THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IMPLEMENTATION AND EVALUATION OF A SOFTWARE-BASED POWER VIRTUALIZATION TECHNIQUE

PAUL LAWRENCE BILY
SPRING 2013

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Computer Science and Mathematics
with honors in Computer Science

Reviewed and approved* by the following:

Bhuvan Urgaonkar
Associate Professor of Computer Science and Engineering
Thesis Supervisor

John Hannan
Associate Professor of Computer Science and Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

As data centers grow in size and popularity, managing utilization of resources like power becomes increasingly important due to rising costs. Effective power management remains challenging due to the dynamic needs of applications across different executions and hardware configurations resulting in occasional peaks in usage that make efficient power allocation difficult. Thus, a software-based virtualization technique is presented that helps coordinate current power techniques, such as DVFS, battery hierarchies, and overbooking capacity, to more effectively allocate and manage power usage across a data center. In particular, it considers how to fairly reallocate power during a power emergency. An implementation is presented and tested via simulation to examine its efficacy. This demonstrates how coordinating power usage in a data center can robustly virtualize power and allow for more aggressive overbooking while maintaining performance and fairness considerations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

Electrical power is a key expense within a data center, amounting to roughly a third of overall costs [1]. Capital expenditures cover a wide array of infrastructure necessary to properly distribute and maintain power requirements: power lines, switchgears, power distribution units (PDU), backup generators, uninterruptible power supplies (UPS), etc. This amounts to roughly $10-$20 per watt deployed, regardless of whether a watt is utilized [2,3]. This can become expensive if a data center decides to deploy enough infrastructure to support the rarely observed peak power usage when all servers simultaneously draw their maximum power, as a portion of this infrastructure will rarely be used. Ideally it is only worth purchasing infrastructure that will be used with a high frequency. Regardless, to prevent failure, enough infrastructure must be provisioned to support a predetermined peak power draw within the data center.

Operating expenditures are composed of energy and demand charges. The energy charge is based on the overall amount of energy used in a given month. Conversely, demand charges are typically based on 15-30 minute demand intervals where the cost is determined by the peak power usage during the interval. In essence, demand charges compensate utility companies for the difficulty of handling sudden spikes across the power grid. The larger the difference between average power usage and peak power, the larger the percentage of operating expenses will be spent on demand charges. Put another way, by redistributing power usage to be more even across time, the energy usage and resulting energy charge will be largely unchanged while the peak power and resulting demand charge will be reduced. As such, it is cost effective to reduce usage spikes and spread energy usage evenly across time [4]. Thus, to reduce energy costs across the

data center, minimize the expected maximum power draw of the data center to reduce provisioned infrastructure, and normalize power usage so the peak draw within a demand interval is closer to the average power used, resulting in smaller demand charges.

Unfortunately, power usage varies greatly between applications, hardware configurations, and executions. Most programming languages and resulting applications are designed based on an abstraction of underlying hardware. As a result, applications generally have little idea how power is being utilized and rarely take peaks in power usage into consideration. In addition, different hardware can result in vastly different power usage of applications [5]. Since hardware power specifications are often insufficient or too inaccurate to predict actual power usage, extensive profiling is often necessary to have reasonable power usage expectations. Even after a reasonable power profile is established, peaks in power usage will occur due to different phases, exceptions, or usage fluctuations of an application. This necessitates that peaks in power usage must be allowed. Regrettably, a study in a Google data center has shown that power usage greater than 90% of peak requirements occurs less than 1% of the time [6].

This results in the dilemma of wanting to minimize peak power usage to provision less infrastructure and incur smaller demand charges while still providing the infrequent peak power used by applications. The general strategy to meet both these criteria is to overbook (or under-provision) the power infrastructure and have contingency mechanisms available should multiple peak power demands occur simultaneously [7], [8]. By overbooking, the excess power needed to meet an application's peak usage can be shared among applications in the data center, resulting in more normalized power usage overall.

Since applications rarely utilize their peak power needs, multiple applications requiring peak power happens infrequently. This infrequency can be increased by using power profiles and

usage history to schedule applications' power needs temporally and spatially across the data center. If a number of simultaneous peak power draws occur resulting in a power emergency, a number of power techniques (or 'knobs') can be used to alleviate power needs such as dynamic voltage and frequency scaling (DVFS), discharging of UPS batteries, or application migration.

Together, this creates a virtual impression of available power for applications where they expect all power needs will be met when in reality power is a limited resource. This is similar to how virtual memory works, wherein the operating system provides the illusion of infinite memory to an application when in fact there is a finite amount of memory available. While each of these techniques for managing power is beneficial by themselves, standard software to manage and coordinate them all has not been established. In particular, little work has been done on deciding which applications should have their power throttled in an emergency. It would be useful if such selection took into account priority, behavior, and proportional allocation.

Thus, a software implementation is presented that monitors data about all levels within the power hierarchy and uses it to coordinate which techniques are utilized when faced with a power emergency. It takes into account factors such as an application's priority, behavior, and proportional allocation when making power decisions to allow for better performance. The implementation is then used for a simulation to evaluate the efficacy of the power hierarchy and reallocation policy. Based on simulation, the framework works as intended, allowing for aggressive overbooking while maintaining robust performance and fairness criteria. This lays groundwork for extending such a framework into actual deployment within data centers.

2. Background and Related Work

Data center power hierarchies consist of roughly four levels. There is typically an automated transfer switch that connects the data center to the utility line and backup generators. Below that is some arrangement of power distribution units that distribute power to server racks, which in turn distribute power to individual servers. Monitoring power usage is typical at most if not all of these levels. Power usage is typically monitored with meters connected to outlets or built-in hardware, but work has also been done to predict power usage base on observed operating system information [9]. Research has started to look into monitoring power usage of individual virtual machines within the same server [10]–[12]. In addition, research has also begun looking into metering power for individual applications [13]. Monitoring and coordinating power usage at all levels in the power hierarchy is becoming increasingly popular as it allows more aggressive under-provisioning and overall financial savings.

Scheduling is a major component that enables aggressive power provisioning and overbooking [6], [14]–[20]. Spatial and temporal allocation of workloads can employ numerous strategies to carefully spread workloads across the data center, taking into account factors such as temperature, geographic load balancing, and performance. A simple example is scheduling batch jobs which can run anytime during the day around known hotspots for service oriented workloads. Another example is to offset applications that have periodic power usage so that their periods of activity do not correlate. Alternatively, it may be desirable to schedule workloads to correlate their power usage to create alternating charge and discharge periods for batteries.

A related aspect of scheduling is the inevitable migration of applications between servers [21]. In general migration is a fairly costly process and should only be used when a prolonged power emergency is experienced or relocating an application allows shutting down portions the data center. In these situations, special consideration must be given to service applications, especially those that have long run durations.

Another important way to manage power is dynamic voltage and frequency scaling (DVFS) [22]–[25]. By using power state modulation to throttle hardware, power demands can be spread across time to fit within a desired power cap. The most common example of DVFS is lowering a CPU's clock speed or replacing a portion of instructions with No Operations (NOP) to reduce power usage. It should be noted that DVFS is best used as a temporary solution, as throttling power usage will also result in reduced performance. Despite reducing performance, DVFS is a useful tool as it is fairly ubiquitous and provides an effective last resort in handling a power emergency.

UPS batteries are generally present in data centers to handle power outages. By allowing the use of a portion of a UPS batteries' charge, they can temporarily alleviate a power emergency without degrading application performance. This practice allows UPS batteries to function as power buffers within data centers with acceptable impact on availability for power outages [4], [26]–[28]. UPS batteries can be arranged in numerous ways within a data center including a single centralized UPS, rack level UPS, or per server UPS [28]. In general, UPS batteries are becoming increasingly distributed across data centers for numerous reasons such as fault tolerance. The more levels batteries exist on within the power hierarchy, the more aggressive an overbooking policy can be used with careful coordination for power outage guarantees.

These numerous ways to enforce power caps are coordinated in a variety of ways, but a general framework has yet to emerge. Some work has been done for virtualizing batteries for mobile and embedded devices [29], [30]. While a similar concept, this differs in the overall goal of prolonging use of a finite resource as opposed to setting power cap goals to achieve more regular energy usage. The embedded systems attempt to establish a rigorous accounting system that is not entirely portable, as a data center is coordinating multiple power sources between applications of different priority and execution time rather than attempting to fairly spread a finite resource across applications. Other work has been done to coordinate multiple power usage within clusters [31]–[33]. These works are helpful, but make assumptions about homogeneity across servers within the clusters.

