or greater [15], [16], to state-of-the-art facilities having PUEs as low as 1.12 [6], [17], i.e much closer to the ideal value of 1.0.

For data centers with low PUE values, the next challenge is to increase the amount of computing work done per unit energy. Publicly available data shows that data centers are under-utilized for the majority of the time, i.e a lot of servers do no work. One of the key tenets of system design is: in making a design trade-off, favor the frequent case over the infrequent case. The systems community, for long, has focused on optimizing for performance — making the common case fast [18]. But the common case for many of the servers in such clusters is *doing nothing*; that is, being idle. This common case of doing nothing consumes power because servers are not power proportional (i.e power consumption is not directly proportional to utilization [19], [20]). If we are to design for the common case from a power perspective, we need to "do nothing well". This requires understanding cluster idleness as deeply as we understand performance, and harnessing that idleness to obtain energy savings.

Building power proportional clusters comprising of servers which are not power-proportional is a well-studied problem ([21]–[26]), and has been shown to provide large energy savings. The underlying concept in each of the techniques involves coalescing the available work onto the minimum number of servers and powering down the remaining.

In spite of the large body of existing work, cluster power proportionality has seen scant deployment. Some of the main reasons are that most existing techniques require modifications to already complex cluster frameworks and configuration scripts. Also, some prior work does not deal with the possibility of software and hardware failures which can arise when servers are regularly power cycled [27], making these solutions non-viable for deployment on production clusters. We mitigate these concerns through *Hypnos*, a defensive meta-system, which is unobtrusive, extensible and fault-tolerant.

[1]PUE is the ratio of total data center consumption to that consumed by the computing equipment

1.3 Contributions

We describe our contributions on two fronts - first in reducing the capital cost of a cluster through safe and stable powercapping; and second, in reducing the operating cost of a data center through unobtrusive and fault-tolerant power management.

Power Capping: We quantify the benefit of using power capping to lower power provisioning costs in data centers through the analysis of a real world data center power trace. From the same trace, we characterize the rates at which power changes in a data center. We make a case for one-step power controllers by showing that existing closed-loop techniques for coordinated power capping across a large number of servers may not be fast enough to handle data center power dynamics. We then show that existing power capping techniques do not explicitly shape demand, and can lead to instability and unexpected failures in online applications.Finally, we present admission control as a power capping knob. We demonstrate that admission control integrated with existing power capping techniques can achieve desirable stability characteristics, and evaluate the trade-offs involved.

Energy Savings: We develop and evaluate a defensive meta-system *Hypnos* which enables power-proportionality unobtrusively for a production cluster running HPC workloads.[2]. Hypnos has three main design principles - unobtrusiveness, fault tolerance and extensibility. Hypnos requires no modification to a cluster, ensuring easy deployment or removal. *Hypnos* infers and gracefully handles various software and hardware faults. The modular design of *Hypnos* enables easy adaptation to different HPC frameworks by simply changing the framework specific parser. We evaluate Hypnos by deploying it on a production cluster running Torque [28] in an academic environment. Hypnos was able to achieve a 36% reduction in energy consumption while circumventing over 1500 network and software faults over a 21-day deployment.

1.4 Roadmap

In the rest of the thesis, we first address the stability and safety requirements for an effective power capping in Chapter 2. Next, we describe in detail the design and evaluation of *Hypnos* to achieve

[2]Hypnos only enables power proportionality, and does not enable powercapping

cluster power proportionality in Chapter 3. Finally, we mention relevant related research in the fields of power capping and power proportionality and conclude in Chapter 4.

Chapter 2

Safe and Stable Power Capping

In this chapter, we will delve into the constraints that should be present in any data center power capping algorithm in a production environment. We first describe the reasons why power capping is essential in a data center, and then lay out the *Safety* and *Stability* criteria, which is overlooked in existing techniques.

2.1 Background : Power Costs and Capping Potential

Most new servers ship with power capping mechanisms. System management software, such as Windows Power Budgeting Infrastructure, IBM Systems Director Active Energy Manager, HP Insight Control Power Managment v.2.0, Intel Node Manager, and Dell OpenManage Server Administrator, provide APIs and utilities to take advantage of the capping mechanisms. In this section we discuss why power capping has become a significant feature for data centers.

2.1.1 Power Provisioning Costs

The designed peak power consumption of a data center impacts both the capital expense of provisioning that capacity as well as the operating expense of paying for the peak since there is often a charge for peak usage in addition to that for energy consumed.

The capital expense (cap-ex) includes power distribution infrastructure as well as the cooling

infrastructure to pump out the heat generated from that power, both of which depend directly on the peak capacity provisioned. The cap-ex varies from $10 to $25 per Watt of power provisioned [4]. For example, a 10MW data center spends about $100-250 million in power and cooling infrastructure. Since the power infrastructure lasts longer than the servers, in order to compare this cost as a fraction of the data center expense, we can normalize all costs over the respective lifespans. Amortizing cap-ex over the life of the data center (12-15 years [3], [4]), server costs over the typical server refresh cycles (3-4 years), and other operating expenses at the rates paid, the cap-ex is *over a third* of the overall data center expenses [3], [29]. This huge cost is primarily attributable to the expensive high-wattage electrical equipment, such as UPS batteries, diesel generators, and transformers, and is further exacerbated by the redundancy requirement mandated by data center availability stipulations.

The peak power use affects operating expenses (op-ex) as well. In addition to paying a per unit energy cost (typically quoted in $/kWh), there is an additional fee for the peak capacity drawn, even if that peak is used extremely rarely. Based on current utility tariffs [30] for both average and peak power, the peak consumption can contribute to as much as 40% of the utility bill [31]. Utility companies may also impose severe financial penalties for exceeding contracted peak power limits.

The key implication is that reducing the peak capacity required for a data center, and adhering to it, is highly beneficial.

2.1.2 Lower Cost Through Capping

Power capping can help manage peak power capacity in several ways. We describe some of the most common reasons to use it below.

2.1.2.1 Provisioning Lower Than Observed Peak

Probably the most widely deployed use case for power capping is to ensure safety when power is provisioned for the actual data center power consumption rather than based on server *nameplate ratings*. *Nameplate ratings* on servers denotes its maximum possible power consumption, computed as the sum of maximum power consumption of all the server sub-components and a conservative

safety margin.The name-plate rating on servers is typically much higher than the server's actual consumption. Since no workload actually exercises every server subcomponent at its peak rated power, the name plate power is not reached in practice. Data center designers thus provision for the *observed peak* on every server. The observed peak is the maximum power consumption measured on a server when running the hosted application at the highest request rate supported by the server. This observed peak can be exceeded after deployment due to software changes or events such as server reboots that may consume more than the previously measured peak power. Server level power caps can be used to ensure that the provisioned capacity is never exceeded and protect the circuits and power distribution equipment.

Server level caps do not eliminate waste completely. Setting the cap at each server to its observed peak requires provisioning the data center for the *sum of the peaks*, results in wasted capacity since not all servers operate at the peak simultaneously. Instead, it is more efficient to provision for the *peak of the sum* of server power consumptions, or equivalently, the estimated peak power usage of the entire data center. The estimate is based on previously measured data and may sometimes be exceeded. Thus a cap must be enforced at the data center level. Here, the server level caps will change dynamically with workloads. For instance, a server consuming a large amount of power need not be capped when some other server has left its power unused. However the former server may have to be capped when the other server starts using its fair share. Coordinated power capping systems [10]–[14] can be used for this.

Additionally, even the observed peak is only reached rarely. To avoid provisioning for capacity that will be left unused most of the time, data centers may provision for the 99-th percentile of the peak power. Capping would be required for 1% of the time, which may be an acceptable hit on performance in relation to cost savings. If the difference in magnitude of power consumed at the peak and 99-th percentile is high, the savings can be significant. To quantify these savings, we present power consumption data from a section comprising of several thousand servers in one of Microsoft's commercial data centers that host online applications serving millions of users, including indexing and email workloads. The solid line in Figure 2.1 shows the distribution of power usage, normalized with respect to the peak consumption. If the 99-th percentile of the observed peak is provisioned for,

the savings in power capacity can be over 10% of the data center peak. Capacity reduction directly maps to cost reductions.

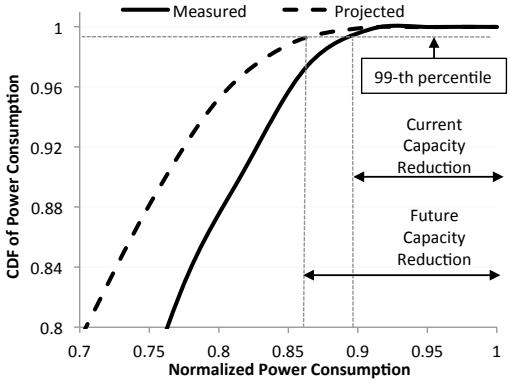


Figure 2.1. Cumulative distribution function (CDF) of power consumption for a cluster of several thousand servers in one of Microsoft's commercial data centers. Future capacity reduction refers to the power consumed by the same workload if hosted on emerging technology based servers.

Trends in server technology indicate that the margin for savings will increase further. Power characteristics of newer servers accentuate the difference between the peak and typical power (power consumed by a server under average load) usage because of their lower idle power consumption. Power measurement for an advanced development server at different CPU utilizations shows only 35% of peak consumption at idle, much lower than the over 50% measured in current generation servers. Using processor utilizations from the real world servers, we project the power usage of the same workloads on the future generation servers assuming that power scales with processor utilization [32] (the dashed curve in Figure 2.1). The present day data and technology trends both indicate a significant margin for savings.

2.1.2.2 UPS Charging

Large data centers use battery backups, also referred to as Uninterrupted Power Supplies (UPSs). UPSs provide a few minutes of power during which time the diesel generators may be powered up. After power is restored, the UPS consumes power to re-charge the batteries. This implies that the power capacity provisioned for a data center should not only provide for the servers and cooling equipment but also include an additional margin for battery charging. This additional capacity is

almost always left unused since power failures are relatively rare. Even when power failures do happen, they may not occur at the time when data center power consumption is at its peak.

The capacity wasted due to reservation for battery charging can be avoided if the batteries are charged from the allocated server power capacity itself. Should the servers happen to be using their full capacity at recharging time, power capping is needed to reduce the server power consumption by a small amount and free up capacity for recharging batteries at a reasonable rate. Since power failures are rare, the performance impact of this capping is acceptable for many applications. Any data center that uses a battery backup can use power capping to reduce the provisioned power capacity.

2.1.2.3 Total Capital Expenses

Many power management methods are available to reduce server power consumption by turning servers off or using low power modes when unused. Using less energy however does not reduce the cost of the power infrastructure or the servers themselves. The amortized cost of the servers and power infrastructure can be minimized if the servers are kept fully utilized [33]. Workload consolidation can help achieve this. Suppose a data center is designed for a given high priority application and both servers and power are provisioned for the peak usage of that application. The peak workload is served only for a fraction of the day and capacity is left unused at other times. During those times, the infrastructure can be used to host low priority applications.

In this case capping is required on power, as well as other computational resources, at all times to ensure that the low priority application is capped to use only the resources left unused by the high priority applications and up to a level that does not cause performance interference with the high priority tasks. Since power is capped by throttling the computational resources themselves, the implementation may not require an additional control knob for power. However, settings on the throttling knobs should ensure that all resource limits and the power limit are satisfied. The end result is that in situations where low priority workloads are available, power capping can be used in conjunction with resource throttling to lower both power and server capacity requirements.

2.1.2.4 Dynamic Power Availability

There are several situations where power availability changes with time. For instance, if demand response pricing is offered, the data center may wish to reduce its power consumption during peak price hours. If the data center is powered wholly or partly through renewable energy sources such as solar or wind power, the available power capacity will change over time. Power capacity may fall due to brown-outs [34]. In this situation too, a power capping method is required to track the available power capacity.

The above discussion shows that power capping can help save significant cost for data centers. However, existing power capping methods suffer from speed and stability limitations in certain practical situations. In the next sections we quantitatively investigate these issues and discuss techniques to enhance the existing methods for providing a complete solution.

2.2 Speed: Power Capping Latency

The actuation latency of power capping mechanisms is an important consideration. Server level power capping mechanisms, typically implemented in server motherboard firmware, change the processor frequency using dynamic voltage and frequency scaling (DVFS) until the power consumption falls below the desired level [8]. These local methods can operate very fast, typically capping power within a few milliseconds. However, capping speed can become an issue for coordinated power capping methods that dynamically adjust server caps across thousands or tens of thousands of servers [12]–[14]. To understand this issue in depth, we first study the temporal characteristics of data center power variations from the trace analyzed in Figure 2.1. We then quantify the required actuation latencies for a power capping mechanism, and compare it to the state-of-the-art.

