Boosting Data Center Performance Through Non-Uniform Power Allocation *

Mark E. Femal Vincent W. Freeh
Department of Computer Science
North Carolina State University
{mefemal,vwfreeh}@ncsu.edu

Abstract

Data center power management is evolving from ad hoc *methods based on maximum node power usage to systematic methods that employ power-scalable components. In addition, it is possible to exploit the power and throughput relationship to increase the total work performed and safely overprovision the rack space while staying below an aggregate power limit. This research describes a general framework for boosting throughput at a local level while load-balancing the available aggregate power under a set of operating constraints. Our solution is useful for those data centers that cannot expand the number of power circuits or seek effective usage of their available power budget due to unplanned power fluctuations. The framework is particularly well suited for environments with a heterogeneous workload and hence, a non-uniform power allocation requirement. Based on a representative workload for a two minute period, this paper shows a non-uniform power allocation scheme increases throughput by over 16% versus a uniform power allocation mechanism.*

1. Introduction

The tremendous increase in computer performance has come with an even greater increase in power usage. As a result, power consumption is a primary concern. According to Eric Schmidt, CEO of Google, what matters most to Google "is not speed but power—low power, because data centers can consume as much electricity as a city" [23]. This does not imply speed is not important for computing centers, but more and more sites find themselves operating with a power constraint due to increased performance. Such a limit might exist due to either a limited power supply or heat dissipation and removal capacity. In addition, reducing the metered energy or associated power and cooling infrastructure costs might be a high priority. Regardless of the reason, a power constraint is a *performance-limiting* factor.

Many data center operators utilize manually intensive methods that are prone to error to avoid exceeding power circuit capacity. A conservative power management approach is to ensure the maximum power consumption of all nodes never exceeds the power circuit capacity based on a global limit, G. In such a cluster, this conservative approach first defines the maximum power a single node might consume, L_{max}. Data center personnel then subsequently deploy as many similarly configured nodes as possible under the global limit (*i.e.*, $n = \lfloor \frac{G}{L_{max}} \rfloor$). In general, the maximum power consumption of a node is much more than the average power consumption. Therefore, the conservative approach under utilizes power and artificially lowers the limit to $n \cdot L_{avg} < G$.

One can *overprovision* by deploying $m > n$ nodes; however, unsupervised methods risk exceeding the global limit. Exceeding the limit will likely trigger a reaction that reduces power consumption. This reaction could be as drastic as a circuit breaker tripping or less severe as to require other manual intervention in the recovery process. Both of these situations are undesirable, but are typically the normal plan of action in many environments. Therefore, we have developed a mechanism to facilitate safe overprovisioning that automatically controls power usage while avoiding excessive, long-term power consumption [10].

This paper presents improvements to our prior work, as well as a novel, dynamic algorithm to manage power *globally* in server clusters with non-uniform workloads.

* This research was supported in part by an IBM UPP award.

This extension performs *power load balancing* to efficiently utilize available power given the circuit capacity. Because power consumption is irregular, as is task demand, one needs a dynamic and adaptive solution.

There are two autonomic managers within our framework. One manages a node's target power consumption, beneath a local power limit. The other component assigns this local power limit based on aggregate cluster workload and the global power circuit capacity. For simplicity, we confine the remaining discussion to managing instantaneous power. Because energy is power integrated over time, managing energy is merely managing average power. Similarly, heat generation is a function of energy consumption and can be addressed in a similar fashion.

This paper makes three contributions. First, it introduces the idea of power load balancing and motivates its necessity. Second, it establishes the need for and creates a global power allocation mechanism to assign power in order to achieve efficient application throughput. Finally, this paper presents refinements to our local power controller to ensure a node is assigned a local power limit indicative of its workload contribution in a cluster.

The remainder of the paper is organized as follows. The next section presents related work. Section 3 describes the overall model and Section 4 discusses the implementation. Section 5 presents our results and we conclude with a summary and discussion of future work.

2. Related Work

The case for a closer relationship between the operating system and power management is explored in [32, 8]. Flinn and Satyanarayanan [12, 13] show that coordination with applications can yield significant power savings. Dynamic voltage scaling (changing both frequency and voltage) to reduce power consumption is explored in [11, 15, 26, 29, 19]. In [21], forecast methods are used on a single node to minimize power consumption. Unlike this work, we choose to automatically change the forecast model based on past prediction efficiency as well as increase throughput subject to power limits.