There has also been work done examining a middleware representation of power hierarchies with an admission, accounting, and enforcement component [34]. This work creates a distinction between actual and allocated power. This distinction is crucial as it is necessary to allow allocated power to be greater than actually available power in order to overbook. This work also provides useful insight into the order in which to use power mitigation techniques when encountering a power emergency, namely first use batteries, then DVFS, and finally migration. The following implementation builds on this work by examining a more intricate power reassignment mechanism while focusing less on scheduling and energy. In addition, this work monitors application behavior in a vastly different manner as it does not use an accounting mechanism.

3. Design

3.1 Power Hierarchy Representation

The software implementation is designed as a middleware package that interfaces between all the relevant parts of the power hierarchy. This requires creating a representation of the power hierarchy and then implementing necessary functionality.

Power flows down the hierarchy such that the power usage at a given PDU or rack is a function of the power needs of the units to which it distributes power. As such, a violation at a given level can be alleviated by reducing power usage downstream. This relationship between power levels forms a strong parent-child relationship and makes a tree an ideal data structure for representing the power hierarchy.

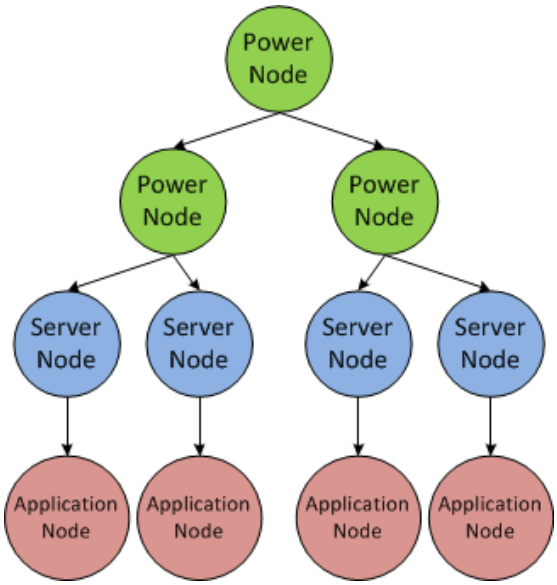

Figure 1: Example power hierarchy tree.

In this implementation, a node class is used as the base object which is extended into a representation of any entity within the power hierarchy. Configuring a power hierarchy will consist of specifying the properties of and relationships between nodes, as shown in Figure 1. In this sample configuration, the root node represents power flow to the entire data center. It distributes this power to two power nodes, which would represent racks. These power nodes then

distribute power to their servers, which use the power to run the applications they have been assigned.

Generally, interaction will only occur between parent and child nodes. For this initial implementation the power tree is controlled as elements within a single process, but could be made scalable by enabling parent-child interactions across a network, allowing tree management to become a distributed application. In addition to interacting with its parent and child nodes, a given node will contain the functionality to retrieve updated information from the entity the node represents as well as send it new policy instructions. In the simulation, this functionality will be based on models, but in actual deployment it is expected nodes would use Simple Network Management Protocol (SNMP), secure shell (SSH), or other similar utilities for remote power control.

3.2 Maintaining Power Caps

Both updating a node's information and considering power techniques (knobs) to maintain power caps will occur at regular intervals at all nodes in order to manage the power hierarchy. Since a node cannot use its parent to alleviate a power emergency, considering power usage for the next time interval begins at the lowest level, as these power emergencies must be handled locally. Power usage will then be considered at every level moving up the tree, pausing to manage emergencies as they are discovered.

When faced with a power emergency, a given node will use knobs available locally at the node or delegate the power deficit to child nodes until the power cap is met. Children chosen as victims make a best effort to alleviate the deficit rather than trying to proportionally spread the

deficit across all children for a number of reasons. Firstly, certain knobs do not alleviate power usage at a fine granularity. Proportionally reallocating power in an emergency could result in fragmentation where each child ends up with unused power when trying to meet its new allocation. Thus, it is better to use as few children as necessary to alleviate a deficit, which will result in at most one child having surplus power. Secondly, some children may have no priority, and should always be used to alleviate a power emergency. Lastly, rotating children to alleviate a power emergency rather than using all children simultaneously is important for certain configurations that use batteries, as it allows opportunities for the batteries to recharge.

3.2.1 Victim Selection Algorithm

Selecting a victim to alleviate a power emergency is similar to choosing a page to evict in virtual memory, except that multiple victims may be chosen to meet a power cap, as unlike page eviction, a chosen child will not alleviate a set amount of power. The amount of deficit a child can alleviate is limited by the amount of power it currently draws from the parent.

For victim selection, an inverse lottery provides an apt mechanism to select the next victim

$$p = \frac{1 - \frac{t}{T}}{n - 1}$$

where p is probability, t is the number of tickets of a participant, T is the total ticket pool, and n is the number of participants [35]. As a participant in the lottery gains a larger portion of tickets in relation to the total ticket pool $\frac{t}{T}$, their probability of being chosen diminishes. Conversely, as a participant's portion of the ticket pool approaches zero, their probability of being chosen as

victim is limited by $\frac{1}{n-1}$. This scheduling algorithm ensures fairness by guaranteeing that all participants have some chance of being chosen as the victim, preventing the same participant from being chosen for every lottery. In addition, desired performance criteria such as priority can be implemented with careful allotment of tickets, giving more tickets to participants with more importance.

In general, a child's quantity of tickets should be a function of its priority, proportional watt allocation, and behavior. For proportional allocation, it makes sense to allocate an application a number of tickets directly related to the power (in watts) it requested. In order to give applications different priorities, allow applications to specify the number of tickets to receive per watt requested. For example, a high priority application might receive five tickets per watt allocated, while a low priority application only receives one ticket per watt. This gives applications a base number of tickets such that

$$Base\ Tickets = Watts\ Allocated * Tickets\ per\ Watt$$

and effectively takes priority and proportional allotment into account. This base number of tickets can then be modified by some measure of behavior to establish a final allotment of tickets prior to an inverse lottery.

Since a given node represents the power needs of all applications assigned below it in the power hierarchy, it follows that a node's base number of tickets should equal the sum of tickets of descendant applications in the power tree. Consequently, a node that is an ancestor to no applications will always have zero tickets. A node that is not providing power to an application does not have performance guarantees, and so any node with zero tickets will always be chosen as a victim before any lotteries occur. This creates an adequate triage where first children with no

applications are chosen as victims, and then children with applications are chosen where less important children have a higher probability of being chosen.

3.2.2 Determining Node Behavior

In addition to allotting tickets based on allocation and priority, it is beneficial to take into account how much power a node is using relative to its allocation. In other words, nodes that are consistently using more power than they were allocated should have an increased chance of being selected as a victim. Similarly, nodes that are currently using less power than allocated may soon start using more power, and should be chosen as a victim less frequently. Furthermore, applications that are using more power relative to their allocation are also the applications most likely to provide ample power deficit relief with certain knobs such as DVFS. To increase a child's chance of becoming a victim, it will need to have its number of tickets reduced when it starts to misbehave. It should be noted that misbehaving applications should also be the first applications migrated when faced with a long lasting power emergency.

Thus a behavior modifier must be established that will reduce the base number of tickets allotted based on power usage in relation to power allocated. Rather than using the power usage of the most recent reading to determine behavior, an exponentially smoothed average is used

$$Average\ Power_n = Average\ Power_{n-1} + \alpha(Current\ Power - Average\ Power_{n-1})$$

where α is some smoothing factor that determines the ratio between the instantaneous power usage and average power usage of the relative past. An example of what such a weighted average looks like in relation to actual power usage is shown in Figure 2. The weighted average is a

representative average of the most recent power measures and resilient to bias from an instantaneous power usage spike, providing a better indicator of the behavior of an application.