2.2.1 Data Center Power Dynamics

Data center power consumption varies due to workload dynamics such as changes in the volume of requests served, resource intensive activities such as data backup or index updates initiated by the application, background OS tasks such as a disk scrubs or virus scans, or other issues such as

simultaneous server restarts. We study the data center power trace previously shown in Figure 2.1 to quantify the rate of change of power.

Since capping is performed near peak power levels, only power increases that occur near peak usage matter for capping; power changes that are well below the peak, however fast, are not a concern. So we consider power increases that happen when power consumption is greater than the 95th percentile of the peak. We measure the rate of power increase, or *slope*, as the increase in normalized power consumption (if over the 95th percentile) during a 10 second window.

Figure 2.2 shows the CDF of the *slope*, normalized to the peak power consumption of the cluster. For most 10 second windows, power increases are moderate (less than 2% of the peak cluster power consumption). However, there exists power increases as high as 7% of the peak consumption over a 10 second window. To ensure protection and safety of electrical circuits during such extreme power surges, the power capping mechanism must be agile enough to reduce power consumption within a few seconds.

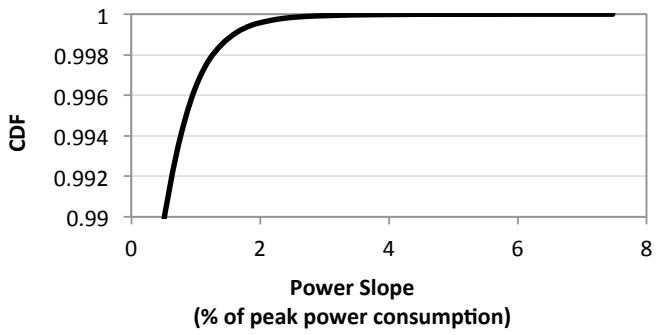


Figure 2.2. Cumulative distribution function (CDF) of power slope [increase in power conumption of the cluster over a 10 second window]. The slope is normalized to the peak power consumed by the cluster during the period of the study.

2.2.2 Power Control Latency

This section experimentally investigates the limits on how fast a power capping mechanism can throttle multiple servers using DVFS. The experiments were performed on three servers with different processors: Intel Xeon L5520 (frequency 2.27GHz, 4 cores), Intel Xeon L5640 (frequency 2.27GHz, dual socket, 12 cores with hyper-threading), and an AMD Opteron 2373EE (frequency

2.10GHz, 8 cores with hyper-threading). All servers under test were running Windows Server 2008 R2. Power was measured at fine time granularity using an Agilent 34411A digital multimeter placed in series with the server. The multimeter recorded direct current values at a frequency of 1000Hz, and root mean square was computed over discrete 20ms intervals where one interval corresponds to 1 cycle of the 50Hz AC power. Since in a practical power capping situation, the cap will likely be enforced when the servers are busy, in our experiment the servers were kept close to 100% utilization by running a multi-threaded synthetic workload. This kept the server near its peak consumption level from where power could be reduced using power capping APIs.

To estimate the fastest speed at which a data center power capping mechanism can operate, the latency to be considered is the total delay in determining the desired total data center power level, dividing it up into individual server power levels, sending the command to each server, the server executing the power setting command via the relevant API, and the actual power change taking effect (Figure 2.3). Since we are only interested in the lower limit on latency, we ignore the computational delays in computing the caps. A central power controller is assumed to avoid additional delays due to hierarchical architectures. In the following sections we investigate each of these latency components.

2.2.3 Network Latency in a data center

Table 2.1 shows the network latency of sending a packet between the controller (hosted within the data center network) and the power capping service at a server, for varying network distances. This data was obtained using a Microsoft data center management tool, `PingMesh`, that allows measuring ICMP ping latencies across a data center network. The data shows that the average packet delay on a network is less than a millisecond. This latency component is hence not likely to be a concern for coordinated capping.

2.2.4 System Latency

Once a DVFS setting from the controller reaches a server, it is applied by calling the relevant APIs. In this experiment, the frequency was decreased from the maximum to minimum to obtain the

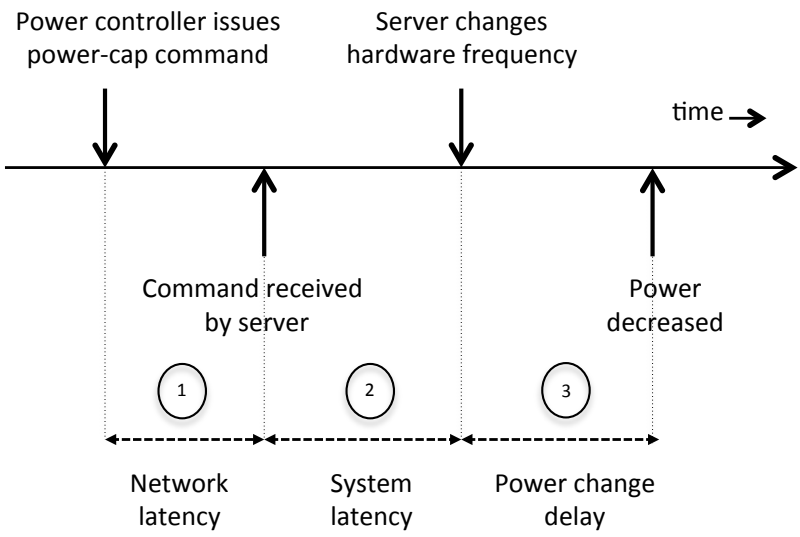


Figure 2.3. Timeline showing the smallest set of latency components for a coordinated power capping solution. Additional latency components may get added when the cap is enforced in a hierarchical manner such as in [12], [14].

Table 2.1. Network latencies in a data center.

Sender and Receiver placement	**No. of samples**	**Avg (ms)**	**Std. dev (ms)**
Within same rack	21	0.331	0.098
Within same aggregation switch	32	0.342	0.030
Under different aggr. switches	61	0.329	0.032

highest resolution power change for measurement of latency. Low level frequency APIs offered through `powerprof.dll` in the Windows OS were used to avoid as much of the software stack delays as possible. The threads for applying and reading the frequency setting were set to higher priority so as to not be delayed due to the server workload. The latency incurred for changing the frequency ranged between 10-50ms for multiple runs on the different servers.

2.2.5 Power Change Delay

After the processor frequency changes there is an additional delay before the power drops to the new level at the wall outlet, due to factors such as capacitance in the server and the power supply circuits. This effect requires a fine time granularity power measurement. Figure 2.4 shows a sample power reading plot measured using the Agilent multimeter. The latency was found to be between 100ms and 300ms, for a frequency change from the maximum to the minimum, across multiple measurements over the three servers. The minimum latency was observed when the frequency was changed between two adjacent DVFS levels requiring a smaller change in power. The smallest latency across all adjacent frequency levels was 60ms. These measurements are similar to the fast capping latency of 125ms reported in commercial product data-sheets [35].

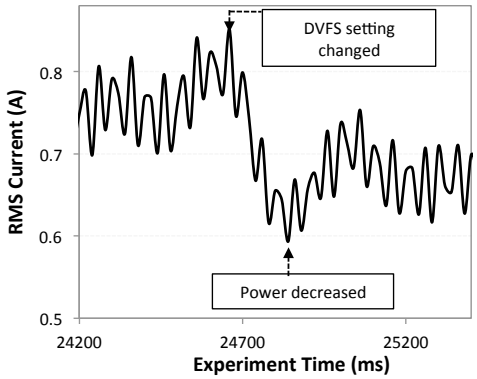


Figure 2.4. Typical latency between the hardware frequency change and power reduction. The current readings are rms values over discrete 20ms windows. The power decrease latency in this experiment was approximately 200ms.

2.2.6 Total Delay

A summary of the latency results is provided in Table 2.2 and totals to approximately 110ms to 350ms. This implies that for a feedback based controller, it takes approximately 110ms to 350ms for one iteration of a control loop. Much of this delay is coming from the power change at the server itself rather than the computational overhead or network delay of the coordinated capping algorithm.

Table 2.2. Summary of actuation latencies for power capping

Latency type	**Approx. Latency**
Network	<1ms
OS	10-50ms
Wall power change	100-300ms
Total	110-350ms

Implications: An important implication of the above measurements is that a feedback controller using multiple iterations can incur several seconds of delay. Many controllers use a hierarchy to scale to a large number of servers or to logically separate the power division among multiple applications in a data center [12], [14]. When feedback loops operate at multiple levels in the hierarchy, control theoretic stability conditions require that the lower layer control loop must converge before an upper layer loop can move on to the next iteration. Suppose the actuation latency is denoted as l (where $l \approx 110ms - 350ms$ from Table 2.2) and the number of iterations required for convergence at the $i-th$ layer in the control hierarchy is n_i, then the total latency of the capping mechanism using N layers in the hierarchy becomes:

$$l_{total} = l * \prod_{i=1}^{N} n_i$$

As an example, considering the two layer hierarchy ($N = 2$) with $n_1 = 6$ and $n_2 = 16$ used in [12], and plugging in the measured l value in the above equation, we would get a control latency of 10.56s to 33.6s. For the three layer hierarchy used in [14] and similar number of convergence iterations required, the latency will be even higher. While this latency is not a concern for adapting to the slow changes in workload levels that only cause the power to change every few minutes, these latencies are not acceptable for the fast power changes observed in real data centers (Figure 2.2).

Some of the power distribution components in the data center can handle capacity overages for a few seconds or even minutes [36], [37]. However, when power is changing at a rapid rate, the feedback based controllers cannot meet their stability conditions. The dynamics of the system being controlled must be slower than the convergence time of the controller. The requirement for stability implies that power should not change beyond measurement tolerance within the 10.56s or

33.6s control period. That however is not true since the power can change by as much as 7% of the data center peak power within just 10s, in real data centers (Figure 2.2).

2.2.7 Summary

The latency analysis above implies that feedback based controllers using multiple iterations are not fast enough to operate safely under the data center power dynamics. The design implication for power capping methods is that the system may not have time to iteratively refine its power setting after observing a capacity violation.

Observation 1:

A safe power capping system should use a single step actuation to apply a power cap setting, such as using DVFS, that will conservatively bring the power down to well below the allowed limit (say, the lowest DVFS setting)

The conservative setting is needed to avoid unsafe operation in the presence of model errors. Once power has been quickly reduced to a safe limit, feedback based controllers can be employed to iteratively and gradually increase power to the maximum allowed limit to operate at the best performance feasible within the available power capacity.

2.3 Stability: Application Performance with DVFS Based Capping

It is well known that for system stability, the incoming *request rate* should be lower than the sustained *service rate* across the multiple servers hosting a given application [38]. This requirement is often the basis of capacity planning, such as for determining the number of servers required. The service rate is experimentally measured for a variety of requests served by the hosted online application and the number of servers is chosen to match or exceed the maximum expected request rate[1]. As request rate increases, more servers are added to the deployment. [2]

[1] Power management methods may be employed to turn off or re-allocate unused servers when request rate is lower than the maximum rate that can be served.

[2] The terms *request rate*, *demand* and *workload* have been used interchangeably in the subsequent sections

Under normal conditions, the service rate exceeds the request rate. However, whenever power capping is performed, power consumption of some server resource must be scaled down. Typically the processor power is scaled down using DVFS for practical reasons, though in principle, one could scale down the number of servers or some other resource as well. Regardless of the mechanism used to reduce power, engaging it reduces the service rate. The incoming request rate may or may not change when service rate is reduced. If the system is *closed*, where each user submits a new request only after the previous response is received, the request rate will fall to match the service rate. Batch processing systems such as Map-Reduce, HPC workloads, or workloads such as mail where a user issues a new request only after a previous request is completed can be closely approximated as a closed system. However, if the system is *open*, where the request rate is not directly affected by the service rate, a decrease in service rate due to power capping may not lead to a equivalent decrease in the request rate. Most web based online applications, such as web search, can be approximated as open systems since the requests are coming from a large number of users and new requests may come from new users who have not yet experienced the reduced service rate. Even users experiencing reduced service rate may not stop submitting new requests. Delays may even lead to rapid abort and retry.

2.3.1 Open-loop systems

Capping is enforced primarily when the system is at high power consumption. This happens when serving close to the peak demand that the system can support. Hence, the reduced service rate after capping is very likely to be lower than the demand at that time. Queuing theory says that response time shoots up uncontrollably in this situation in an open loop application. We experimentally demonstrate this in an open system.