Power management in commercial servers is important for web servers [4, 22]. Much of this work relies on load balancers to distribute work. An investigation of load balancing was done in [27, 28] to turn cluster nodes on or off based on load. Additional research has also been done by Elnozahy *et al.* [9] for developing mechanisms for energy-efficient clusters using combinations of IVS, CVS, and VOVO policies. Although the VOVO policy is not considered in our initial implementation, its importance is less significant with heterogenous workloads. In [7], an economic approach is chosen to determine the minimal number of servers required to handle the load. Unlike [7] we favor a decentralized approach and seek to increase throughput. In [31], Sharma *et al.* applies real-time techniques to web servers in order to conserve energy and maintain QoS. Managing to service metrics is an instance of a target power allocation mechanism in our model once a power limit is defined.

The approach of estimating power consumption using performance counters is taken in [2, 20, 14] and is complementary to our notion of target power assignment. We are investigating the usage of program counters as a means of identifying intra-node performance bottlenecks. Identifying such occurrences may provide the opportunity for an additional power reduction in other power-scalable components with no loss in throughput.

In server farms, disk energy consumption is also important. One study of four energy conservation schemes concludes by stating that reducing spindle speed is the only option for clusters [6]. DRPM is a scheme to modulate the speed of the disk dynamically to save energy [16, 17] rather than stopping disk rotation. We plan to investigate this approach in future efforts.

While analyzing energy efficiency and operating points in [25], it is found that the most energy efficient gear is not always the lower performance point. All past power research for server clusters (*i.e.*, as in [5, 7, 9]) has focused on uniform workload distribution with migratable loads. This work has a contribution towards policy development in the management of power limits, but not all environments have the luxury of migrating work due to expense, complexity, or other factors.

3. Non-Uniform Power Allocation

With more than one node, power can be allocated to nodes non-uniformly. A uniform allocation is best only when the workload (and application performance) is identical on each node. There are four primary goals in global power allocation. The first goal ensures total power consumption is below the global limit and available power is equitably distributed (*e.g.*, no node starves). The second goal ensures each node receives or releases power according to expected changes in workload. The third goal is to allocate power effectively pro-

vided a degredation in the power supply occurs (*i.e.*, decreases due to partial power loss). The final power allocation goal ensures the solution is easy to deploy, maintain, and operate transparently with minimal tuning. To this end, a distributed algorithm is used to learn and react to cluster modifications automatically. No central point of failure exists and the per-node computational requirement is minimal. In addition, all current power consumption and minimum power requirements are available to administrators. This information offers a site the ability to identify power deficiencies and facilitates safe overprovisioning.

The global power allocation model dynamically and non-uniformly allocates power among nodes to increase aggregate performance given irregular workloads. To make this work in practice, time must be divided into discrete processing intervals. At time t, the data for past intervals is evaluated for the next interval. From this information, a forecast of expected work is calculated on each node. Given this knowledge, a prediction is made of expected need for each node and is then used in the next reallocation. Each node's expected need is weighed in conjunction with the expected aggregate workload. There are two power limits in our framework. The first exists to limit local node power consumption and is a calculated quantity based on cluster constraints. The second is the global limit and is assigned to a cluster based on circuit capacity. Each of these limits will now be covered in greater detail.

3.1. Local Power Limit

The calculated, locally assigned power limit, L, for each participant in the network is regarded as a mutual decision based on all node and global constraints (*e.g.*, power reserve). Using its assigned local limit, a node is responsible for suballocation of power at a fine-grain level. To ensure local minimum service constraints are met, L_{min} represents the power needed to guarantee a minimum service level. In addition, L_{max} is the maximum quantity of power a node uses due to technical limitations (*i.e.*, maximum consumption possible given connected devices).

At the node architectural level, each device in a node has an interface to relay or provide information related to power draw, performance states, as well as minimum and maximum power required to function. Although elements of this are available in laptop systems using the ACPI specification [18], future development of consistent interfaces to hardware should promote similar hardware sensors and functionality for servers using the same methods.