Once the ratio of average power usage to power allocation is determined, it must be used to manipulate the tickets of a node. A sigmoid function asymptotically approaches a limit in both positive and negative directions, providing an easy way to map the ratio of power usage into a finite range. A graph of the equation used for this implementation is shown in Figure 3. If an application is using less than its allocated power, then it is allowed to use close to 100% of its base tickets. As the power used nears 100% of that allocated, the tickets an application is allowed to use are reduced to 70% of its base tickets. Past 100% of an applications allocated power, the number of tickets it is allowed to use for lotteries quickly diminished to 10%. Specifically, a Gompertz function was used as it allows a more gradual reduction in ticket allotment when

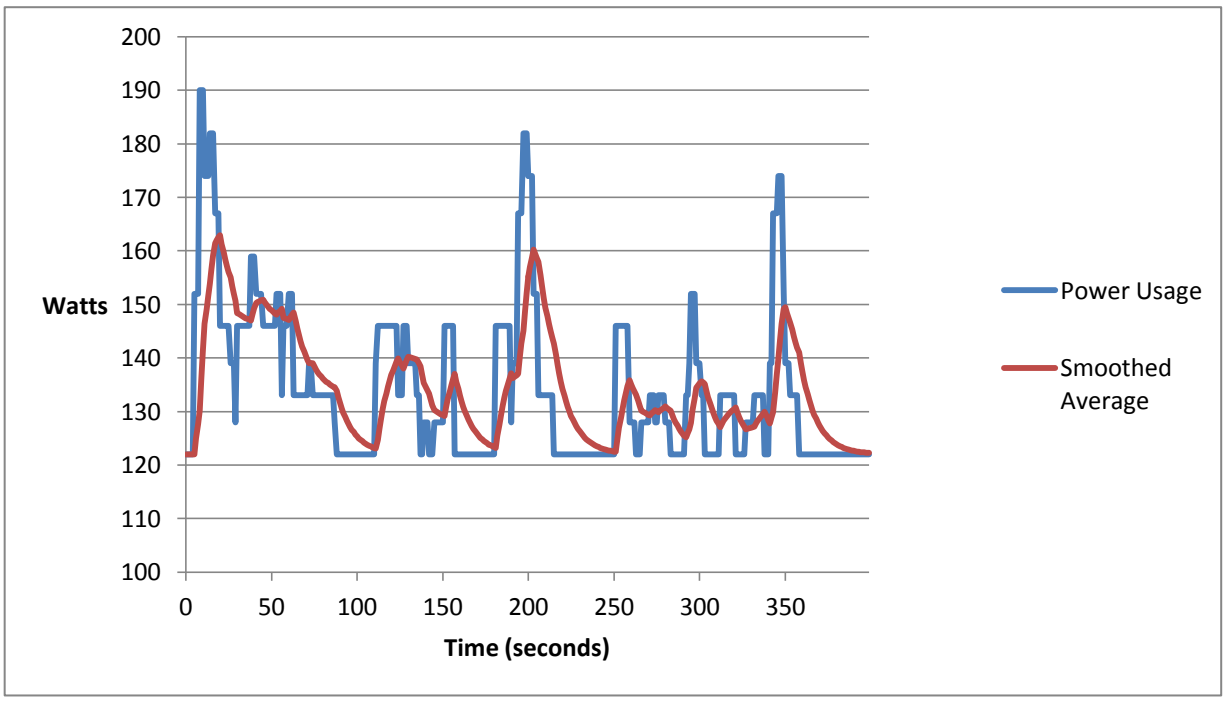

Figure 2: Instantaneous to average power comparison.

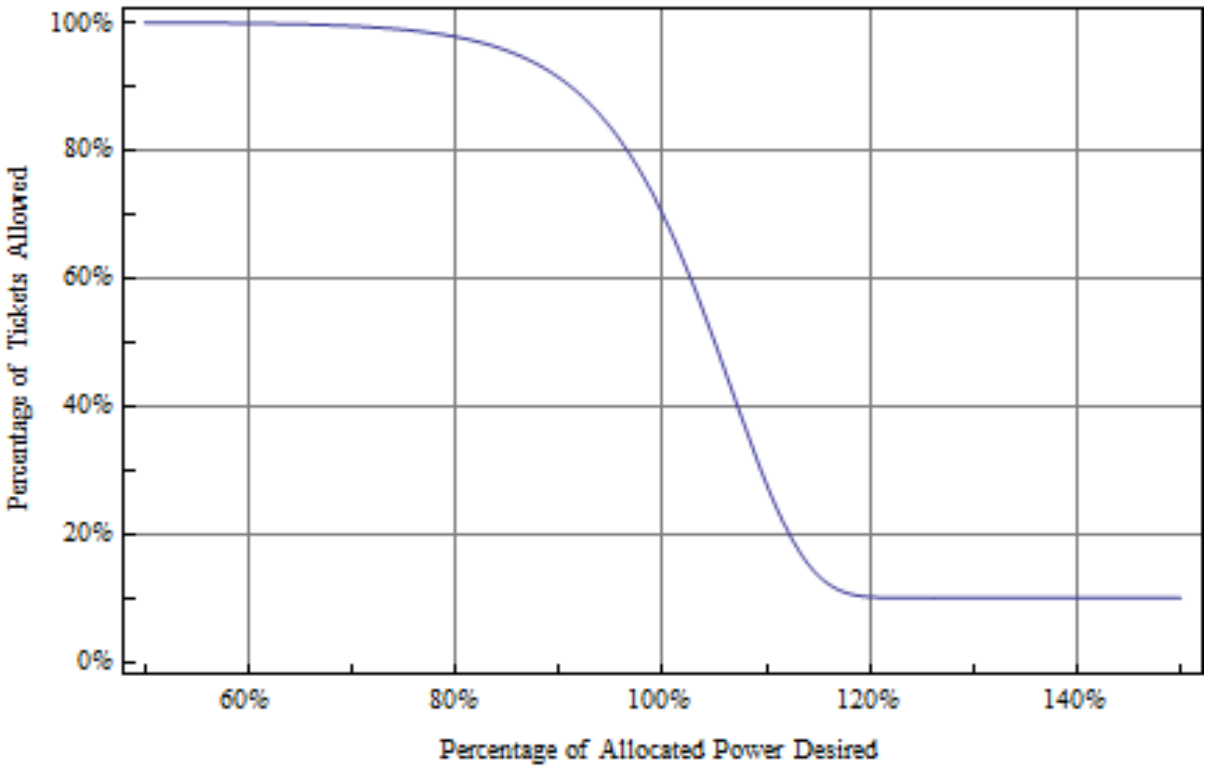

Figure 3: Implementation's Gompertz function.

approaching the allocated power and a faster reduction when exceeding the allocated power. This establishes a behavior modifier that reduces the tickets a node can use as it approaches and exceeds its power allocation. The behavior modifier is based on a weighted average that is representative of the power usage of a node within a recent time interval.

3.3 Implementation Details

The base node class contains all values to provide the functionality discussed, such as parent-child relationships, power allocated, power used, average power usage, base number of tickets, etc. Regular update calls are made to every node in the power hierarchy to refresh node data and enact power policy for the next time interval. In this implementation, the update call

refreshes both power measurements and policies. In an actual deployment it may be ideal to split this functionality into two separate calls to allow for more flexibility.

The two core classes extended from node are power and application nodes. Application nodes provide a representation of workloads assigned to machines within the datacenter. Generally, applications are not expected to measure power usage nor have mechanisms to manipulate power used. The application node specifies information about an application's needs, such as power necessary for allocation and the number of tickets it receives per watt. The application node would also describe necessary runtime and power profile information, but such advanced scheduling is not considered in this implementation.

Since this implementation uses simulation, a simulated application node is created that plays back a power profile. In essence, it accesses a file that has a list of the power used at regular intervals across an execution. The simulated application node uses this data to update its internal structure and inform the parent node of its power usage.

Power nodes provide a base representation for nodes that have sources of power to monitor and mechanisms to affect power use. They contain functionality for monitoring power consumption, enacting policies when faced with power violations, and determining if they can admit a given application. Power nodes start with zero allocated power and tickets, only having them as long as descendant application nodes are in the tree. When an application is assigned in the power hierarchy, all ancestor power nodes have the application's power allocated and base tickets added to the given power nodes current amount. When an application finished execution, the tickets and power allocated are then removed from ancestor power nodes.

Power nodes contain power supplies, which take the form of power lines or batteries in this implementation. A supply will specify the amount of power it can provide, as well as a

policy that describes what portion of the power should be taken into account for management. For example, power supplies higher in the tree will want to enact a policy that allows overbooking or allocating more power than is actually available. When a power node tries to use more power than its power sources can supply, a power emergency occurs. During a power emergency the power node will employ local power knobs as well as delegate the power deficit to children using the lottery algorithm described previously. In this implementation power nodes may or may not have a UPS battery available.

In addition to power nodes, server nodes provide a special case of power nodes that have applications as children. Their functions tend to be slightly different than power nodes to reflect this distinction, for example they will not delegate a power emergency to an application. Servers can use DVFS to alleviate power usage in addition to UPS batteries if available.

4. Evaluation

A simulated implementation is presented to evaluate the efficacy of the framework. The simulation is based on a combination of ideal models and collected data with some assumptions. The simulation will allow various configurations to be explored and demonstrate aspects of the framework. Ideally, the framework could be extended into actual deployment with network protocols such as SNMP and SSH. In addition, the simulation could be made more robust to take additional factors into account in future work.