Experiment: We use a web server hosting Wikipedia pages using MediaWiki[3], serving a 14 GB copy of the English version of Wikipedia, as an example of an open loop system. The front end is hosted on an Intel Nehalem Xeon X5550 based server (16 cores, 48GB RAM) running Apache/2.2.14 and PHP 5.3.5. The database uses mysql 14.14 hosted on another similar server. Both servers run Ubuntu 10.04 as the Operating System. HTTP requests for Apache are generated

[3] `http://www.mediawiki.org/`

by a workload generator using the `libevent` library, which has support for generating thread-safe HTTP requests. Seven workload generators were used to ensure that the workload generators themselves were not a bottleneck. To avoid crippling disk latencies, and to ensure that all requests were served out of the back-end cache the workload generators repeatedly request the same wiki page. All the workload generator servers have their system time synchronized and log performance (throughput and latency) in separate files that are then aggregated.

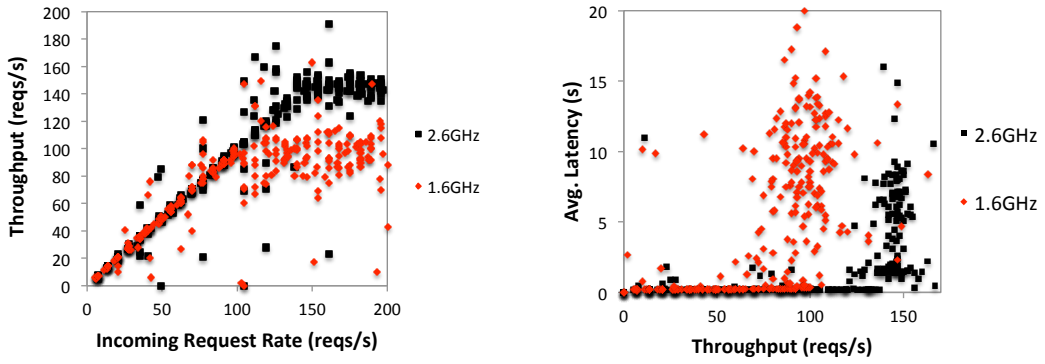
We ramp up the incoming request rate from 0 to 200 requests per second, to study the variation of delivered application throughput and latency at two different processor frequencies - 2.6GHz and 1.6GHz.

Observations: From Figure 2.5(a), we find that the maximum application throughput remains equal to the incoming request rate at approximately 130 and 90 requests for 2.6GHz and 1.6GHz respectively. For larger magnitudes of incoming request rate, application throughput becomes unpredictable. We thus conclude that within the above-mentioned frequencies, the application remains stable. From Figure 2.5(b), we find that the average latency within the stable range was 0.2 seconds. Incoming request rates beyond the stable region, also led to the request latency increase orders of magnitude (touching a maximum of 20 seconds). This experiment clearly shows that driving an open system with a request rate higher than its service rate results is degraded performance.

Experiment : Next, we operate the system at a throughput of 120 requests/s, which is below the maximum supported service rate at 2.6GHz. We conduct two experiments - one in which we keep the server frequency at 2.6GHz, and one in which we power cap the server to 1.6GHz using DVFS, an existing power capping mechanism, to reduce power consumption.

Observations: Figure 2.6 shows the impact on performance using DVFS based power capping. The gray curve shows the normal operation at 2.6GHz. The black curve shows the operation when the server is operated at a lower frequency but the incoming request rate is not changed. Throughput falls since the computational resource available is lowered. However, latency starts to increase uncontrollably to much higher values than the initial 0.2s, even though the input request rate is constant throughout (at 120 requests/s).

Performance plummets by orders of magnitude in a relatively short time when operating at the



(a) Application throughput against different incoming request loads

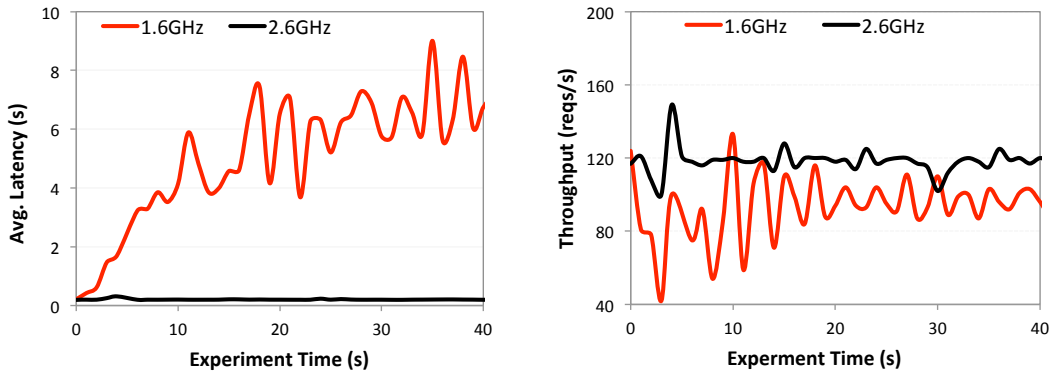
(b) Increase in application latency as response rate increases

Figure 2.5. Server throughput and latency under different incoming request rates in an open-loop system.

lower frequency. This is expected since several undesirable effects start to manifest in this situation. First, any buffers in the system, such as in the network stack or the web server's application queue for incoming requests will get filled up and they will unnecessarily add to the latency without yielding any advantage on throughput [39]. Second, requests not served will be re-attempted, increasing the total number of requests coming into the system. Since some of the requests served will not be fresh requests but re-attempted ones, the total request service latency will increase. Even with a small reduction in service rate, the number of dropped requests will start piling up and the average latency will continue to rise, leading to the plummeting performance observed. Third, if semantically, each user activity consists of multiple requests (such as a accessing a web page may consist of accessing multiple embedded image and resource URLs from the web server), since some of the requests may have been dropped from each semantic activity, no user activity will have been served. This implies that a small reduction in power can actually render a system *unstable*.

2.3.2 Closed loop systems

In a closed loop system, a reduction in service rate leads to a commensurate reduction in request rate because a new request is only issued when the previous request terminates. While average ap-



(a) Application Average Response Time

(b) Application Throughput

Figure 2.6. Effect of DVFS based power capping on throughput and latency in the experimental Wikipedia server.

plication latency does rise due to slower service rates, a closed loop application does not experience the same cascading failure as shown in Section 2.3.1.

Experiment: As an example of a closed system, we use an installation of Stocktrader 4.0[4] which mimics an online stock-trading website similar to IBM Websphere Trade 6.1. The workload generators were set to behave as if in a closed system, where they submit the subsequent request only after the current request is served. We use processor capping (reducing the running time of the application) to reduce the service rate and power consumption of Stocktrader. An initial load of 2000 requests per second is applied, following which processor caps were used to throttle the application. The processor cap is reduced from 100% to 10% in decrements of 10% every 60 seconds.

Observations: Even though the application throughput decreases on the application of processor caps, Figure 2.7the latency does not rise uncontrollably. A lower service rate from the applciation server forces a reduction in the request rate generated by the workload generators, resulting in no queue buildup or packet drops as seen in Section 2.3.1. Thus, application stability is not compromised during power capping of closed-loop systems.

[4] `http://msdn.microsoft.com/en-us/netframework/bb499684.aspx`

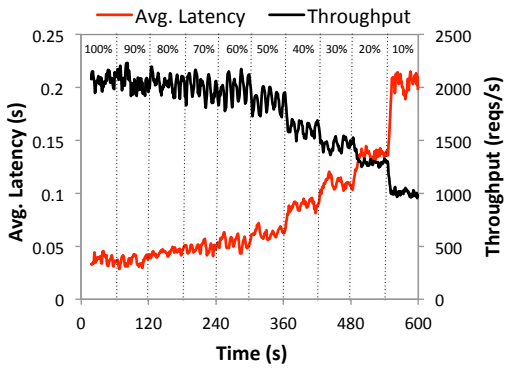


Figure 2.7. Variation of throughput and latency when different processor caps are applied to StockTrader. The processor cap was reduced by 10% every 60 seconds, as is indicated by the dotted vertical lines.

Observation 2:

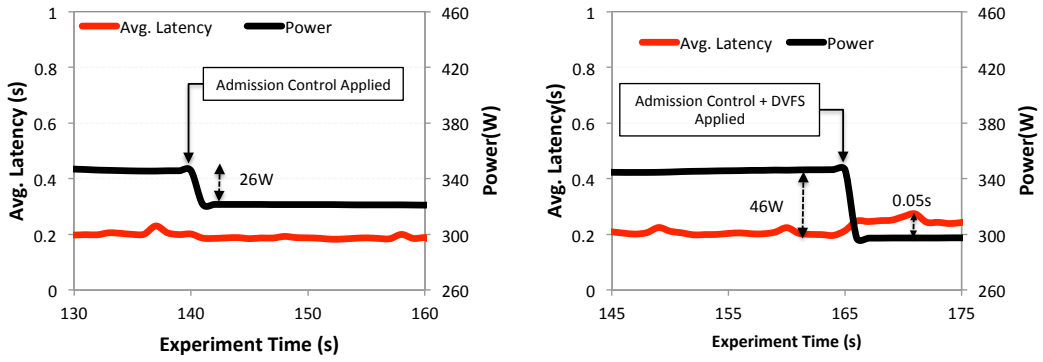
A stable power capping system when reducing the service capacity of a server running an open-loop application through DVFS, should be able to implement a commensurate reduction in incoming application demand (through admission control)

2.4 Stable Power Capping with Admission Control in Open Loop Systems

2.4.1 Admission Control and Power

Power capping reduces the service rate, which can make a system unstable. To maintain stability, the input request rate should also be reduced within a modest time window, and admission control is one technique to achieve that. This would result in some users receiving a "request failed" message or a long wait before service, but the system will be able to serve the remaining workload within acceptable performance bounds.

If admission control is applied, the amount of work performed, and correspondingly the amount of computational resource used, is reduced. This implicitly reduces the power consumption since the processor has more idle cycles that it can spend in lower power sleep states. Intuitively, this suggests



(a) Power reduction and average request service latency before and after admission control is applied.

(b) Variation of latency and power with time when admission control is used along with DVFS. The small increase in latency is discussed in Section 2.4.3.

Figure 2.8. Power and average latency variation when using only admission control, and admission control+DVFS

that admission control can be used as an alternative power capping mechanism. We experimentally verify that this intuition is correct. However, there are practical issues that prevent admission control from directly replacing DVFS based or other existing power capping mechanisms.

Experiment: Using the same experimental testbed as used in Section 2.3, we measure the power reduction provided using admission control. We implemented admission control using the `iptables` utility and selectively filter out requests from some of the workload generators (based on IP address) to reduce the incoming request rate to the Wikipedia server[5].

As in Figure 2.6, suppose the server is originally operating at 120 requests/s (at processor frequency 2.6GHz). Suppose the desired power reduction can be achieved using DVFS by lowering the processor frequency to 1.6GHz. The throughput sustained at this lower frequency is measured to be 85 requests/s and the reduction in power is 46W. Keeping the input request rate at 120 requests/s, we enforce admission control to allow only 85 requests/s to be presented to the server. Figure 2.8(a) shows the impact on power when admission control is applied at the time tick of 140s (approx). As intuitively expected, admission control does reduce power and can be used as a power capping

[5]In practice, admission control may be implemented by the application or in the load balancers, among other options. Our purpose in this thesis is only to study the effect of admission control on power and performance, and the above implementation suffices.

mechanism. However, the reduction in power is only 26W (instead of 46W that was achieved using DVFS for the same reduction in throughput).

2.4.2 Practical Issues with Admission Control

Power Efficiency: To investigate the power difference further, we measure the power consumption at varying throughput levels at two different DVFS frequency settings. Figure 2.9 shows the power measurements. The key take-away is that the same throughput can be served at a lower power level using the lower frequency, though the peak throughput that can be served is lower at the lower frequency. A difference of 20W is apparent. This is because the lower frequency is more energy efficient. As is known from DVFS design, processor power increases with the cube of frequency and even the total server power has been measured to increase super-linearly with frequency [38]. Since the number of processor cycles available for computation increases only linearly with frequency, this makes lower frequencies more energy efficient at a given throughput.