Each node manages average power consumption according to a target, T (a value less than L). Ideally, this average is close to the assigned local limit, but a burst term represented by B is used to reflect an additional quantity above T needed to properly manage average power consumption (there is a delay in reacting to power usage). The power consumption target is adjusted locally as needed and no restrictions are imposed in the general model (*i.e.*, this concept allows an energy conservation or other model to be employed). In general, the local node power relationship is governed by:

$$L_{min} \leq T + B \leq L \leq L_{max}.$$

Intuitively, based on the magnitude of $L - (T + B)$ a node signals to other nodes it has either a surplus or deficit. If this value is zero, a node is using its full allocation. If less than zero, a node has a power surplus. However, these conditions are not the sole means of reallocation. Additional consideration is needed when demand exceeds supply and this is discussed in the next section.

3.2. Global Power Limit

The global power limit itself is known and quantifiable based on circuit capacity. The allocation problem is to calculate L for each node i such that $\sum_{i=1}^{n} L_i \leq G$. Furthermore, given knowledge of the workload demand for each node, increase the efficiency in allocating available power for each node given the contribution of its work with respect to the aggregate demand, $W_{total} = \sum_{i=1}^{n} W_i(L_i)$. This problem is not solvable because work performed for a given power limit changes dynamically and is not known a priori. Therefore, we first allocate as much of the available power as possible to keep $G - \sum_{i=1}^{n} L_i$ small. Second, an estimate is performed for the work contribution of each node in the next interval. This estimate ensures nodes with greater need have a higher priority. Finally, a reassignment of all node local power limits is done for $t + 1$.

Based on the estimate of aggregate global power demand, if it is less than G the predicted aggregate power satisfies demand for the next interval. However, if the global power limit is exceeded a reduction in one or more node power limits is needed. Selected nodes lose power based on its expected change in workload and its associated contribution to aggregate demand in the next interval. In addition to insufficient allocation, if less

power is needed than available, nodes receive an additional allocation of power.

3.2.1. Preconditions. In order for the global allocation model to function, the following preconditions must be satisfied. First, a mechanism is required to manipulate hardware power consumption. In our current framework, this is accomplished using DVS. Second, a means of controlling power locally at each node must exist. This condition is met using a software component discussed in Section 4. Third, a system to forecast and track workload on all nodes must exist. In addition, an estimation of the expected change of workload for each node must be made. Finally, any available power must be allocated to ensure the global limit is not exceeded, yet nodes receive an appropriate value of additional power indicative of demand. The next two sections discuss the latter two requirements in greater detail.

3.2.2. Forecasting Workload. Given the unique characteristics of power and throughput that exist in a cluster environment, the general trend of work completed, $W_{average}$, is determined based on short-term historical demand. Using this same information, a predicted value of demand, $W_{predicted}$, is calculated and utilized to forecast demand for the next interval. Given current node power need, an approximation is now possible for the next interval using:

$$A = (T + B) \cdot \frac{W_{predicted}}{W_{average}}$$

The value A is bounded by several technical constraints. First, node power consumption cannot exceed L_{max}. In addition, a fundamental (if a node is to make any progress) or explicit service constraint establishes a power limit lower boundary, L_{min}. In instances where $G \geq \sum_{i=1}^{n} A_i$, all node limits can be assigned successfully with no node receiving less than its desired power limit. However, if this condition does not hold, some nodes lose desired power as a result of successive iterations of the algorithm. This node power loss depends on its power need with respect to all nodes. Any node receiving less power experiences increased local demand if its need increases faster with respect to other nodes in future intervals. This, in turn, subsequently causes additional local limit increases as the global power allocation method balances aggregate power based on demand throughout the whole cluster.

3.2.3. Additional Node Power. If available power is denoted as p, then

$$p = G - \sum_{i=1}^{n} A_i.$$

With p known, a node specific additional power allocation, S, is found based on work contribution, $c = \frac{W_{predicted}}{W_{total}}$. Thus, $S = p \cdot c$ and this value is added to the initial approximation A. In addition, S is bounded by an administrative limit to prevent excessive allocation. Intuitively, S is determined from the fractional amount of total available power after meeting all power need (*i.e.*, $\sum_{i=1}^{n} c_i = 1$). To further illustrate the determination of per node S values, consider an environment with two cluster nodes, an available power quantity of 20 watts, and an administrative threshold of 15 watts. If each were equally contributing to aggregate workload, the resultant per node S values are 10. However, if one node is idle and its contribution is zero while the other node remains busy, the busy node receives 15 watts and the other receives none. This example is simplistic because from an implementation standpoint, $W_{predicted}$ is restricted to nonzero values. This is to avoid dividing by zero in a completely idle environment as well as ensuring a minimum additional allocation is given to ensure short-term, upward mobility for T in the next interval.