4.1 Experimental Setup

For simplicity, the simulation will ignore power loss between levels of the power hierarchy. As such, it is assumed that a PDU will always use power equal to the sum of the units to which it provides power. The simulation will be based on DELL PowerEdge servers with two Intel Xeon 3.4 GHz processors. Their maximum power draw is listed at 450W, with a typical idle power draw of 122W. The simulation will support a 3.2 GHz, 3.0 GHz, and 2.8 GHz clock rate in addition to the base 3.4 GHz to implement DVFS. The performance and power reductions caused by these power states will be borrowed from the highest workloads measured in a SPECjbb benchmark run on the same hardware configuration [4], and are shown in Table 1. Thus, the implemented DVFS

Power State (GHz)	Power Reduction (%)	Performance Degradation (%)
3.4	0	0
3.2	6	8
3.0	15	13
2.8	22	18

Table 1: DVFS power assumptions.

model takes some liberty in simply recording intervals that use DVFS as having lowered performance and power usage, whereas in real applications the runtime would be extended due to the shifting of instructions executed to a later time interval. Still, this simple model provides the desired functionality of using DVFS to alleviate a temporary power emergency in order to enforce a power cap.

The battery model used is entirely theoretical. Discharing is based on Peukert's law with a Peukert constant of 1.1

$$\Delta Capacity = Discharge\ Rate^{1.1} * Time$$

It ignores degradation of maximum charge over time. It expects a specification of available charge in the form of ampere-seconds, as well as a minimum charge to retain for power emergencies. It is assumed that the batteries can supply and receive charge at fine granularity. In reality, most batteries only allow discharge at a number of fixed rates. It also assumes that batteries are used as a second source of power, rather than an intermediary between child and parent node that only discharges when experiencing an outage. This would require that servers have two power inlets. In simulations that use a battery, if there is surplus power available it will be used to recharge the battery with lowest overall charge. While this model for a battery is optimistic, it is representative of how batteries would be used as a power buffer to temporarily alleviate power emergencies during execution.

4.2 Workloads

As previously stated, applications are simulated by playing back a recorded power profile of a given application. Thus, the power used by a given type of application will be the same for

every execution, which is convenient for comparing simulations of different configurations. The power usage of applications were profiled by running them on the previously described DELL PowerEdge server setup, using SNMP to poll the connected PDU for outlet power draw at one second intervals. It follows that the simulation will be based on one second intervals.

The workloads on which the simulation is based are Hadoop Map Reduce [36], Nutch Web Crawl [37], and Solr Web Search [38]. Specifically, the base deployments of these respective workloads were used. These workloads are representative of typical map reduce, web crawl, and web search applications that are common in data centers. For the remainder of the paper, when a map reduce application is referred to, it is referring to the playback of power usage recorded from executing Hadoop Map Reduce on the describe Dell PowerEdge. Likewise when referring to a web crawl or web search application. Map reduce and web crawl are typical batch applications, whereas web search is a typical service application. Thus the applications provide representative examples of data analytics and web search applications that are frequently observed in data centers [39].

For simplicity, this implementation only allows one application per server. This simplifies the simulation as power usage can vary drastically when multiple applications execute on the same server. Specifically, the power used by a server would be split across applications resulting in an application using less energy over time, but resulting in higher power usage as applications would be contending with other applications for the server's resources. By avoiding the complexity of multiple applications executing on the same server, it is assumed that many of a servers properties, such as power usage, are identical to the application the server is hosting.

4.3 Initial Simulations

As a first example, consider the power usage of all three types of applications. Let there be a rack that distributes power to three servers. One server will run a simulated map reduce application, another web crawl, and the last a web search simulation. The start of execution will be lined up so all three applications intentionally use their peak draw at the same instant. This will force what would otherwise be a rare maximum power draw. The power usage during the simulation is shown in Figure 4. Note that this simulation shows the power profiles recorded for all three applications. In subsequent simulations, it is these power usage profiles that will be mitigated, and thus this initial simiulation should be used for comparison as no power cap is enforced.

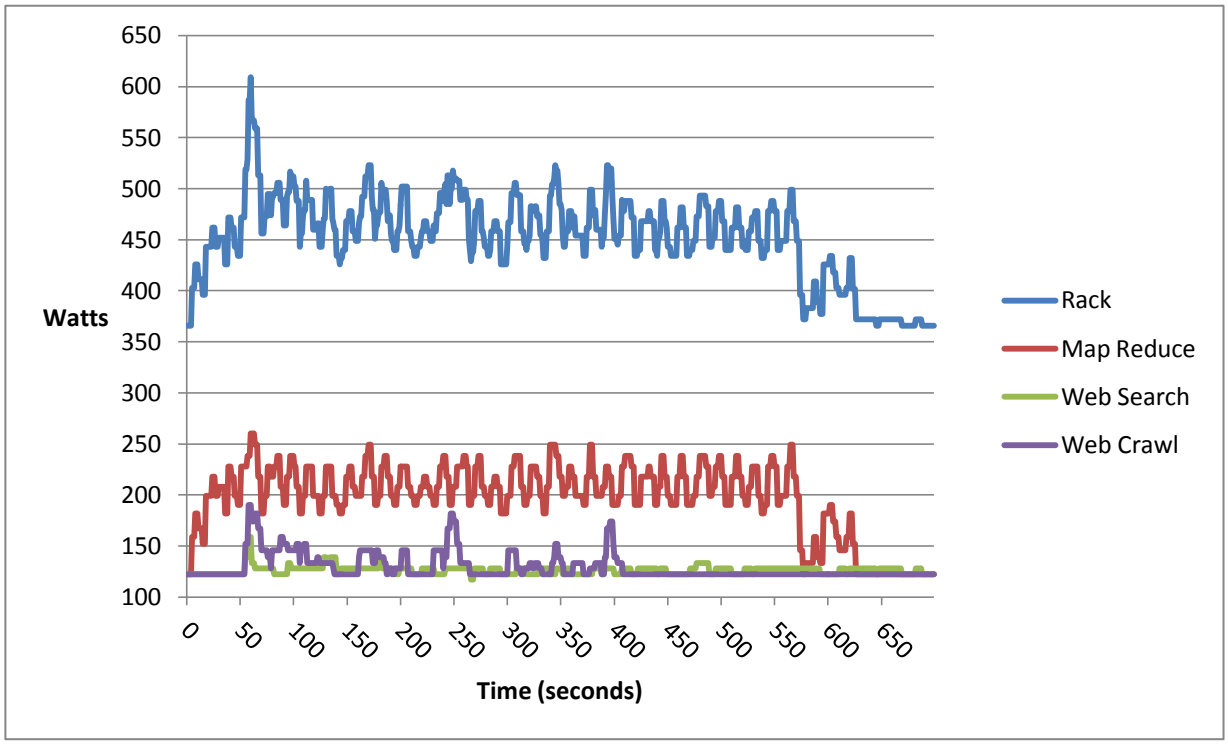

Figure 4: Power usage of initial setup.

Individually, map reduce uses a maximum power of 260 watts, web search 159 watts, and web crawl 190 watts. Altogether, the rack requires a total allocation of 609 watts in order to be able to handle all possible power demands from this configuration of applications. The rack never uses over 530 watts except for the brief spike that occurs when starting web crawl and web search simulatenously. To better illustrate the rarity of using peak power, a cumulative distrubtion function is presented for the rack and individual application power usage in Figures 5 and 6. As shown, web search uses over 140 watts less than 1% of the time. Web crawl uses over 170 watts about 5% of the time. Map reduce uses over 240 watts less than 4% of the time.

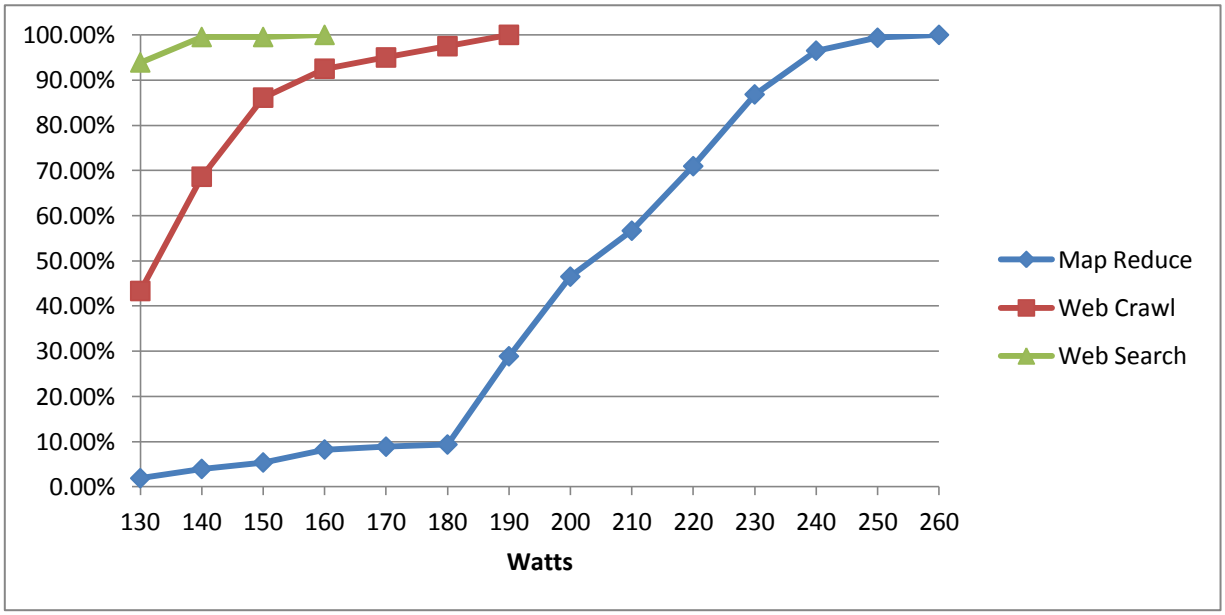

Figure 5: Application power usage cumulative distribution function.