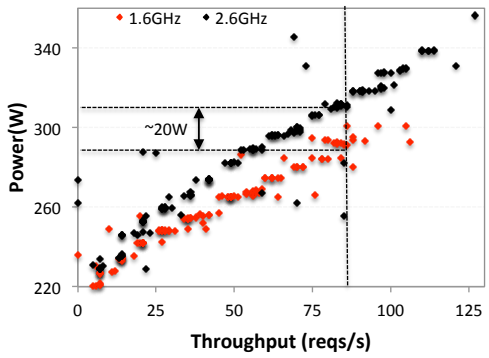


Figure 2.9. Power vs throughput in the stable region of the Wikipedia front end server at 2.6GHz and 1.6GHz. Note the extra reduction in power available by using DVFS in addition to admission control

The observation above indicates that while admission control is required for stability, DVFS is more efficient from a power perspective. Hence a practical power capping system must use DVFS in combination with admission control to achieve stability without sacrificing efficiency. Figure 2.8(b) shows the effect on power when both mechanisms are applied simultaneously (around time tick 163s in the figure). The throughput achieved (not shown) is the same in both Figures 2.8(a) and 2.8(b) but

the power is capped by a greater amount in Figure 2.8(b). As technology improves and idle power consumption falls further, the above power difference may be reduced since the higher frequency state with more idle cycles will likely become more power efficient as well.

Delay: Another practical issue that requires DVFS is the effect of queuing delays. If the application has a large buffer for incoming requests, then a large number of requests will be served from that queue. Admission control will reduce the incoming request rate but the service rate in the servers may remain high while the queues are being emptied, leading to a delay before the power is actually reduced. This is a concern when speed of the capping mechanism is important.

Safety: Admission control reduces the workload offered to the server but does not force the server power to be lowered. While power is expected to fall with reduced workload, in some cases it may not, such as when the server is running a background virus scan or operating system update. With DVFS all computations related to the workload or background tasks will be throttled down simultaneously to reduce power.

2.4.3 Application Latency

Another metric worth comparing between Figures 2.8(a) and 2.8(b) is the application performance in terms of latency. While throughput reduction is the same and stability is ensured in both cases, the latency shows a small increase when DVFS and admission control are combined. Suppose servicing each request requires an average of n_r processor cycles. Then the latency component attributable to the processor, denoted l_{cpu} can be computed as $l_{cpu} = n_r/f_i$ where f_i is the processor frequency in use at the $i-$th DVFS setting. When DVFS is used to reduce the frequency from a higher value f_0 to a lower value f_1, clearly l_{cpu} will rise. Other latency components such as the network round trip delay, queuing delay, and the latency of accessing the backend storage are not significantly affected by DVFS and the increase in l_{cpu} shows up as a small increase in overall application layer latency.

2.4.4 Summary

From the above analysis , we conclude that using admission control alone leads to a smaller power reduction, higher possible actuation delay, and the possibility of unforseen software events which might cause a power spike.

> **Observation 3:**
> *Admission control, while necessary for application stability, should be used in conjunction with DVFS to increase its effectiveness as a power capping knob.*

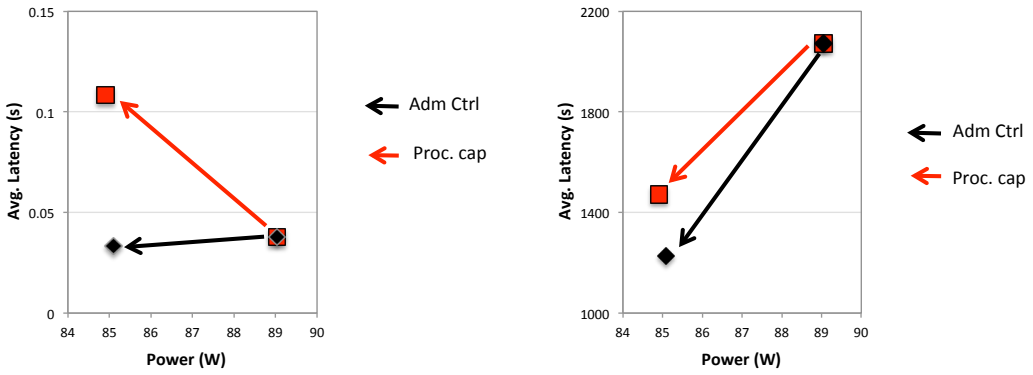
The design implication is that power capping techniques should coordinate with admission control agents, such as load balancers, to maintain application stability.

2.5 Admission Control in Closed Loop Systems

Admission control is also applicable in a closed application scenario, when the latency increase due to processor based power capping mechanisms (DVFS or processor capping) is not desirable. Unlike an open system, a closed system remains stable when the service rate is reduced (as shown in Section 2.3.2) . Under a frequency scaled regime, both latency and throughput degrade due to the slower clock speed and (if applicable) additional buffering delay. In contrast to DVFS, employing admission control could idle system components to achieve the same power reduction while keeping latency unaffected. However, from the discussion in Section 2.4.2 we have seen that admission control has a lower throughput per unit power than DVFS. Thus, admission control trades off additional throughput loss for its latency gains for power capping in closed loop systems.

Experiment: We use the same experiment setup as in Section 2.3.2. We reduce the power consumption of the server running StockTrader from 89W to 85W through processor capping and admission control. The network admission control was achieved by reducing the number of workload generators. Thus, we do not capture the additional load on the StockTrader server due to active rejection of requests.

Observations: Figure 2.10 illustrates the use of admission control in this closed loop system and compares it to the use of a processor based power capping mechanism. The system is first operated at a throughput near 2100 requests/s where the latency is just below 0.04s. At this point if the processor is throttled to reduce power, throughput falls to near 1500 requests/s while latency rises to above 0.1s (shown by the red arrows). On the other hand if the same power reduction is achieved through admission control, throughput falls to around 1200 request/s but the latency improves slightly due to lower number of requests being serve (shown by the black arrows). Thus, in a closed-loop system, admission control provides improvement in latency (due to load reduction), but reduces throughput. The exact nature of the tradeoff depends on the specific application.



(a) Change in Application Latency for Processor Capping and Admission Control

(b) Change in Application Throughput for Processor Capping and Admission Control

Figure 2.10. Difference in throughput and latency when admission control and processor capping is applied to reduce power consumption in a server running the closed loop StockTrader application

2.6 Discussion: Issues in Implementing Admission Control in Distributed Applications

As mentioned in Section 2.4, a power capping action needs to be accompanied by admission control to maintain application stability. Admission control for open-loop replicated applications can be performed efficiently at the load balancer, since it is the common point of entry for all incoming requests. In this section, we identify the need for coordinating the powercap controller (which would implement power caps) and the load balancer (which would implement admission control).

2.6.1 Typical enterprise cluster configuration

Enterprise open-loop services are replicated and run on multiple servers behind a load balancer. The loadbalancer's job is to distribute new incoming connections between the replicated servers, employing algorithms such as weighted/non-weighted round-robin, least-connection scheduling (scheduling a new request to the server currently serving the least number of requests), source hashing scheduling (scheduling a new request using a hash of the source IP address), etc. Load balancers dynamically update server weights to reflect server load profiles based on some periodic reporting mechanism [40]. Servers with higher weight receive a higher number of incoming connections and vice versa. Thus, a well-tuned load balancer may infer power capping actions dynamically through its reporting mechanism, initiating the necessary admission control. This would remove the need for any coordination between the powercap controller, and the load balancer.

2.6.1.1 Issues with lack of powercap-controller and load balancer coordination

Frequency of reporting mechanisms: There is a tradeoff between the frequency with which a load balancer updates server weights and the amount of network traffic caused by the generation of this information. For instance, if the load balancer balances load across 100 servers, a reporting mechanism which provides updates every second generates 100 new network packets, increasing the network incast traffic and affecting the performance of the load balancer (especially under overload conditions when a power capping action may be necessary). A slower rate of update, e.g once every 5 seconds, reduces the number of control packets, but fails to inform a load balancer of possible server frequency scaling for a longer period of time, during which the application could experience high latencies (as described in Section 2.3.1).

Incompatibiliy between powercapping and load balancing feedback loops: When the power capping controller and the admission controller do not coordidnate (such as in [14]), the interaction between a slow software power capping control loop and the load balancer's admission control loop should be studied for incompatibilities. Instabilities could crop up if the settling time of the software power capping control loop is comparable to the settling time of the loadbalancer control loop. Consider the following scenario: A power cap controller reduces the frequency of a server

to reduce power consumption. The load balancer infers the reduction in server frequency through its reporting mechanism and allocates less load to it. The lower load would drive down the power consumption of the server further. This might lead the powercap controller to increase the server frequency, creating a cycle, where the above-mentioned set of actions will be repeated. Coordination mechanisms such as the one mentioned in [41] could be used to mitigate this incompatibility.

2.6.1.2 An example implementation of coordination:

A possible way to achieve stable coordination is that when a capping action is enforced, the powercap controller instantly reduces the weight of the affected servers through load balancer APIs, such as those provided by LVS. Additionally, the powercap controller stops the reporting mechanism from modifying server weights (to prevent oscillation). While this could be achieved in our setup through temporarily stopping the Feedbackd daemon running on the power capped server, this technique is not general. The server weight for a power-capped server could be determined offline through benchmarking, or online by dynamically analyzing server load and power profiles. Under this particular coordinated approach, the need for frequent updates from the load balancer reporting mechanism would be reduced because the load balancer no longer *infers* power capping actions, and the possiblity of oscillations noted in Section 2.6.1.1 would be eliminated.

Chapter 3

Unobtrusive Power Proportionality

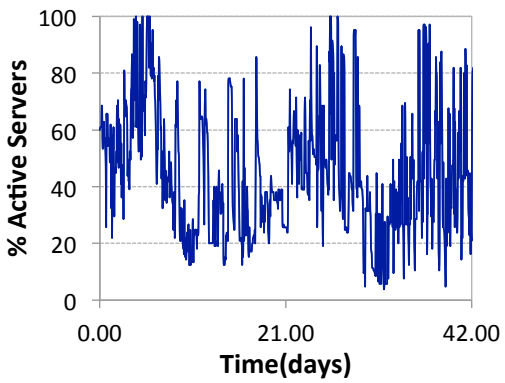
In the previous chapter, we described out two properties of power capping algorithms which become relevant in a production cluster. Power capping reduces primarily the capital costs of a cluster. In this chapter, we tackle the challenge of reducing the operating cost of a cluster. We present a system *Hypnos*, a power proportionality enabler. We first provide some background to the concept of power proportionality, before describing in detail the design, implementation and evaluation of *Hypnos*.

3.1 Background

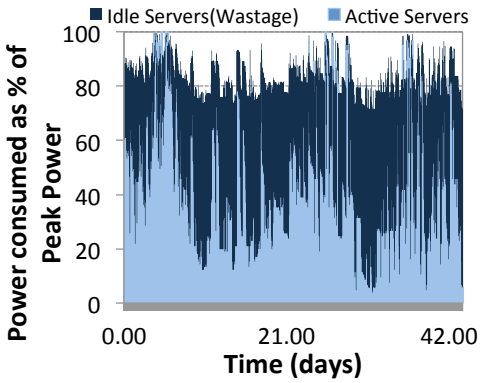
3.1.1 The Relevance for Cluster Power Proportionality

Servers are not power-proportional [1]. and can consume 30-80% [19], [20] of their peak power even when idle, implying that even largely idle clusters can consume copious amounts of energy (Figure 3.1). Even though the processor is becoming more power proportional through mechanisms such as frequency scaling and clock gating, other components (such as memory, IO, etc) and peripherals (fans, etc) are still non-power proportional [26]. Sleep states, which are common in mobile devices, are still not widely available in servers [42]. Trends do show that the server idle power consumption is decreasing. However, it will a long time before legacy servers are completely re-

[1] power consumption is not directly proportional to utilization



(a) Percentage of servers used by the cluster



(b) Energy consumed by active and idle servers in the cluster, as a percentage of the power consumed during 100% utilization (peak power)

Figure 3.1. Energy wasted by idle servers in a lowly utilized and over provisioned cluster in Berkeley

placed by completely power proportional servers, making cluster-level power proportionality the only option to reduce the energy wastage.

Consider a cluster with n servers, each consuming power P_{peak} at peak utilization and P_{idle} when it is idle. To get a very simple intuition into the energy savings possible due to power-proportionality, let us assume that each non-idle server is operating at its peak utilization, and the cluster is not over-provisioned [2]. Let the average number of servers running jobs over a time period T be n_{avg}.