3.3. Global Allocation Solution

The need to maximize the total power used in the cluster given the current and forecasted workload under a set of constraints is actualized as a Linear Programming (LP) solution. This is a multi-step process and is informally defined as follows:

1. Based on the broadcast data of all nodes, calculate the aggregate workload for the entire cluster, W_{total}.
2. For each node, determine its contribution, c, to aggregate workload. Changes in c will reflect a gain or loss of power from the prior interval.
3. Next, calculate the local power limit subject to all node L_{min}, L_{max}, B and T values.
4. For each local power limit, bound it according to its unique A value. This value accounts for all cluster demand and provides additional power to a node if needed.
5. Finally, ensure a sufficient reserve, R, is kept and the global limit, G, is not exceeded.

maximize:

$\sum_{i=1}^{n} L_i$

subject to:

$L_i \geq L_{min_i}$
$L_i \leq L_{max_i}$
$L_i \leq A_i + S_i$
$\sum_{i=1}^{n} L_i \leq G - R$

Figure 1. LP model to assign $L_i \; \forall i \in [1, n]$.

The informal steps outlined above are translated into the LP model shown in Figure 1. It is important to recognize this model is infeasible if $\sum_{i=1}^{n} L_{min_i} > G$. One policy to handle this occurrence is to perform a controlled shutdown of all or some nodes based on quality of service constraints, minimizing lost revenue, or some other measure. The current implementation does not address such an instance.

The LP objective function solution is the current total power allocation. In addition, the amount of power after allocations are made and the minimum power needed to meet all node constraints is determined, $\sum_{i=1}^{n} L_{min_i}$. However, the critical outputs are all node local power limits for the next time interval. Each node determines its local limit by saving its offset within the objective function as the model is built at runtime. It is important to notice that A accounts for the relative power need for each node (either surplus or deficit). Thus, each node indirectly decreases or increases L by adjusting T or B. In addition, the upwards pressure of both $W_{predicted}$ and the administratively assigned upper threshold set for S provide additional power to nodes.

Given the model in Figure 1, a linear programming solver [3] calculates the outputs using the simplex method. All values are considered to be real, continuous values within their respective bounds. The complete model is built dynamically using current cluster data. Rather than add additional constraints for L_{min} and L_{max}, lower and upper variable bounds on L are used to limit the number of true constraints. This reduces the internal model size and solves the problem more efficiently. There is no technical limit to the number of constraints (*i.e.*, only one additional solver constraint is needed for each node) and a timeout is used to generate suboptimal solutions.

4. Implementation

Two per-node autonomic managers comprise the core framework. Starting at the lowest level, device interface drivers provide the intelligence to determine and manipulate the state of a device in a cluster node. Each node aggregates multiple drivers into a cohesive entity referred to as the *Local Power Agent* (LPA). It is responsible for determining the power consumption target, given the local power limit. The LPA selects device gears to meet the power consumption target. A message queue is used by the LPA as the bridge to device drivers as well as for other external requestors, such as the next major component.

The second major software component of the framework is the *Global Power Agent* (GPA). The GPA is responsible for the coordination and interchange of related messages between nodes. It analyzes messages from the network and makes the appropriate requests to the LPA using the message queue. Communication between multiple GPAs is done based on a group identifier, subsequently referred to as a Power Management Group (PMG). The GPA learns the state of all other nodes by broadcasting to and receiving relevant information from all nodes in its PMG. There is not a one-to-one correspondence between a PMG and subnet; however, the current implementation limits PMG nodes to the same broadcast network. In addition to receiving state information from all other nodes, the GPA responds to other administrative control requests (*i.e.*, global limit changes).