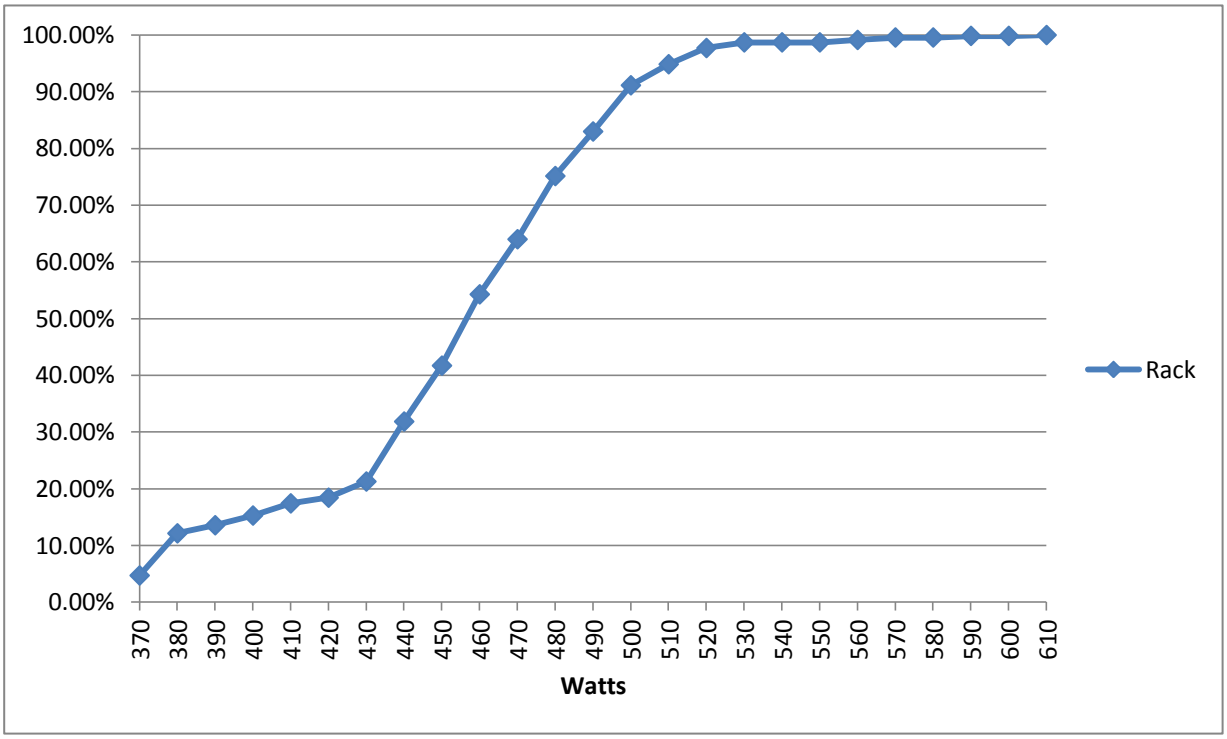

Figure 6: Rack power usage cumulative distribution function.

If the power allocations for each server were capped at the above levels, the overall allocation needed for the rack would be lowered to 550 without under provisioning at the rack level. If another simulation is performed with this configuration using DVFS to maintain power caps, it results in the servers using the clock states with frequencies shown in Table 2. While the power caps cause a reduction in execution 5% of the time, it allows a significant decrease in total power needed as shown in Figure 7. This illustrates the core concept of capping power requirements below the actual requirements for lower power usage. Further savings could be gained by overbooking the rack, for example capping the rack power

	3.4 GHz	3.2 GHz	3 GHz
Map Reduce	96.51%	2.86%	0.63%
Web Search	99.53%	0.00%	0.47%
Web Crawl	95.00%	2.50%	2.50%

Table 2: DVFS state frequency.

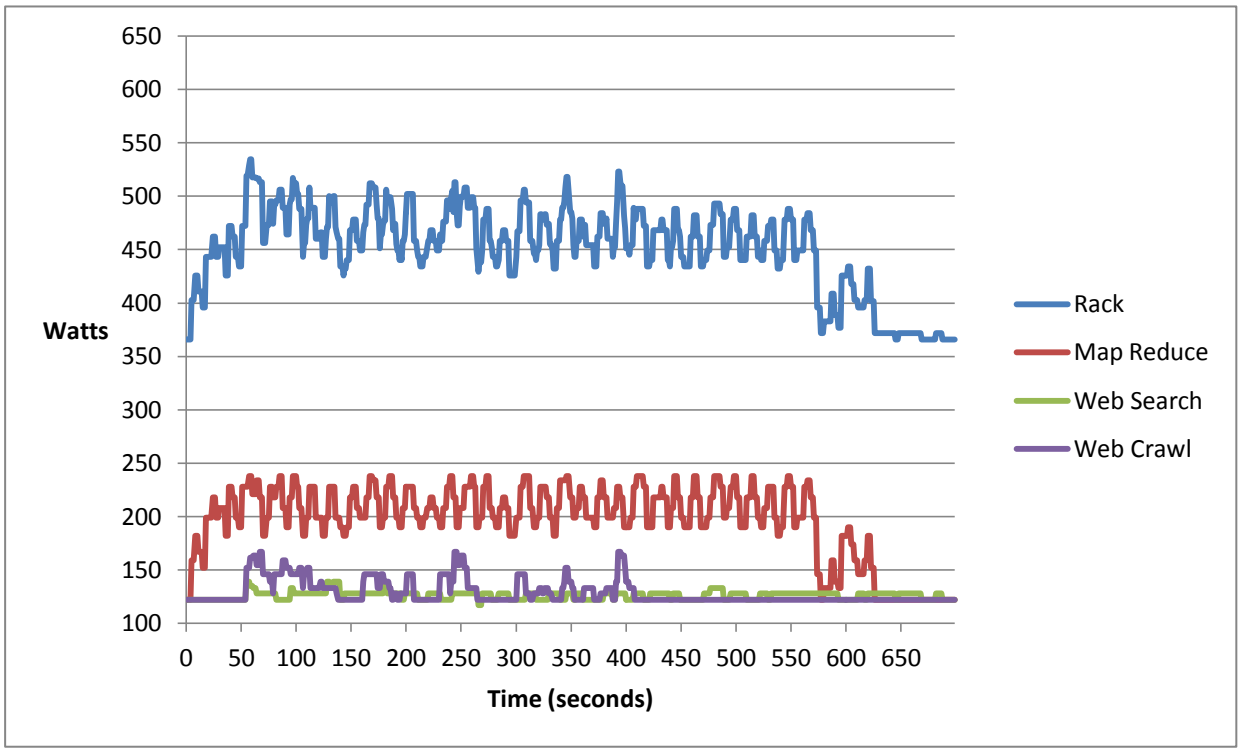

Figure 7: Power usage of power capped servers.

draw at 520 watts. More aggressive provisioning as well as batteries will be considered after examining the lottery and behavior algorithms.

4.4 Lottery Examination

First, a simulation that examines the effects of base tickets will be considered. Since altering power allocated and tickets per watt have the same effect on scheduling and victim selection, only watts per ticket will be explored. To illustrate the effect, consider a rack with three servers running the previously described simulated map reduce application. This is an ideal setup as all three applications will attempt to use the same power, but certain applications will be selected as a victim less frequently. Since all three servers are simulating map reduce, they will

attempt to draw 260 watts during execution. To force a power emergency that must be delegated to a victim, the rack is limited to only having 730 watts.