The energy consumed by an ideal power-proportional cluster (E_{pp}) over the time period T would be

$$E_{pp} = n_{avg} \times P_{peak} \times T$$

The energy consumed by a non-power-proportional cluster [3] (E_{npp}) would be

$$E_{npp} = E_{pp} + (n - n_{avg}) \times P_{idle} \times T$$

Hence, the percentage energy savings obtained by converting a completely non-power proportional

[2] We consider the peak load in the trace to be the provisioned capacity

[3] a cluster which keeps all servers powered on the entire time

cluster to an ideally power-proportional one would results in energy savings ($\%E_{savings}$)given by

$$\%E_{savings} = \frac{(n - n_{avg}) \times P_{idle} \times T}{E_{npp}} \times 100$$

If we denote the ratio $\frac{Ppeak}{P_{idle}}$ by s[4], and the ratio of the total number of servers in the cluster to the average number of servers utilized over the time period as $f_{peak/avg}$, we can simplify the expression $\%E_{savings}$ as

$$\%E_{savings} = \frac{1}{\frac{1}{(f_{peak/avg}-1)\times s} + 1} \times 100$$

Three interesting points spring out of this simple analysis. First, if individual servers were power proportional (i.e $s = 0$), there would be no possible energy savings because the cluster would already be completely power-proportional. Second, the higher the idle power of a server in comparison to its peak (s), the higher is the energy savings possibile. Finally, the most important factor controlling the amount of energy savings is the peak to average ratio ($f_{peak/avg}$) of cluster utilization. The higher the value of $f_{peak/avg}$, the more is the incentive to deploy cluster-wide power proportionality mechanisms. A high value of $f_{peak/avg}$ indicates that the peak load the cluster is provisioned for occurs rarely, and a much lower average load keeps most of the servers spinning idly.

Table 3.1 shows the $f_{peak/avg}$ and $\%E_{savings}$ of some publicly available HPC traces, assuming $s = 0.5$ (i.e the server consumes 50% of its peak power when idle). From the analysis, it is evident that cluster power proportionality need not benefit all clusters. The NERSC clusters' average utilization closely matches its provisioned capacity and has very few idle servers to power down. The remaining clusters, mostly associated with academic institutes, display considerable amount of idleness and energy wastage. There is, thus, a strong case for implementing power proportionality in such under-utilized HPC clusters with $\%E_{savings}$ ranging from 38%-82%.

3.1.2 Feasibility of current techniques

Despite the potential for significant energy savings and the large body of existing research in this area, cluster power proportionality is not widely deployed. Through private conversations with

[4] s quantifies the server non-power proportionality (or *local* power proportionality)

Table 3.1. Statistics from different HPC clusters, whose utilization is archived in [43]. The degree of under-utilization of a cluster can be determined by its $f_{peak/avg}$ value. The notation used in the table is same as that in Section 3.1.1

Cluster	**Avg Util.**	$f_{peak/avg}$	$\%E_{savings}$
LCG	35%	2.9	49%
Grid 5000	17%	6	71%
Nordu Grid	10%	10	82%
AuverGrid	35%	2.9	49%
Berkeley PSI	45%	2.2	37.5%
NERSC Franklin	93%	1.08	3.8%
NERSC Hopper	88%	1.14	6.5%

system administrators managing HPC clusters, the primary concerns seem to be the obtrusiveness and reliability of the power management system.

There exists quite a few existing power managers available for HPC job-sharing frameworks. The licensed Green Computing scheduler from Moab [44] integrates power management into its framework and requires configuration files to be modified to enable power proportionality. System administrators are, thus, compelled to update the cluster configuration files every time the power management features need to be enabled or disabled. Also, reconfiguring a cluster setup increases the risk of misconfigurations and failures. Rocks-Solid [45] has a feature to enable power saving for a clustering running ROCKS [46], but deploys a very naive algorithm. It does not power down any server if even as long as there exists queued jobs. Also, it wakes all servers up until a queued job is executed. This is clearly inefficient because a job can be queued for fairness or other purposes, and waking up servers may not enable it run. It is, thus, unreliable. R-energy [47] is a remote energy management tool which implements power proportionality only for IBM servers which support IBM EnergyScale technology. Power-proportionality mechanisms which propose changing part of the existing cluster framework codebase (such as [21]–[25]) do not provide any guarantees about fault-tolerance or reliability, rendering them unsuitable for deployment.

Hypnos, takes an unobtrusive meta-system approach. Hypnos does not require any modification to the existing cluster software or network stack, and uses only existing interfaces provided by the cluster framework (in our case Torque and Maui). Also, any power-proportionality mechanism has to gracefully handle possible software and hardware faults and framework-specific idiosyncrasies that result from frequent power-cycling. [27] shows that the rate of non-terminal hardware failures in a cluster can be up to 1 in 190. Hypnos adopts a state-machine approach, where transitions of a server from any state to a failure state is inferred and handled. Additionally Hypnos' design is modular, enabling easy update and improvement of power management logic.

3.1.3 Loitering

To harness the idleness for energy savings, Hypnos utilizes the well-established technique of powering down servers when idle. However, if a job request arrives for a powered down server, that request will incur very high latency because the server must be woken up before that request can be served. This takes on the order of minutes. So care must be taken; we cannot put servers to sleep as soon as they become idle without harming performance.

Loiter time is the duration of time a server will remain idle before going it is powered down. Shorter loiter times means servers switch off more frequently, causing more job requests to suffer performance penalties. Longer loiter times keep servers remaining idle longer, decreasing energy savings. The loiter time for servers in a cluster needs to be set only after evaluating these tradeoffs.

3.2 Hypnos

Hypnos is a defensive meta-system which unobtrusively provides power proportionality for an HPC cluster. We explain Hypnos in the context of Torque [28], but it can be easily extended to the other HPC job-sharing frameworks such as IBM Load Sharing Facility (LSF) [48] and the Oracle (formerly Sun) Grid Engine (SGE) [49]. The principles guiding the design of Hypnos were

- Unobtrusiveness : Hypnos should not interfere with any existing cluster software or network

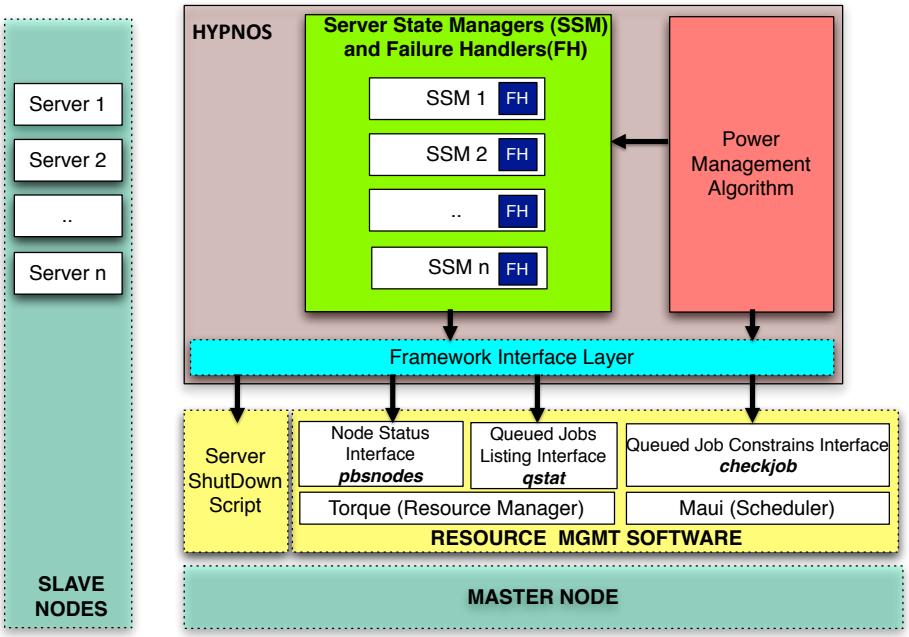


Figure 3.2. Component diagram of the various components of Hypnos (enclosed in the brown-shaded box). Hypnos lies on top of the Torque deployment on the cluster master node. It uses the *pbsnodes*, *qstat* and *checkjob* interfaces provided by Torque and Maui. For each server, the Server State Manager implements the state diagram shown in Figure 3.4

stack. The administrator should be able to turn the power management feature on or off without having to change any cluster configuration.

- Fault-Tolerance : Hypnos should be able to tolerate various software and network faults which might occur due to power-cycling, and should not stall or in any way affect the functioning of the underlying cluster framework or scheduler (Torque and Maui).

- Extensibility and Adaptability : Hypnos should allow easy deployment of different power management algorithms, and should be easily adaptable to platforms other than Torque and Maui.

Hypnos resides on the master node of the cluster, and only uses data available from standard framework interfaces to update its state. We assume, that each cluster node has a health-check script, which is integrated with the job-sharing framework (such as [50]). Also, we assume that the master node has the ability to power cycle servers remotely[5] . Hypnos satisfies all the design principles mentioned. The Hypnos system design is shown in Figure 3.2. We first give a brief description of Torque, before we describe in detail the Hypnos design and the rationale behind each of its modules.

About Torque : While the architecture of Hypnos is general, we describe it in the context of the open-source cluster management framework called Torque [28] and a job scheduler called Maui [51]. Torque manages the availability of and requests for compute node resources in a cluster and Maui implements and manages scheduling policies, dynamic priorities, reservations and fair shares of jobs. The Torque server and Maui scheduler resides centrally on the master node of a cluster. The remaining compute servers runs a Torque daemon which executed submitted jobs.

A sample job flow involves a script submitted to Torque specifying constraints. Maui periodically retrieves from Torque a list of potential jobs, available nodes, etc. When desired servers become available, Maui instructs Torque to execute jobs on them. Torque then dispatches the jobs to the compute servers, which then execute the job script. Maui periodically updates its information regarding job execution status. A job spools its output data on to local storage, and at the completion of job execution copies them to the user's NFS directory.

[5] These assumptions are satisfied by most production clusters

Table 3.2. Analogous interfaces to those used in Hypnos for LFS and SGE

Torque/Maui	**LFS**	**SGE**
pbsnodes	*badmin*	*qmod*
qstat	*bjobs*	*qstat*
checkjob	*bjobs*	*qstat*

3.2.1 Framework Interface Layer

The Framework Interface Layer abstracts out the detail of obtaining cluster information and actuation from the rest of the Hypnos system, and provides an API which the remaining modules use. This ensures easy adaptability of Hypnos to different cluster frameworks. Table 3.2 shows the analogues of the Torque interfaces we use on other job-sharing frameworks.

To implement power-proportionality in a cluster running a job-sharing framework with a shared file system, one only needs three pieces of information : the status of each node (whether it is active, idle, unable to run jobs or powered off), the list of jobs currently executing or queued up on the cluster, and the placement constraints of each of those jobs. Hypnos obtains this information from the Torque's *pbsnodes*, *qstat* and Maui's *checkjob* interfaces respectively. On the actuation side, a power-proportionality software only needs the ability to manipulate the state of each server. Hypnos achieves this through Torque's *pbsnodes* interface and through the cluster's shutdown script. The description of each of the used interfaces is shown in Figure 3.3(a).

The Framework Interface Layer, with the three methods listed in Figure 3.3(b), enables Hypnos to decouple the failure handling, sever state management and its power management logic from specific syntax and semantics of Torque and Maui.

3.2.2 Server State Manager

To implement Hypnos, we adopt a state-machine model to ensure easier analysis of the server. Hypnos creates one state machine for each distinct server under its control. The Server State Managers (SSM) implements the state machine shown in Figure 3.4 and a Failure Handler (FH) for each

Interface used	Associated Software	Use
pbsnodes	Torque	1. Get the state of a server. A server can be a. Job-exclusive (Server has no resources left to serve another job) b. Free (Torque can schedule jobs on this node) c. Offline (Torque cannot schedule jobs on this node) d. Down (cannot communicate to Torque client) 2. Change the state of the server from offline to free and vice versa
qstat	Torque	Lists the jobs current running and queued up on the cluster
checkjob	Maui	Provides details (constraints, status and resource requirements) of a submitted job

(a) Torque interfaces used by Hypnos

Module	Methods offered	Use
Server State Manager Objects	getState()	Returns the server's current state.
	getStateTime()	Returns time spent in the current state
	getConfigs()	Returns the server's capabilities
	changeState(*newState*)	Changes the server's state to the new state specified (if possible)
Framework Interface Layer	getServerState (serverid)	Returns the server state reported by Torque
	setServerState (serverid)	Changes server state by the pbsnodes command
	getQueuedJobs()	Returns list of queued and running jobs from the qstat interface
	getJobConstraints (*jobid*)	Returns a job's queue and resource demands from checkjob interface

(b) Methods exposed by each module of Hypnos

Figure 3.3. Description of the interfaces used between Hypnos modules, and between Hypnos and job sharing framework. The establishment of these interfaces ensures that the Framework specific library or the Power Management Algorithm can be easily modified or extended

server. Each server can be in 5 possible states. The transitions can be effected either by the SSM or by the Power Management Algorithm (PMA) module. The objective of the SSM is to decouple the power management logic from failure handling and server state tracking, ensuring a simple and easy-to-improve PMA.