The interaction of both the LPA and GPA is depicted in Figure 2. The GPA calculates and assigns the local power limit based on external information provided by all nodes in the PMG in addition to the knowledge of the global power limit. The LPA is responsible for ensuring the target power goal of an individual node is met as well as managing the target itself, subject to its local limit. Each of these entities is a separate daemon process and both are implemented as non-priviledged processes. Each of these components is now covered in greater depth.

4.1. Local Power Agent

The LPA is the mediator of all power-managed devices in a cluster node. It is responsible for maintaining a power target and listens for inbound messages destined for devices using the message queue. Each device has its inherent power characteristics coordinated with other devices by the LPA, to include changing device

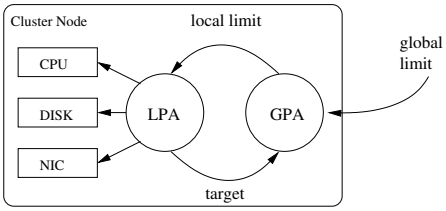

Figure 2. Relationship of the GPA, LPA, and power-scalable devices.

performance states to meet the target. The CPU is the major power consumer and is the initial focus in the LPA. Other devices, such as disks and network cards, might offer the ability to reduce power consumption, but the CPU is the only component that offers gears. Transitioning other devices (*i.e.*, disks) to an off state must be carefully weighed against idle time mispredictions and break-even points. This is not yet considered in our current implementation.

The default gear for the CPU is fastest and has the highest frequency and voltage setting. Because it is the default gear and all processors have a top gear, we denote this performance state as Gear 0. All other gears have less performance with lower frequency and voltage settings. Thus, the gear number increases as the frequency and voltage decrease.

The LPA determines the target power based on the assigned local limit. This derivation remains flexible to have different policies implemented depending on desired behavior. Two such sample policies include one based on load and another to optimize for a performance delay characteristic. This policy is not restricted to a single rule; a combination of rules could certainly be employed. In addition, as device-specific performance bounds are reached the LPA could reduce target consumption with no loss in overall performance (*i.e.*, if tasks are memory bound, a reduction in the CPU performance may be possible). The implementation of this effort is currently a work in progress.

The LPA manipulates device gears to meet the target system power set by its policy. To maintain the local power limit, the controller employs a predictor to determine the expected usage in the next epoch. We currently regard the local power limit as a *soft* upper bound on instantaneous power usage. A sampling window facilitates keeping system power consumption close to the target. To prevent excessive gear switching and allow stabilization, a minimum time between changes is enforced. This delay also helps manage the differing capabilities of devices and their subsequent ability to transition to different performance settings in a specified time interval. The core LPA controller uses a PID algorithm [24] to meet the power target and is discussed in [10].

4.2. Global Power Agent

Each node's GPA assigns the local power limit, L, based on information received from all nodes (including itself). This limit is calculated using knowledge of the administratively defined global limit, G. This global limit is assigned to all PMG nodes with a support tool. For reliability and scalability, each node's GPA is responsible for determining its respective L. Although explicit trust exists for well-behaved nodes, this precondition should be acceptable in most managed environments. All nodes in the PMG are synchronized by periodic UDP broadcasts. Nodes are added or removed from the subnet with corresponding changes done automatically to local power limits in the next time interval. Because the algorithm is shared on all nodes, each must have a notion of the current state of all other nodes. However, the only relevant output of the global allocation algorithm for the LPA is L.

All relevant LPA-GPA shared data structures are contained in memory managed by the LPA. Thus, the GPA is restartable with no immediate adverse impact; however, the state of the cluster still depends on the data sent and received by the GPA. A GPA that restarts does not make any local power limit changes while it relearns cluster state. The global allocation mechanism considers its restart as either a new cluster addition or an update to the last state of the node based on an administratively defined retention time. Network disruptions can hinder the algorithm from proper operation. In such instances, there are likely other prerequisite recovery steps and notification systems in place. Given a catastrophic network disruption, the last known state of all cluster nodes is used during the retention period. So in this case, reallocation of power is unlikely to significantly differ as the state of all nodes only changes when new information is available. A more conservative policy is to transition nodes to their lowest power consumption state. To account for several short-term failures, the value for the power reserve, R, in global allocation can be increased.

4.2.1. Power Management Group. To allow for multiple logical assignments and allocations on the same broadcast subnet, a cluster identifier is configured for each GPA on daemon startup and is referred to as the

PMG. In normal operation, there is a one-to-one correspondence between the physical power circuit and the PMG.