The rack is setup as a simple power node with no local power knobs, so an emergency will be delegated to a server to handle with DVFS. Since all three applications will attempt the same power usage and are given the same allocation, their behavior will be essentially identical and their watts per ticket will be the determining factor for frequency of selection as victim to alleviate a power emergency. Let one map reduce application be configured with 10 tickets per watt, another 5, and the last application 1 ticket per watt.

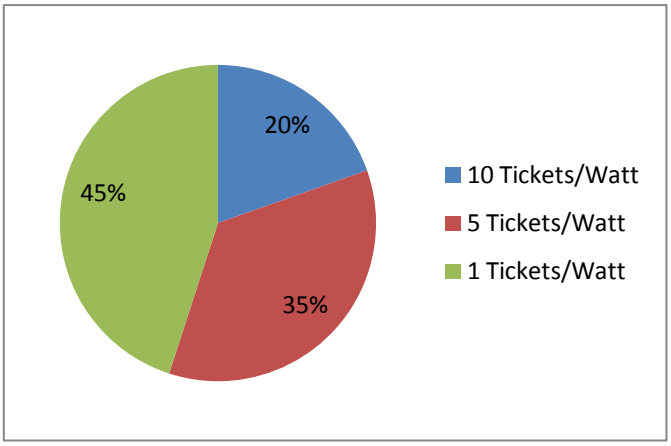

Figure 8: Victim frequency for difference in tickets per watt.

The easiest way to examine the effect of differences in tickets per watt between the application configurations is to look at the resulting frequency of being the victim. Since victim selection is probabilistic, and the number of power emergencies resultant from the simulation relatively small, this configuration is run a large number of times so the victim frequencies are more representative of the underlying probabilities.

Figure 8 shows the resulting frequency of selection as a victim during multiple simulations. As show in the figure, the smallest weight of 1 ticket per watt results in being chosen as the victim roughly 45% of the time. Conversly having the highest weight of 10 tickets per watt allow the application to be chosen as victim only 20% of the time. Based on the

previously discussed inverse lottery equation the probability expected for the 1 ticket per watt allocation is

$$p = \frac{1 - \frac{1 * 260}{(1 + 5 + 10) * 260}}{3 - 1} = \frac{1 - \left(\frac{1}{16}\right)}{2} = \frac{15}{32} \cong 46.9\%$$

This result is fairly close to the observed 45% victim frequency, which indicates that the inverse lottery and base tickets are working as intended. Next, a situation where the behavior is the determining factor in victim selection is considered.

To achieve this, a similar setup with three servers each running a simulated map reduce application is used, but this time each applications allocated power usage is configured differently. One application will be allocated the maximum required 260 watts and thus will never use more power than it was allocated. Another will be allocated 240 watts which is representative of a resonable power cap, as the map reduce was profiled to use over 240 watts less than 4% of the time. The last application will be allocated 200 watts, which will result in frequent violation of power allocation and severe throttling of alloted tickets for lotteries.

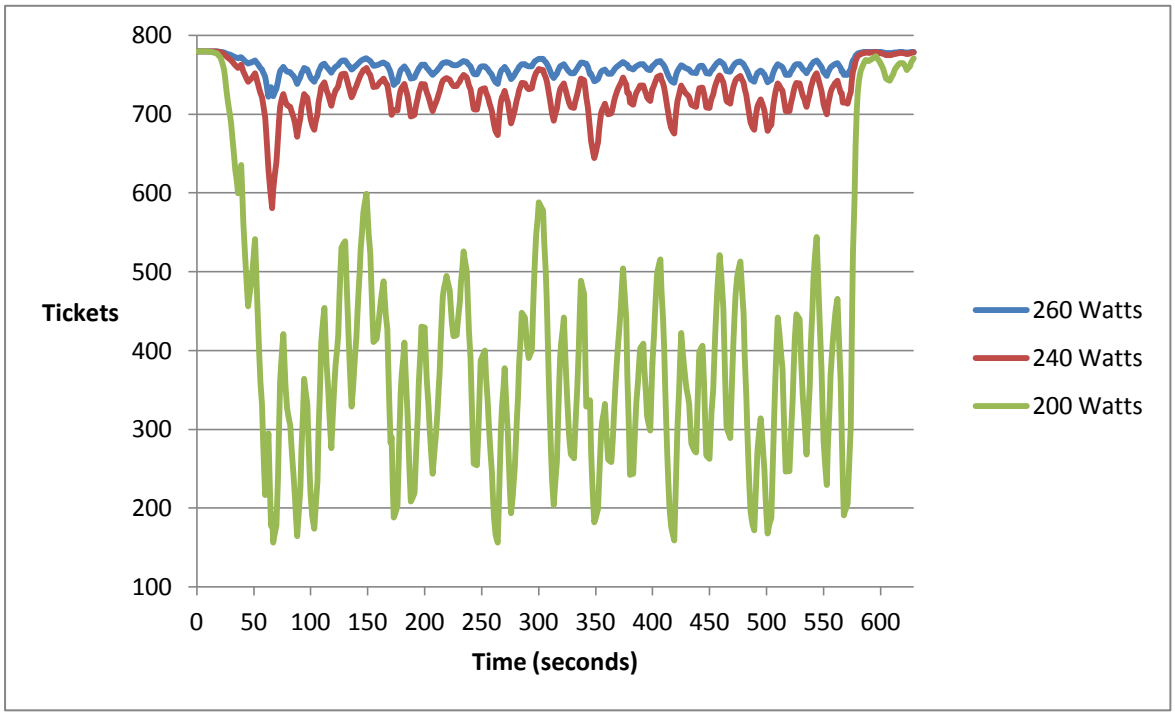

Figure 9: Behavior effect on available tickets during execution.

The applications will be assigned tickets per watt such that they all have a base 780 tickets. This will ensure that the determining factor in frequency of being chosen as victim is the behavior modifier. During the simulation, the number of tickets an application is allowed for lotteries is recorded at every interval, even when no power emergency is occuring. As shown in Figure 9, the server with 200 watts allocated finds itself frequently misbehaving which in turn results in less tickets during the application's execution. Meanwhile the server allocated 260 watts rarely comes close to using its full allocation, resulting in a fairly level amount of tickets available for lotteries.

Similar to the previous example, this simulation ran a number of times in order to observe the victim frequencies. Figure 10 shows that the 200 watt application is chosen as the victim much more frequently than the other two servers. Conversly, the 240 watt server rarely misbehaves and thus is chosen as a victim about as much as the server that never uses more

power than allocated. Due to the wide varience in behavior and resulting available number of tickets, a calculation of the expected probabilitiy is not considered.

These two simulations show that the inverse lottery and ticket allotments seem to be working as intended. Applications with fewer tickets are chosen more frequently as a victim to alleviate power emergencies. Applications that start to use more power than they are allocated have their tickets diminished, resulting in more frequently being chosen as a victim.

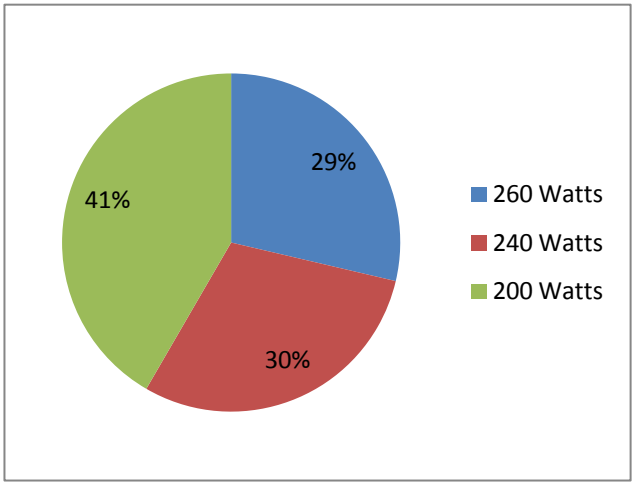

Figure 10: Victim frequency as function of behavior.

4.5 Ideal Configuration

For a final configuration, the tightest responsible allocation of a rack with three servers, one simulating map reduce, another simulating web search, and the last simulating web crawl, is considered. In addition, UPS batteries are utilized at all levels within the hierarchy. The rack is capped at 490 watts, which provides just enough power for all three applications under the highest possible DVFS throttling. The map reduce server is capped at 230 watts through its power source, web search server at 140 watts, and web crawl server at 160 watts. The UPS batteries are configured to be capable of providing all power needs for four minutes in an outage. The batteries are set such that only 10% of the total available charge can be used to alleviate power emergencies, although that much is never utilized in this simulation. This respects the

expectation that most of a UPS batteries charge is reserved for actual outage emergencies rather than power cap emergencies.

As shown in Figure 11, the simulation was able to achieve the desired power caps successfully. Furthermore, the batteries are more than capable of handling all power emergencies so DVFS is never used during the simulation. Battery discharged is signified by sudden spikes downward in power usage, especially below typical idle power. If a longer power emergency occurred DVFS would eventually be used to handle the emergency if batteries were fully discharged.