The SSM has a timer associated with each state, which tracks how long a server has been in a particular state. The state machine (and associated failure handling) of each state is described below:

Online: This state signifies that the server is powered up and is either executing jobs or is idle. When a server is *Online*, the timer associated with this state is kept fixed to 0. Thus, the timer associated with this state indicates how long a server has been idle.

Offline: A race condition might occur between the PMA and Maui scheduler when the former wants to power a node down to save energy. A node being shut down by the PMA, may linger in the online node list of Torque for a brief period therafter. During this intermediate period, Maui may schedule a job on this server, leading to possible job failure. Thus, the *Offline* is a transitory state a server goes through before it is shut down. In this state, no job can be scheduled on the server.[6].

[6]Torque, LSF and SGE has support for the *offline* state for a server, where it removes it from its list of schedulable nodes

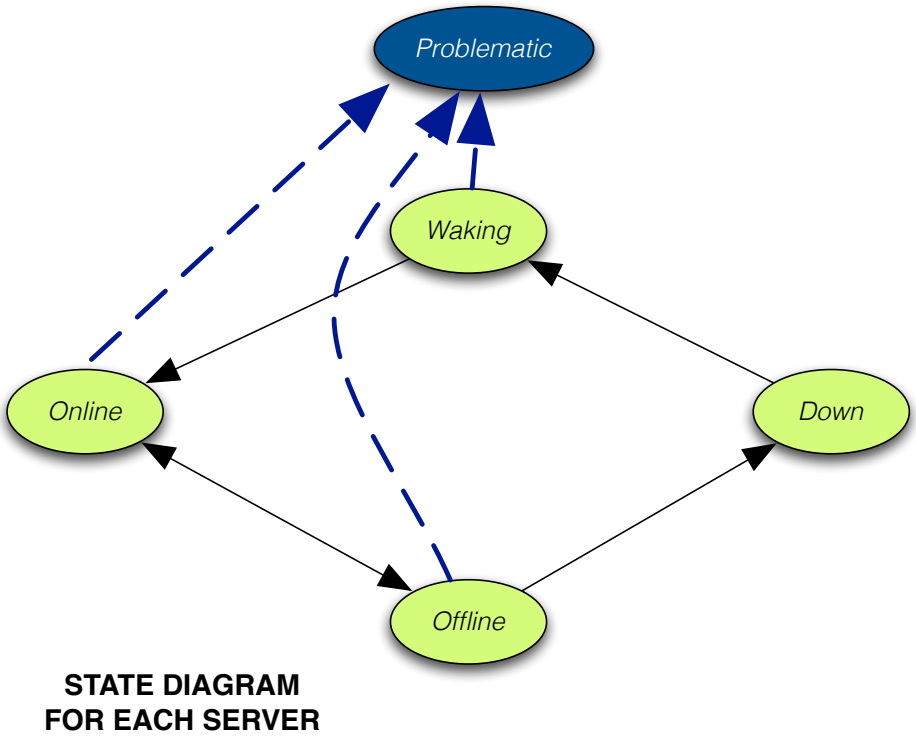


Figure 3.4. The different states and possible transitions of each server implemented by the Server State Manager

Down: This state signifies that the server is powered off. A server is not transitioned to the *Down* state until Torque reports that the node is unreachable.

Waking : Once the PMA wants a server powered up, the SSM sends a `wakeonlan` packet, and transitions the server to the *Waking* state. A server may taken anywhere between 3-5 min to reboot. This state is an intermediate transition transition state to account for the time elapsed between servers being powered on to when they become ready to execute jobs. This state also captures the possibility of failures when servers get powered on. Once the *pbsnodes* command confirms that a server is online, the state of the server is changed to *Online*.

Problematic: A server is in the problematic state, if the PMA or the SSM detects some kind of failure associated with the server. Since, a server can be in only one of three possible non-failure

states *Online, Offline, Waking* when it is powered up, any failure must result when the server was in one of these three states. We consider the reasons for such failures, and elucidate the way to detect and gracefully handle them.

- Failure from state *Waking* : For a server to successfully execute jobs, it might have to load multiple different software services over the network (such as the networked file system, DHCP configuration, working directories, etc), the failure of any of which might render it incapable of running jobs. Typically, a health-check script integrated with Torque is run at boot-time (and periodically thereafter) to check whether the list of necessary services is running properly [50]. If any of the required services are not loaded properly, Torque is instructed to mark the server as non-schedule-able. Once a server is powered on, the SSM waits for a configured time interval[7] to check if the node is reported to be schedule-able by Torque. In case, the node continues to be marked as non-schedule-able by Torque, the SSM changes the state of the server to *Problematic*, repeats the process of sending a `wakeonlan` packet and waiting. In case the server still does not wakeup properly after a fixed number of retries, the current version of Hypnos sends the system administrator is sent an email reporting the problem, while the SSM keeps retrying.

- Failure from state *Down*: This happens mostly when some running service prevents the server from shutting down, or the shutdown command packets are lost in the network. Once the SSM wants a server to be powered down, the SSM sends the shutdown command over the network, and waits for fixed time interval to check if the server is reported as shutdown by Torque. In case, Torque does not report the server to be shutdown, the entire shutdown process is repeated a fixed number of times, before the system administrator is sent an email notifying him of the problem.

- Failure from state *Online*: It may happen that the health check script on a server does not anticipate every failure scenario. For instance, one failure mode we encountered during deployment was that a few servers ran out of local disk space due to data spooling by Torque (a scenario that was not being checked by the health-check script at that time), rendering them

[7] 5 minutes in our deployment cluster. This was a reasonable amount of time for servers to shut down or wake up in our cluster. This value is taken as a parameter by Hypnos and can be modified to suit specific cluster needs.

unable to run jobs. There servers, though, were still being reported by Torque as schedule-able. To counter such a failure in the *Online* state, the SSM tries to run small dummy jobs periodically on a server which has been idle for a fixed amount of time, to ensure that they can still execute jobs. The Hypnos Director discounts these jobs while calculating server idle time. A successful execution of the small job implies that the target server is in a correct state and is schedule-able.

3.2.3 Power Management Algorithm

The Power Management Algorithm module defines the logic behind power down servers and waking them up new jobs arrive. Thus, the PMA in Hypnos, which is executed periodically (taken as a parameter in Hypnos; every 60 seconds in our deployment), only has to define two control loops - the wakeup and shutdown control loop for the servers. Algorithm 1 shows a simple pseudocode for the the PMA currently implemented in Hypnos.

3.2.3.1 Wakeup Control Loop

There are the two objectives of the PMA in the wakeup control loop. First, the PMA obtains a list of queued jobs through the `getQueuedJobs()` method and determines which servers to wake up. Hypnos achieves this through a naive bin-packing algorithm, where it goes through the list of*Online*, *Offline*, *Waking* and *Down* servers (which have been switched off for at least $T_{shutdown-timeout}$ period of time) in that order to look for servers to run the queued jobs. The order is important because, we want to avoid waking servers for jobs which can already be served by currently powered on, or already waking servers. If a queued job can be packed into an *Online* server, Hypnos assumes that job arrived in-between two Maui scheduling iterations, and will be scheduled at the next iteration. If a job can be run on an *Offline* server, Hypnos brings the server back *Online* because it is faster than to wake up a powered down server. If a job is packed onto a *Waking* server, it means that Hypnos had already woken up that server in a previous iteration for this job, and does not need to wake an additional server. If a job can only be run on a powered down server, or the *Online,Offline* and

s_i ▷ Server State Manager for server i

$S = \langle s_1, \cdots, s_n \rangle$ ▷ List of Servers State Manager Objects , one for each server controlled by Hypnos

$J = \langle j_1, \cdots, j_m \rangle$ ▷ List of queued jobs

$T_{online-loiter}$ ▷ Loiter time in the Online state

$T_{offline-loiter}$ ▷ Loiter time in the Offline State

REFRESH_RATE ▷ Time between two iterations of Hypnos

function MAIN

  while True **do**

    WakingControlLoop()

    ShutdownControlLoop()

    sleep(REFRESH_RATE)

function WAKINGCONTROLLOOP

  J = GetQueuedJobs() ▷ Get list of queued jobs from Framework Interface Layer

  WakeupRequiredServers(J) ▷ Wakes up servers in order to serve the queued jobs

  maintainHeadroom() ▷ Maintains headroom in each server class

function SHUTDOWNCONTROLLOOP

  for s_i in S **do**

    if s_i.getState() == "ONLINE" and s_i.getStateTime() > $T_{online-loiter}$ **then** :

      s_i.changeState(OFFLINE)

    if s_i.getState() == "OFFLINE" and s_i.getStateTime() > $T_{offline-loiter}$ **then** :

      s_i.changeState(DOWN)

Algorithm 1: Simplified Pseudocode to show the logic of the Power Management Algorithm module in Hypnos. The intricacies of the algorithm is given in Section 3.2.3

Waking servers have already been filled by the bin-packing algorithm, Hypnos instructs that server's state machine to wake up the server. [8]

Second, Hypnos maintains a constant headroom of idle servers. A typical cluster may contain servers of different configurations and capabilities. Jobs may have associated constraints that force them to be scheduled on a specific type of server. The PMA automatically groups the server configurations into different classes (each class consists of servers having the exact same configuration), and ensures that there is a headroom of idle spinning servers in each distinct *server class*. If server configuration heterogeneity is not explicitly maintained in the headroom, all servers of an infrequently used *server class* might get powered down. Thus, jobs which run on such servers will face a higher performance degradation, as compared to jobs which can run on a more frequently used *server class*.

3.2.3.2 Shutdown Control Loop

The shutdown control loop simply looks at the number of idle *Online* servers in each distinct server class. If the number of servers in that class is more than the required headroom, it switches the balance to the *Offline* state if they have already been loitering for more than $T_{loiter-online}$. Then, *Offline* servers, which have been loitering for more than $T_{offline_loiter}$ and have not accidentally been scheduled jobs by Torque are powered down.

3.2.4 Remarks

There are some intricacies in the power management logic worth noting. First, the PMA runs the wakeup control loop before the shutdown control loop to ensure that if *Offline* servers can serve queued jobs, they are brought back *Online* and not shutdown.

Second, let us have a look at how the PMA automatically circumvents the failure cases, and reliably ensures maximum energy savings.

- If a *Waking* server fails to come *Online*, it is flagged as problematic by the SSM. While

[8] A *Down* server is only woken up if it has been at least 5 minutes since it was transitioned to the *Down* state. This time period ensures that the server shuts down safely

running the next iteration, the PMA (which disregards *Problematic* servers during its bin-packing phase), will automatically bin-pack the jobs on to a new *Down* server, which will then be woken up. This ensures that enough servers are eventually woken up to serve a queued job.

- In case an *Online* server cannot run the SSM's dummy jobs and is transitioned to the *Problematic* state, the PMA in its next iteration will wake up a *Down* server to ensure a constant headroom of *Online* servers. Thus, the headroom will always consists only of non-problematic servers which can successfully run jobs.

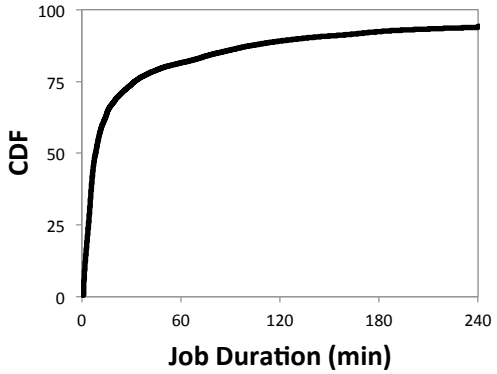
- Consider the case in which the power-manager Rocks-solid [45] (see Section 3.1.2) fails: a queued job is unable to run on the currently *Online* servers due to fairness violations or other reasons. Rocks-Solid wakes up all the powered down servers because it is unsure which server the job will run on. On the other hand, PMA will keep packing the queued job on the currently *Online* servers, and not wake the others up. This ensures that jobs which are not scheduled to run due to fairness violations or other reasons do not needlessly wake up powered down servers.

The complexity of the PMA is very low, because it has access to the abstraction of a state manager which automatically detects and handles failures (SSMs).