The cluster data structure to manage the PMG is an AVL tree, so tree operations are bounded by $O(lg\ n)$ where n is the number of PMG nodes (non-member broadcasts are simply ignored). Another important facet of this structure is a consistent (*i.e.*, sorted) handling of processing in the allocation. A dedicated thread is responsible for receiving UDP packets describing the state of other nodes in the PMG as well as reacting to administrative requests (further explained below). This thread uses *select()* with a timeout to prune the tree based on the time stamp of the last broadcast received for a cluster node and a predetermined maximum broadcast retention value.

In the current implementation, the retention time is 30 seconds and the broadcast rate is once per second. It is important to note the trade-off in retention time. Removing data too soon might adversely impact the state of power allocation. For instance, a server may temporarily lose network connectivity yet still remains powered and connected to the circuit. In addition, having a retention time too large might impact efficient allocation. In normal maintenance or to complement other policies (*i.e.*, controlled shutdown if work is migrated to another node),servers are intentionally removed from the power circuit and should therefore increase the available power for other PMG participants. As a result, retention time should reflect an appropriate site policy.

4.2.2. Broadcast Messages. There are two types of broadcast messages sent to participants in a PMG. First, broadcast utilization data packets are sent containing a node's power and current workload information. The power data consists of L_{min}, L_{max}, B, and T. The workload information consists of the number of tasks running or runnable since the last broadcast. As the server workload increases, this number subsequently increases. The preceding data metrics can be easily modified or extended should the need arise.

In addition to broadcast data packets, administrative messages can be broadcast to all PMG nodes. Such notifications consist of modifications to the overall power limit as well as provisions for setting an immediate administrative limit for all nodes used by a support tool. Membership in a PMG is further refined to be either *active* or *passive*. In passive mode, broadcasts are sent and received as normal, but inbound administrative messages are ignored. In active mode, the node responds to administrative messages. This capability is exploited by a monitoring tool discussed in Section 4.3.

Workload data is fed into a dynamic programming solution that updates the most recent forecast for the next interval as new node data is received. The number of data points kept for statistical significance is configured at compile-time. Empirically, ten data points provide a reasonable short-term approximation and this is the minimum needed for the GPA to begin local power limit management. The current implementation is focused on short-term prediction, but a more comprehensive solution should account for hourly, daily, or longer trends. Multiple estimation models are used by the forecast algorithm simultaneously. At the point the node forecast is needed for the next time interval, a quick sort is done based on mean square error (MSE). The lowest MSE represents the best forecast.

4.3. Support Tools

The current implementation utilizes two support tools to send some messages and monitor cluster activity. The Agent Controller Tool (*agentctl*) provides a command-line interface to send messages to the Local and Global Power Agents. The Cluster Monitor Tool (*dashboard*) receives broadcast data and monitors the state of the entire cluster. Each of these tools is now discussed in greater detail.

4.3.1. Agent Controller. For administrative control of a given node, a tool exists to interface directly with the LPA (as does the GPA) through shared memory or by using the message queue. For remote requests, the tool communicates indirectly through the remote node's GPA using Remote Procedure Call (RPC). This tool faciliates setting an immediate and administrative local power limit for all nodes. In addition, it is the only tool available for broadcasting the global power limit to the Power Management Group (PMG).

4.3.2. Cluster Monitor. Similar to the handling of messages by the GPA, the distributed allocation algorithm is also utlized in a console, monitoring tool (*dashboard*). This tool utilizes the broadcast data to monitor the state of either a specific PMG or all nodes on a subnet. Per the cluster participation modes previously discussed, *dashboard* listens in passive mode. This tool displays the current state of the cluster and can ensure nodes properly adhere to the power allocation strategy.

Gear	Frequency (Mhz)	Voltage	CPU (watts)
0	2000	1.5	89
1	1800	1.4	66
2	1600	1.35	added
3	1400	1.3	added
4	1200	1.2	added
5	1000	1.1	22
6	800	1.0	added

Table 1. AMD64 3000+ CPU gears and power consumption.