Figure 12 shows the state of charge of the batteries during the simulation's execution. The rack battery policy is such that it is only used for power emergencies over a certain threshold, otherwise delegating small emergencies to a server's battery. By the end of the simulation all batteries have been fully recharged.

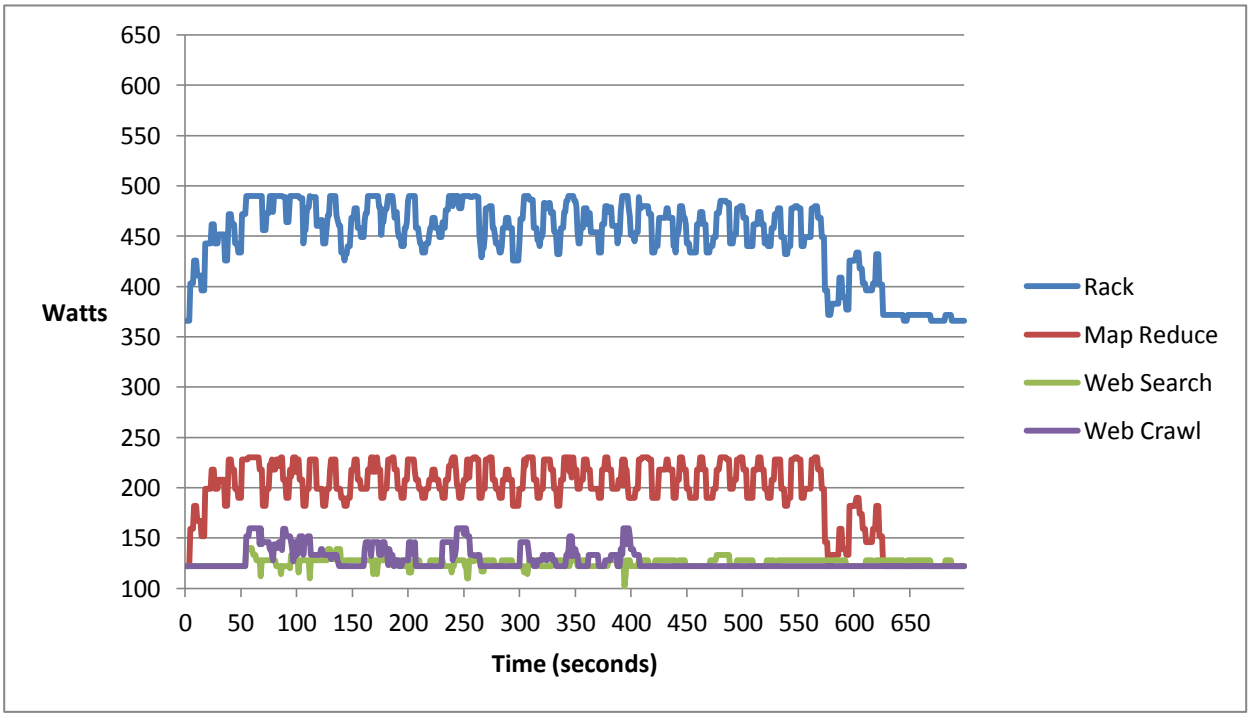

Figure 11: Power usage during execution of battery supported configuration.

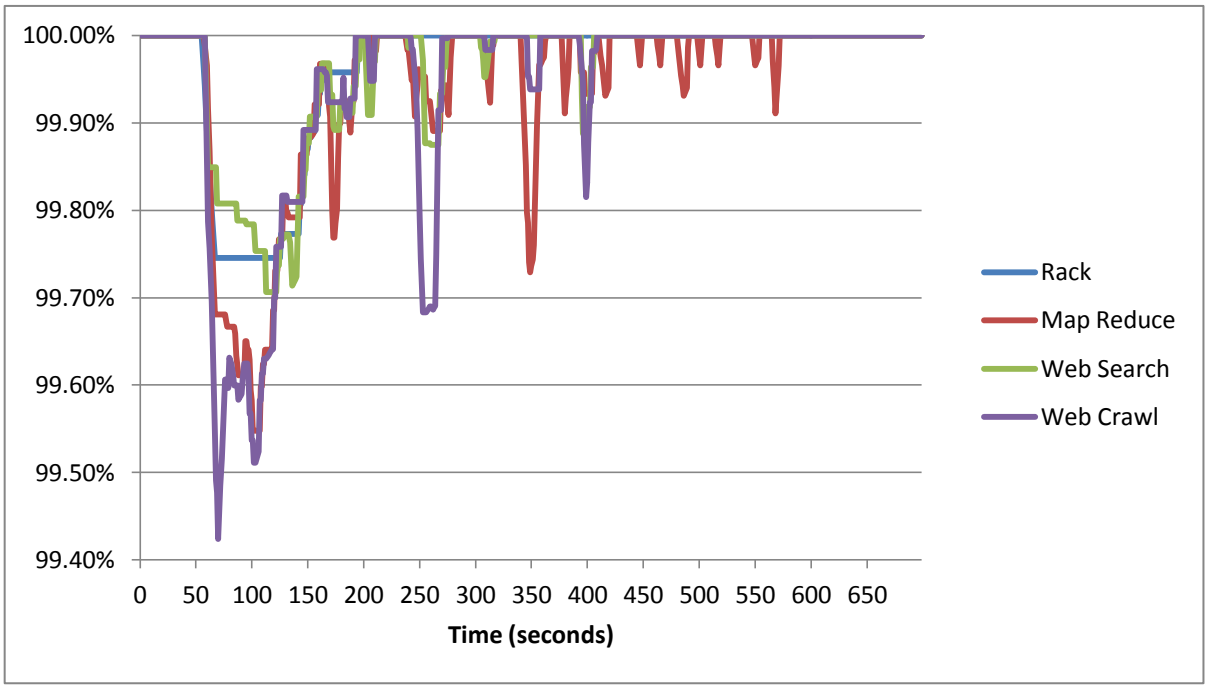

Figure 12: State of UPS battery charge.

This overbooked configuration has much lower peak power requirements as well as more consistent power usage than the originally presented configuration with no overbooking. Thus, this simulation shows an example of an ideal configuration for running the three different applications on the same rack. The total power provisioned is reduced, performance is maintained, and only a marginal portion of the UPS batteries are used.

4.6 Discussion

Overall, the simulation indicates that the software implementation works as intended, monitoring and coordinating the use of power across a power hierarchy. Specifically, the implementation properly honors ticket allocations and behavior considerations for victim selection. In addition, the implementation is capable of coordinating a number of power management knobs such as DVFS and UPS batteries, as well as capable of having a node higher

in the hierarchy delegate a power emergency to a child node. This establishes a solid starting ground for virtualizing power across a power hierarchy.

That being said, there are a number of limitations to the evaluation of this implementation. There is no optimal power management algorithm to compare the implementation against, so it is hard to evaluate the efficiency of this implementation. In addition, there are limitations in the evaluation due to the nature of simulation. Simulation by its very nature is an assumption, and will not be entirely predictive of actual deployment. Furthermore, the models on which this simulation was established should also be scrutinized.

As stated previously, the simulations presented make a number of assumptions. The applications are simulated in a very simplistic manner, and the power profiles used are fairly short and lacking variation to be highly reflective of a power profile from an actual data center. The evaluation would benefit greatly by more robust power profiles of greater length and variation. The way in which the batteries are assumed to discharge and recharge at fine granularity is fairly unrealistic, and in a more robust simulation should be confined to discharging and recharging at certain rates.

By far the largest area of concern is how control across the data center is maintained. The presented simulation behaves as if the power coordination is instantaneous. In an actual deployment, considerations about delay communicating across the network would need to be taken into account. Without a more thorough examination of delay in communication and reaction to new policies, it is hard to determine whether the presented software implementation runs fast enough to be effective. It may be the case that due to network delay performing highly coordinated power management is unrealistic. The means in which victims are selected is also rather complex and may cause a bottleneck in execution. Finally, the uncertainty about updating

power policy across the network may require more conservative estimates when planning for power usage in future intervals.

Such considerations were not considered both for their complexity and the difficulty in providing a model that would accurately generalize to most data centers. How power is managed across the network, what applications are in use, and what UPS battery configuration is available can be highly specific to an individual data center deployment. As such, the provided simulations are considered accurate indications of the possibilities of power virtualization. It is left for future work to either improve the simulations' models or extend the framework into actual deployment.