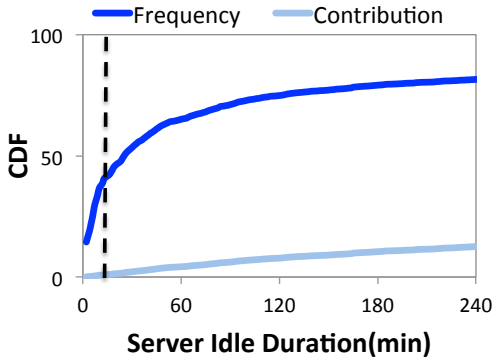
3.3 Implementation and Results

We deployed Hypnos on an academic cluster in Berkeley. The cluster is used by about 40 Artificial Intelligence, Machine Learning and Computer Vision graduate students.

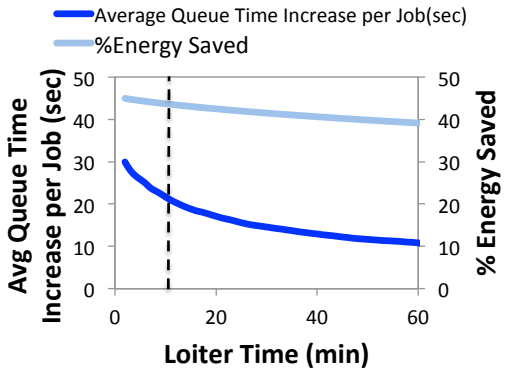
Server configurations: We deployed Hypnos on 57 servers of a cluster for 21 days. 51 of the servers were Dell PowerEdge 1850 servers having 2 cores running at 3.0GHz, 3GB RAM, consuming 192 W at idle and 292W at peak utilization. The remaining 6 servers were Dell PowerEdge 1950 servers with 8 cores at 2.3GHz, 16GB RAM, with an idle power of 253W and a peak power of 387W. The servers were automatically grouped into five classes based on the queues they belonged to and their hardware configurations, according to Torque's node configuration file.



(a) Cumulative Distribution Function of Job durations on our test cluster



(b) Cumulative Distribution Function of the idle periods in the cluster by frequency, and contribution to the total idle period of the entire cluster



(c) Impact on energy savings and job response time for various loiter times in the cluster

Figure 3.5. Characteristics of the test cluster before Hypnos was deployed

Hypnos configuration: Hypnos is written in Python and the entire system is 1349 lines long. The Power Management Algorithm module is only 346 lines. Hypnos is deployed on the cluster head node, where the Torque master resides. A 285-line health-check script was already deployed on each compute server. Servers were powered down remotely using the `shutdown` command in a bash script which had administrator privileges on all servers. Servers were powered up using the `wakeonlan` command.

3.3.1 Characteristics of the test cluster

In order to optimize the parameters of Hypnos for the most possible energy savings and to demonstrate that our testbed was a representative academic cluster, we analyzed a 68-day trace taken from the cluster when power proportionality was not deployed. During this period the cluster had executed about 169,000 jobs, and the average cluster utilization was 45%.

Figure 3.5(a) shows the CDF of the job durations submitted to the cluster during this period of time. Approximately 50% of the jobs take less than 10 minutes duration. This is probably due to the fact that users of the cluster have a debug cycle, where they run their jobs on small portions of data in order to check correctness of their code before running it on the full data set. This graph shows the need for a spinning reserve of idle servers (headroom). A headroom provides fast turnaround times for small jobs. Without the headroom, small jobs (of less than 10 minutes duration) may have had to wait for a server to spin up (which may take 4-5 minutes), resulting in users' debug cycle being extended by at least 40-50%. We keep a headroom of 3 idle servers in *each server class*, unlike Moab's Green Scheduler which only guarantees a spinning reserve but does not consider different server types. We, thus, increase the chances of any job submitted to find at least 3 idle servers ready to serve it.

Figure 3.5(b) shows the characteristics of the idle durations of the various servers in the cluster. The graph shows the CDF of idle durations, as well as the CDF of an idle period's contribution to the sum of all idle durations during the period. While most idle periods are small, their contribution to the total idleness of the cluster is insignificant. This indicates that some servers in our cluster run jobs very infrequently, and thus have large periods of idle which contribute heavily to the total cluster idleness. 40% of the idle periods are less than 10 minutes in duration, following which the CDF curve begins to flatten out. The contribution of the idle durations of less than 10 minutes to the whole cluster idle is about 0.7%. This indicates, that the majority of the cluster idleness is due to a few servers who remain mostly throughout. We choose the total loiter time of an idle server to be 10 minutes because it allows us to be moderately aggressive in shutting down servers, and yet harness most of the idleness in the cluster for energy savings.

Figure 3.5(c) shows the impact of various loiter times on energy savings and the amount of time

a job would have to spend in the queue waiting for a powered down server to spin up. Too small a loiter time would have resulted in lot of energy savings, but increased the job response times. We see that a 10 minute loiter time would increase average job queue times by only 20 seconds, while saving almost 40% of the energy. Note that increasing loiter times does not have a marked response on the amount of energy saved. As explained in the previous paragraph, the servers contributing most to the total cluster idleness run jobs rarely. So, even if we increase the loiter time these servers will eventually get powered down, resulting in large energy savings.

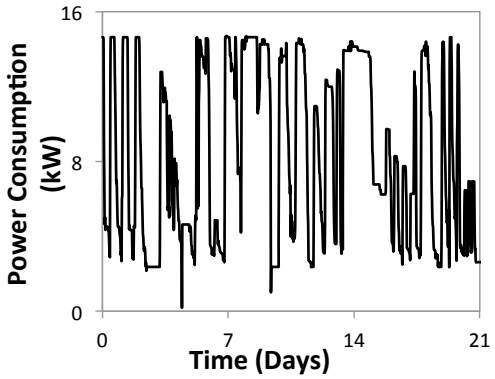
During our experiments, we divide the 10-minute loiter time into $T_{online-loiter} = 7$min and $T_{offline-loiter} = 3$min.

3.3.2 Results from deployment

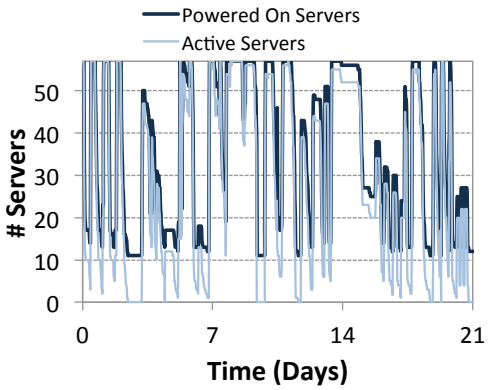
Hypnos was deployed without any changes to the existing Torque deployment. The energy reduction summary and performance impact from the Hypnos deployment is shown in Table 3.3. During our 21-day deployment, the cluster utilization rose marginally from the previous value to 46%.

Figure 3.6(a) shows the variability of the power profile of the cluster after Hypnos was deployed. Hypnos was able to save 36% energy compared to a scenario if it had not been deployed. Comparing it to Figure 3.6(b), we can see that the power profile closely matched cluster utilization, showing that Hypnos was able to switch the idle servers off effectively. Also, the number of servers switched on at any point of time was only slightly more than the number of active servers (servers running jobs).This demonstrates Hypos' reliability in powering servers up and down, and maintaining server headroom.

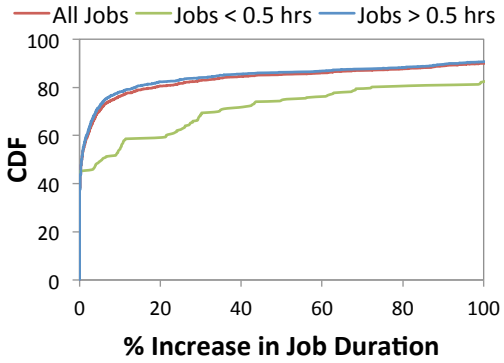
Figure 3.6(c) shows the impact of our parameter choices for loiter time and headroom on job performance. The CDF of the percentage of time a job spent in the queue time as compared to its execution time is shown separately for relatively small jobs (less than 30 min duration) and larger jobs (greater than 30 min duration). Almost 50% of all jobs faced less no increase in execution time. This is because of the headroom in each server class, which was able to serve jobs as soon as they were submitted. Almost 80% of the larger jobs had a less than 10% increase in their execution time because of encountering powered down servers. The remaining 50% of smaller jobs had a



(a) Power consumption of cluster under Hypnos



(b) The number of active servers (that are running jobs) and the number of powered up servers during the course of our deployment.



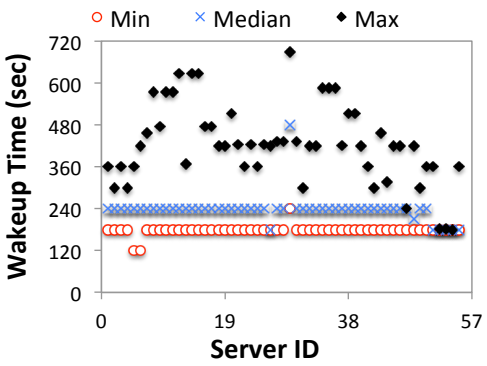
(c) Cumulative Distribution Function of a job's queue time as compared to it execution time (%Increase in Job duration)

Figure 3.6. Energy savings and performance impact under Hypnos' operation

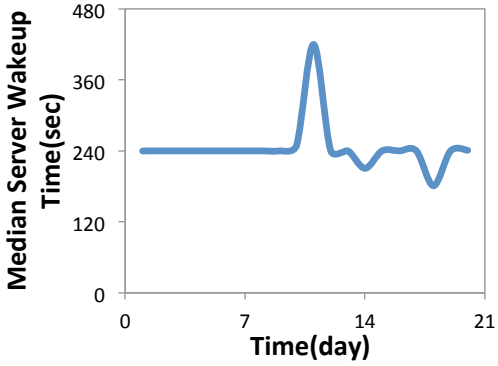
larger percentage increase in their job duration because the time for a server to wake up is large compared to the job's execution time. Some large as well as small jobs showed more than a 50% increase in their execution times. This was due to certain users submitting a large number of jobs at once, resulting in the cluster getting fully utilized, resulting in large job queue durations.

Metric	Value
Avg. Utilization	46%
Avg. Energy Savings	36%
Number of submitted Jobs	3651
Total number of Server Reboots	1094

Table 3.3. Usage statistics from the Hypnos deployment over 21 days on 57 servers



(a) Variation in wakeup time by server ID



(b) Daily variation is server wakeup time

Failed Service	Number of Occurances
Network Information Server	777
Network File System	773
Work Directory Server	4

(c) Frequency of service mount failures at reboot

Figure 3.7. Reliability and Fault-tolerance achieved by Hypnos duirng the 21-day run

3.3.3 Failure handling

Figure 3.7 shows the variable server wakeup times that Hypnos had to deal with. Servers might take a highly variable time for a successful reboot, which may be dependent on architecture as well as external conditions. Figure 3.7(a) shows that the median wakeup time of different servers

varied significantly. At boot time, each server in our test cluster has to contact the Network File System (NFS) server, the Network Information Service (NIS) server, and a server to load separate work directories before it is ready to run a job. Depending on the load on the NIS and NFS and work directory servers, the wakeup time of the cluster nodes may vary. While the median server wakeup time for most servers was 4 minutes, a server reboot might take as long as 12 minutes due to failure in mounting the NFS file system, working directories, or contacting the NIS server. Hypnos ensures fault tolerance by enforcing a restart timeout of 5 minutes, before which it powers up another similar server. There were also some days, where due to some transient conditions, the NFS and NIS servers responded too slowly, causing every server reboot to take much longer (Figure 3.7(b). The number of times the 57 server cluster failed to load the NIS, NFS and work directories is shown in Figure 3.7(c).

Chapter 4

Conclusions and Related Work

4.1 Related Work

There are four major areas of related work this thesis builds on. In this section, we briefy describe the existing literature in each of these areas.

4.1.1 Power Capping

Server level power capping methods [8] have been developed to throttle processor frequency in response to hardware metered power readings at millisecond granularity. Similar techniques for virtualized servers have been investigated in [52], [53], and use processor utilization capping in addition to frequency control. Since single servers methods do not make efficient use of the overall data center capacity, coordinated power budgeting across multiple servers has also been considered [10]–[14]. We build on these methods to address additional challenges. The coordinated methods rely on multiple feedback control iterations that, as we show, may not satisfy convergence conditions under rapid data center power dynamics. Stability concerns with open loop workloads are also not considered in these works. The control of processor frequency in open and closed loop system was considered in [38] but for energy efficiency rather than power capping, and hence the stability issue that arises in capping was not relevant in that context.