	Execution Interval (t)				
	5	10	15	20	30
Minimum	0.3	0.3	0.4	0.4	1.1
Average	1.9	3.5	4.8	6.5	9.2
Maximum	12.7	17.3	16.1	17.9	18.1

Table 2. Minimum, average and maximum time (milliseconds) to determine the LP solution on one node with different execution intervals (seconds).

5. Results

To evalute the implementation a cluster of ten servers was built using frequency scaling processors. Each node consists of the following hardware: 40 GB Maxtor EIDE 7200 RPM disk drives, ASUS K8V motherboards (onboard 1Gb NIC), 1 GB of PC3200 DDR SDRAM, and an AMD64 3000+ CPU. All nodes were interconnected on a dedicated 100 Mb switch. The entire cluster used the Linux 2.6 kernel. For frequency and voltage scaling, the AMD PowerNow *cpufreq* module was used. Modifications were done to this module to augment the ACPI device tables from the BIOS. These modifications to add performance states, along with the original settings, are shown in Table 1. The CPU power usage in this table is from [1].

For system power measurements, two digital multimeters (DMMs) were connected to serial ports on a non-cluster server. Custom software was created to communicate with these DMMs located on this host across the network. Each meter was inserted serially in the main power line of a node to measure amperes. Thus, a maximum of two nodes could be measured concurrently. To calculate power, a fixed supply voltage was used after first measuring voltage with one meter and obtaining little deviation (3% maximum). A TCP-based request server was created to allow the measured node to query power usage as needed. The overhead of network access is relatively small and a mechanism such as this was needed due to a lack of on-board hardware sensors on each node. Isolating the measurement activity from the measured node helps reduce the inaccuracy associated with evaluating the efficiency of developed software components.

To quantify the efficiency and performance of our solution, a series of low-level benchmarks was performed. The first reflects the end to end cost of switching to different gears. This cost was measured by first constructing a kernel module that utilized the *cpufreq* notification mechanism to measure the internal kernel cost for state changes. Our implementation cost was measured using *agentctl* to control gear changes. The measured values from the LPA ranged from 23-363 microseconds. A gear change to either one immediately above or one below from the current was typically the most efficient change when considering the end-to-end cost. This confirms the incremental model used to maintain the target power in the LPA. Even so, the maximum overhead imposed by the LPA is negligible and in the worst case, represents only 6.5% of the total time cost to switch gears using *cpufreq*. This module handles the low-level architectural details to control the frequency and voltage changes in the processor. Note that there is additional overhead to connect and disconnect from shared memory by *agentctl* that added between 164-242 microseconds. The GPA only incurs this cost once when first started.

Broadcast UDP packets sent by each node are 152 bytes. This includes a fixed size 20 byte area for the PMG identifer, adjustable at compile time if needed. In addition, packets contain extra argument type and size information so that low-level packet details can be verified for integrity and unpacked from the payload area. These packets incorporate all the current power usage and limit, minimum and maximum power requirements, target power consumption, and the latest workload data.

There is a corresonding cost, in time and increased node workload, to running the global allocation policy outlined. Measurements were taken on a single node to determine the minimum, maximum, and average time to solve the allocation problem during a two minute period (runtime interval divides 120). The single node results are shown in Table 2. Note that column headings reflect the time, in seconds, between processing runs of the algorithm. The more often the algorithm executes, the busier a node may become, but the execution run-

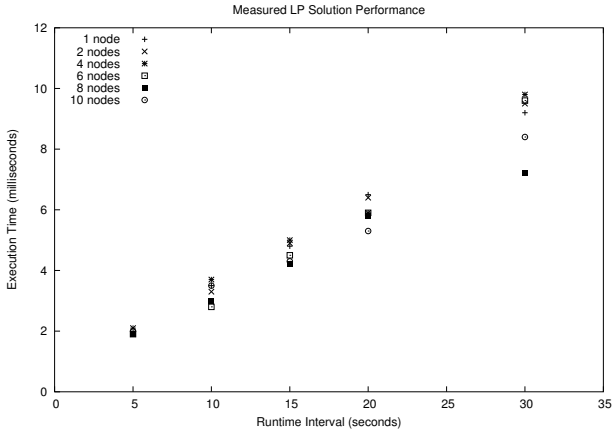

Figure 3. Execution run times to solve the allocation problem with increasing n.