5. Conclusion

Power is a significant expense within data centers both in terms of the infrastructure deployed to provide power as well as the demand charges incurred for the peak power used during metered intervals. In order to reduce these expenses, it is beneficial to overbook applications or under provision the power hierarchy, which better utilizes the total peak power usage of the data center. There is a small chance that the applications within the data center will try to use more power than the data center can actually provide. In these rare power emergencies, a number of techniques such as DVFS, migration, and UPS batteries may be employed to temporarily shave off power needs and meet established power caps.

To simplify the process of managing power across a power hierarchy, virtualization similar to virtual memory is proposed where abstraction is used to provide applications their occasional spikes in power usage while controlling the limited power available. In particular, this implementation establishes a data representation of the power hierarchy that can be used to monitor and maintain the use of power across the data center.

In addition, it considers the task of how to reassign power usage during a power emergency in great detail. It does so by establishing an inverse lottery that allows a node experiencing a power emergency to fairly delegate a portion of its power deficit to one of its children. To provide notions of priority and proportional allocation, applications receive a base number of tickets based on the power they have been allocated and the number of tickets per watt they are allotted. This base number of tickets is then modified by an applications behavior,

wherein an application that uses as much or more than its allocated power will have its number of tickets reduced, which causes it to be chosen as a victim with higher frequency.

The implementation was then examined through a number of simulations based on power usage recorded from an execution of a map reduce, web crawl, and web search application. These simulations demonstrated that the implementation is capable of coordinating a number of power knobs to effectively achieve desired power caps. In addition, the simulation demonstrated that the inverse lottery and behavior algorithms work as intended, causing nodes that misbehave or have less priority to be chosen as victim with greater frequency.

As data centers and their associated power hierarchies continue to grow in size and complexity, efficient power use will become increasingly important. Having a virtual abstraction of the power hierarchy is a useful way to simplify managing power while simultaneously allowing more aggressive overbooking and power savings. This implementation demonstrates how such a virtual framework might be implemented while still providing performance and quality of service mechanisms that would not be available without maintaining a robust software abstraction.

5.1 Future Work

As previously mentioned, future work could be based off of this software implementation in a number of ways. The most ideal would be extending the framework for use in an actual deployment. This would most accurately examine the efficacy of software managing power across a hierarchy. Such a deployment would consist of extending the provided classes to use a

preferred protocol such as SNMP or SSH to remotely control the necessary knobs for maintaining power usage.

Another reasonable next step would be improving upon the models used for simulation to more robustly consider factors within a data center. Specifically, it would be greatly beneficial to begin considering network communication delays that would complicate the coordination of the power hierarchy. This would require improved assumptions and expectations on managing the power hierarchy.

An alternative direction would be implementing other types of power management into the framework, such as more robust scheduling or migration. For example, specifications of energy usage of applications could be established that would allow the framework to perform robust scheduling as discussed in related works. This would allow the examination of simulations across much larger periods of time.

Altogether, the software implementation provides a groundwork that provides a number of desirable properties for power virtualization, such as a data representation of the power hierarchy, a means to fairly reassign power between nodes, and functionality to control power usage at nodes within the hierarchy. This virtualization makes it easier to optimize power usage within a data center resulting in lower power costs.

References

[1] J. Hamilton, “Internet-scale Service Infrastructure Efficiency,” 2009.

[2] L. A. Barroso and U. Hölzle, “The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines,” *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, Jan. 2009.

[3] W. P. Turner and J. H. Seader, “Dollars per kW plus dollars per square foot are a better datacenter cost model than dollars per square foot alone,” *Uptime Institute White Paper, Santa Fe*, 2006.

[4] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, “Benefits and limitations of tapping into stored energy for datacenters,” in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, 2011, pp. 341–351.

[5] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger, “Modeling the relative fitness of storage,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 37–48, 2007.

[6] X. Fan, W.-D. Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 13–23, 2007.

[7] R. Bianchini and R. Rajamony, “Power and Energy Management for Server Systems,” *IEEE Computer*, vol. 37, no. 11, 2004.

[8] A. Gavrilovska, K. Schwan, H. Amur, B. Krishnan, J. Vidyashankar, C. Wang, and M. Wolf, “Understanding and Managing IT Power Consumption: A Measurement-Based Approach,” in *Energy Efficient Thermal Management of Data Centers*, Y. Joshi and P. Kumar, Eds. Springer US, 2012, pp. 169–197.

[9] J. D. Davis, S. Rivoire, and M. Goldszmidt, “Star-Cap: Cluster Power Management Using Software-Only Models,” Microsoft Technical Report, Oct. 2012.

[10] R. Nathuji and K. Schwan, “VirtualPower: coordinated power management in virtualized enterprise systems,” in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, New York, NY, USA, 2007, pp. 265–278.

[11] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, “Virtual machine power metering and provisioning,” in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 39–50.

[12] J. Stoess, C. Lang, and F. Bellosa, “Energy management for hypervisor-based virtual machines,” in *Proceedings of the USENIX Annual Technical Conference*, 2007, p. 1.

[13] H. Lim, A. Kansal, and J. Liu, “Power budgeting for virtualized data centers,” in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, Berkeley, CA, USA, 2011, pp. 5–5.

[14] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, “Power routing: dynamic power provisioning in the data center,” *SIGPLAN Not.*, vol. 45, no. 3, pp. 231–242, Mar. 2010.

[15] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, “Energy-aware server provisioning and load dispatching for connection-intensive internet services,” in *Proceedings*

of the 5th USENIX Symposium on Networked Systems Design and Implementation, 2008, vol. 5, pp. 337–350.

[16] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew, "Greening geographical load balancing," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 2011, pp. 233–244.

[17] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, "Making scheduling' cool': temperature-aware workload placement in data centers," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2005, pp. 5–5.

[18] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 103–116, Oct. 2001.

[19] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 conference on USENIX Annual technical conference*, Berkeley, CA, USA, 2009, pp. 28–28.

[20] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam, "Profiling, Prediction, and Capping of Power Consumption in Consolidated Environments," in *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008*, 2008, pp. 1–10.

[21] J. Stoess, C. Klee, S. Domthera, and F. Bellosa, "Transparent, power-aware migration in virtualized systems," *Proceedings GI/ITG Fachgruppentreffen Betriebssysteme*, 2007.

[22] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," in *ACM Sigplan Notices*, 2009, vol. 44, pp. 205–216.

[23] O. Bilgir, M. Martonosi, and Q. Wu, "Exploring the potential of CMP core count management on data center energy savings," in *Proceedings of the 3rd Workshop on Energy Efficient Design (WEED)*, 2011.

[24] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, 2009, pp. 157–168.

[25] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No 'power' struggles: coordinated multi-level power management for the data center," in *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, 2008, pp. 48–59.

[26] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 75–86.

[27] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy, "Energy storage in datacenters: what, where, and how much?," in *ACM SIGMETRICS Performance Evaluation Review*, 2012, vol. 40, pp. 187–198.

[28] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. Simunic Rosing, "Managing distributed ups energy for effective power capping in data centers," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, 2012, pp. 488–499.

[29] D. F. K. Qing Cao, "Virtual Battery: An Energy Reserve Abstraction for Embedded Sensor Networks.," pp. 123–133, 2008.

[30] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "ECOSystem: Managing energy as a first class operating system resource," in *ACM SIGPLAN Notices*, 2002, vol. 37, pp. 123–132.

[31] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, 2008, pp. 101–110.

[32] G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: A System for Energy-Efficient Management of Virtual Machines," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 1, pp. 6:1–6:27, Nov. 2010.

[33] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," in *Compilers and operating systems for low power*, Springer, 2003, pp. 75–93.

[34] D. Wang, C. Ren, and A. Sivasubramaniam, "Virtualizing Power Distribution in Datacenters," presented at the International Symposium on Computer Architecture, Israel, 2013.

[35] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: flexible proportional-share resource management," in *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, Berkeley, CA, USA, 1994.

[36] Apache, "Hadoop." [Online]. Available: http://hadoop.apache.org/.

[37] Apache, "Nutch." [Online]. Available: http://nutch.apache.org/.

[38] Apache, "Solr." [Online]. Available: http://lucene.apache.org/solr/.

[39] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 37–48.

ACADEMIC VITA

Paul Bily

paul.l.bily@gmail.com

Education

B.S., Computer Science, 2013, Penn State University, University Park, PA

B.S., Mathematics, 2013, Penn State University, University Park, PA

M.S., Computer Science and Engineering, 2013, Penn State University, University Park, PA

Professional Experience

Cisco Systems, Intern, Summer 2011

Applied Research Laboratory, Intern, Summer 2010 – Spring 2011

Quest Diagnostics, Intern, Summer 2009

Organization Membership

International Game Developers Association – Student Member

Penn State Game Development Club – Treasurer