4.1.2 Admission Control

Admission control in web servers has also been studied in depth. Admission control methods drop requests to prevent the server from getting overloaded [54]–[56] and maintain acceptable performance. Feedback control and queuing theoretic algorithms that carefully trade off the number of dropped requests and performance have also been studied [57], [58]. Processor frequency management to maximize energy efficiency for variable incoming request rates along with admission control have been considered in [59], [60]. Techniques to implement admission control by preventing new TCP connections or selectively blocking requests based on the HTTP headers were presented in [61]. However, the integration of processor frequency management and admission control has not been considered for power capping. We discuss the desirable characteristics from both techniques that are relevant for this problem in this thesis.

4.1.3 Power Proportionality in Servers and Clusters

Existing research has looked into reducing the energy consumed by individual servers (i.e reducing idle energy consumed by servers) by using low-power processors [62], [63] or by introducing sleep states into servers [42], [64]. However, sleep states and low power processors are still not widely available in data center type servers. The alternate approach of enabling cluster-level power proportionality has been explored by [21]–[26]. These techniques range from carefully switching off replica servers using a covering subset scheme to switching off the entire cluster at periods of time to maximize energy savings. However, these techniques require extensive modification of currently deployed server software, making it hard to deploy in production clusters.

4.1.4 Existing HPC Power Proportionality Enablers

There are three other existing softwares which enable power proportionality in High Performance Computing clusters. Rocks-solid [45] is general and works with multiple cluster-frameworks such as SGE and Torque, but has an extremely unreliable server shutdown and wakeup policy. Renergy [47]attains energy efficiency only in IBM EnergyScale supported servers. Hypnos' approach is similar to that of the Green Computing Scheduler provided by Moab [44]. Hypnos enables bet-

ter performance because it keeps a spinning reserve in each pool, and also is more general and open-source, and does not require changes to cluster configuration files.

4.2 Conclusions and Future Work

In this thesis, we explore the essential principles that should guide software systems that try to reduce the capital or operating expenditure of a data center. The cost of provisioning power and cooling capacity for data centers is a significant fraction of their expense, often exceeding a third of the total capital and operating expense. Power capping is an effective means to reduce the capacity required and also to adapt to changes in available capacity when demand response pricing or renewable energy sources are used. We described why existing methods for power capping lack two desirable properties of speed and stability and showed where these properties can make the existing power capping mechanisms infeasible to be applied. We also presented an approach based on admission control to ensure stable and efficient operation. While admission control cannot replace existing methods due to multiple practical issues, we showed how it can provide the desirable characteristics in a capping system when used together with existing mechanisms.

We have also demonstrated Hypnos - a power-proportionality meta-system, which is unobtrusive, reliable and flexible in its design. We argued that a meta-system approach is more general (applicable to different resource managers), cost-efficient (in terms of code maintenance and ease of deployment) and felixble (allows the resource management software to update its code base without considering power proportionality). Although, we deployed and tested Hypnos on a Torque cluster, the design decisions and interfaces used have analogues in other HPC job-sharing frameworks such as LFS and SGE. The main aim behind developing Hypnos was to provide an open-source solution to cut down on the idle energy consumed in under-utilized clusters. We report results from Hypnos over a 21-day period, where it was able to save 36% of the energy without succumbing to hardware or software faults.

4.2.1 Future Work

There exist several ways the principles laid out in this thesis could be exploited for better and more efficient power management in data centers. Although the usefulness of admission control in power capping has been illustrated, several open challenges remain. These include the design of specific algorithms that control the extent of admission control applied, its implementation in an efficient manner with minimal modifications of deployed applications, and safe integration of multiple control mechanisms. Future work also includes prototyping rapid power capping mechanisms that can quickly reduce power in case of rapid dynamics and then use feedback to iteratively refine the power settings for maximum performance within the safe operating region.

There are also various ways to improve on the Hypnos architecture and algorithms. First, an optimized power management algorithm that considers the server wakeup actuation as stochastic could reduce queue times for multi-server jobs. Such an algorithm can keep track of the average wakeup times of each server, and power on servers with a low wakeup time. Also, in instances where past data shows that a server wakeup time is unreliable (has a high variance), it could power on more servers than required in order to serve the queued jobs. Hypnos, in its current form does not support job reservations. Overall, we believe that Hypnos can enable easy innovation in power management algorithms deployed against the abstraction of a state machine which takes care of failures.

Overall, our experience is that most of the academic literature in data center power management is primarily towards obtaining the most capacity reduction, or the most energy savings, often discounting the concerns of data center operators. We believe that understanding of relevant issues developed in this work will enable further research towards deploy-able power management solutions.

References

[1] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1109/MC.2007.443

[2] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, "Power routing: dynamic power provisioning in the data center," in *ASPLOS*, 2010.

[3] J. Hamilton, "Cost of power in large-scale data centers," Blog entry dated 11/28/2008 at http://perspectives.mvdirona.com, also in Keynote, at ACM SIGMETRICS 2009.

[4] L. A. Barroso and U. Holzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[5] L. A. Barroso, "The price of performance," *Queue*, vol. 3, no. 7, pp. 48–53, 2005.

[6] J. Koomey, "Growth in Data center electricity use 2005 to 2010. ," CA: Analytics Press, 2011.

[7] R. Brown *et al.*, "Report to congress on server and data center energy efficiency: Public law 109-431," 2008.

[8] C. Lefurgy, X. Wang, and M. Ware, "Server-level power control," in *Proceedings of the Fourth International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 4–. [Online]. Available: http://dl.acm.org/citation.cfm?id=1270385.1270763

[9] P. Ranganathan, P. Leech, D. Irwin, J. Chase, and H. Packard, "Ensemble-level power management for dense blade servers," in *In Proceedings of the International Symposium on Computer Architecture (ISCA*, 2006, pp. 66–77.

[10] M. E. Femal and V. W. Freeh, "Boosting data center performance through non-uniform power allocation," in *ICAC*, 2005.

[11] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, 2010.

[12] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller, "Ship: A scalable hierarchical power control architecture for large-scale data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 1, pp. 168–176, 2012.

[13] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: coordinated multi-level power management for the data center," in *ASPLOS*, 2008.

[14] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *USENIX Annual Technical Conference*, 2011.

[15] "S. V. L. Group. Data center energy forecast," .http://svlg.org/campaigns/datacenter/docs/DCEFR_report.pdf, 2009.

[16] G. Grid, "Green grid metrics data center power efficiency metrics: Pue and dcie," *Technical Committee White Paper*, 2007.

[17] "Google data center PUE performance," http://www.google.com/about/datacenters/efficiency/internal/.

[18] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, 2nd ed. Morgan Kaufmann Publishers, Inc., 1996.

[19] S. Dawson-Haggerty, A. Krioukov, and D. E. Culler, “Power optimization âĂŞ a reality check,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-140, Oct 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-140.html

[20] X. Fan, W.-D. Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA ’07. New York, NY, USA: ACM, 2007, pp. 13–23. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250665

[21] Y. Chen, S. Alspaugh, D. Borthakur, and R. H. Katz, “Energy efficiency for large-scale mapreduce workloads with significant interactive analysis,” in *EuroSys*, 2012, pp. 43–56.

[22] R. Bianchini and R. Rajamony, “Power and energy management for server systems,” *IEEE Computer*, vol. 37, p. 2004, 2004.

[23] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt, “Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system,” in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 274–287. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.109

[24] W. Lang and J. M. Patel, “Energy management for mapreduce clusters,” *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 129–139, Sept. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1920841.1920862

[25] J. Leverich and C. Kozyrakis, “On the energy (in)efficiency of hadoop clusters,” *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 61–65, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1740390.1740405

[26] D. Meisner, B. T. Gold, and T. F. Wenisch, “Powernap: eliminating server idle power,” in *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS ’09. New York, NY, USA: ACM, 2009, pp. 205–216. [Online]. Available: http://doi.acm.org/10.1145/1508244.1508269

[27] E. B. Nightingale, J. R. Douceur, and V. Orgovan, “Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer pcs,” in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys ’11. New York, NY, USA: ACM, 2011, pp. 343–356. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966477

[28] “Torque Resource Manager,” http://www.adaptivecomputing.com/products/open-source/torque/.

[29] D. Bhandarkar, “Watt matters in energy efficiency,” Server Design Summit, 2010.

[30] “Duke utility bill tariff,” http://www.duke-energy.com/pdfs/scscheduleopt.pdf.

[31] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, “Benefits and limitations of tapping into stored energy for datacenters,” in *International Symposium of Computer Architecture (ISCA)*, 2011.

[32] X. Fan, W. dietrich Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” in *In Proceedings of ISCA*, 2007.

[33] J. Hamilton, “Energy proportional datacenter networks,” http://perspectives.mvdirona.com/2010/08/01/EnergyProportionalDatacenterNetworks.aspx, 2010.

[34] A. Verma, P. De, V. Mann, T. Nayak, A. Purohit, G. Dasgupta, and R. Kothari, "Brownmap: Enforcing power budget in shared data centers," in *Middleware(ODP)*, December 2010.

[35] HP, "Power regulator for proliant servers," http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00300430/c00300430.pdf.

[36] X. Fu, X. Wang, and C. Lefurgy, "How much power oversubscription is safe and allowed in data centers?" in *The 8th International Conference on Autonomic Computing*, June 2011.

[37] D. Meisner and T. F. Wenisch, "Peak power modeling for data center servers with switched-mode power supplies," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED '10, 2010.

[38] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *SIGMETRICS*, 2009.

[39] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the internet," *Commun. ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2063176.2063196

[40] "Feedbackd," http://jk.ozlabs.org/projects/feedbackd.

[41] J. Heo, D. Henriksson, X. Liu, and T. F. Abdelzaher, "Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study," in *RTSS*, 2007, pp. 227–238.

[42] A. Gandhi, M. Harchol-Balter, and M. A. Kozuch, "The case for sleep states in servers," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, ser. HotPower '11. New York, NY, USA: ACM, 2011, pp. 2:1–2:5. [Online]. Available: http://doi.acm.org/10.1145/2039252.2039254

[43] "The Grid Workloads Archive," http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Main.Home.

[44] " Green Computing powered by Moab," http://www.clusterresources.com/solutions/green-computing.php.

[45] "ROCKS-SOLID," https://code.google.com/p/rocks-solid/wiki.

[46] "ROCKS-CLUSTERS," https://code.google.com/p/rocks-solid/wiki.

[47] "RENERGY," http://xcat.sourceforge.net/man1/renergy.1.html.

[48] "IBM Platform LSF," http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/index.html.

[49] "Oracle Grid Engine," http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/index.html.

[50] "Warewulf Node Health Check," http://warewulf.lbl.gov/trac/wiki/Node%20Health%20Check.

[51] "Maui Scheduler," http://www.adaptivecomputing.com/resources/docs/maui/pbsintegration.php.

[52] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. L. R. III, "Application-aware power management," in *IISWC*, 2006.

[53] R. Nathuji, P. England, P. Sharma, and A. Singh, "Feedback driven qos-aware power budgeting for virtualized servers," in *FeBID*, 2009.

[54] L. Cherkasova and P. Phaal, "Session-based admission control: a mechanism for peak load management of commercial web sites," *Computers, IEEE Transactions on*, vol. 51, no. 6, pp. 669 –685, jun 2002.

[55] J. Carlstrom and R. Rom, "Application-aware admission control and scheduling in web servers," in *IEEE INFOCOM*, 2002, pp. 506 – 515.

[56] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS'03, 2003.

[57] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi-tiered web applications using queueing predictor," in *NOMS*, 2006, pp. 106–114.

[58] M. Kihl, A. Robertsson, and B. Wittenmark, "Analysis of admission control mechanisms using non-linear control theory," in *ISCC*, 2003, pp. 1306–1311.

[59] V. Sharma, A. Thomas, T. F. Abdelzaher, K. Skadron, and Z. Lu, "Power-aware qos management in web servers," in *RTSS*, 2003.

[60] C. Poussot-Vassal, M. Tanelli, and M. Lovera, "A control-theoretic approach for the combined management of quality-of-service and energy in service centers," in *Run-time Models for Self-managing Systems and Applications*, ser. Autonomic Systems. Springer Basel, 2010, pp. 73–96.

[61] T. Voigt and P. Gunningberg, "Adaptive resource-based web server admission control," in *ISCC*, 2002, pp. 219–224.

[62] 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 ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 1–14.

[63] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White, "Low-power amdahl-balanced blades for data intensive computing," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 71–75, 2010.

[64] D. Meisner and T. F. Wenisch, "Dreamweaver: architectural support for deep sleep," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2012, pp. 313–324.