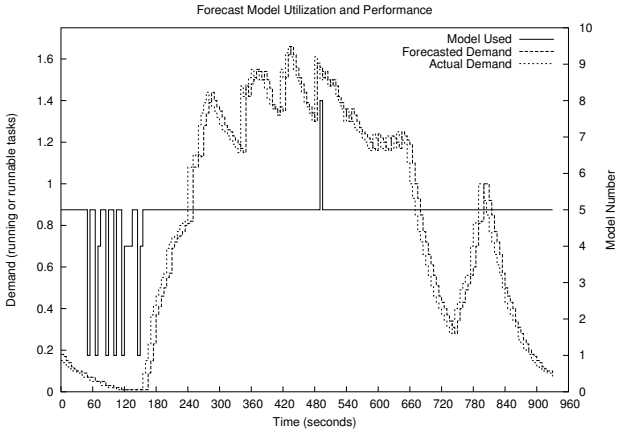

Figure 4. Forecast model utilization.

time decreases due to cache behavior. Running the algorithm more often allows adjustments to node power limits to meet demand more quickly. This, in turn, balances the power more efficiently and is the reason the five second interval is the default.

Intuitively, the general problem size grows as the number of nodes increases due to the additional constraints. In Figure 3, the same measurement method as the prior result was used but in addition, multiple cluster nodes were each configured with a GPA to broadcast data. There is a slight variation that grows as the runtime interval increases, but times are typically clustered near the same execution time. There is no reason to assume a cluster of 64 nodes is solvable in the 2 millisecond range, but the data does support lowering the interval to make the allocation mechanism as responsive as possible.

To consider the precision of the short-term forecast method, a series of ten models was constructed and workload data was fed into each of these models simultaneously. There were two model variants, one relied on a weighted average and the other used a moving average based on a last specified number of values. The different models were generated with either different weights or moving average window sizes. These models were then utilized for each inbound update from all nodes and a quick sort performed to find the model with the least mean square error when determining the new node local power limit. Figure 4 shows Model 5, a weighted average with $\alpha = 0.9$, was the most effective to both classify prior workload and make a prediction for the next interval. Notice that three other models were selected during this benchmark, although the total time they were used was small relative to Model 5. The global power limit was changed during this, and all other results, using the *agentctl* tool. In addition, the standard workload applied in this and remaining results was a gcc compiler run of the linux kernel. It is expected that changing the global allocation interval would likely impact the estimation model; however, the same model selection process would still choose the most applicable model dynamically.

Utilizing this same data, the effectiveness of the LPA in maintaining the limit on a single node is shown in Figure 5. For this and all other results, $L_{min} = 90$ and $L_{max} = 190$. These values were approximated using *cpuburn* [30]. Notice as G decreases and node workload exists, L subsequently decreases between 220 and 360 seconds into the run. The measured power usage exceeded L within three intervals due to a high power target and accumulated feedback error in the LPA. This is not a deficiency of the global allocation mechanism or policy, but does represent the most severe conditions that could occur when considering an LPA policy. It is also important to consider that L represents the next interval limit while usage is a measure of the last interval. To maintain the indicated power usage, the CPU gear utilization is shown in Table 3. The full spectrum was utilized with lower gears used when idle. There was a minimum of 5 seconds imposed between gear changes except when the current node usage temporarily exceeded L. The LPA reads current power usage several times per

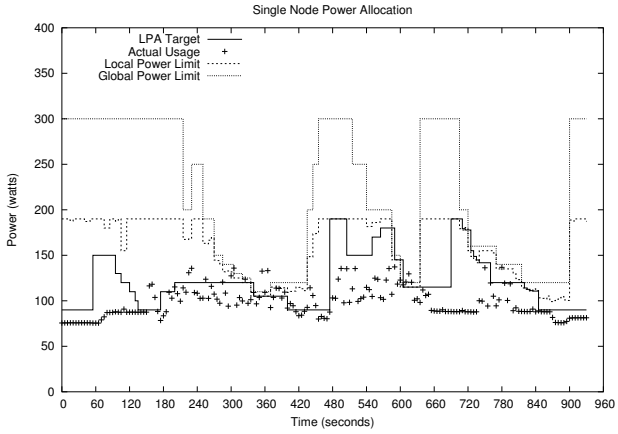

Figure 5. Single node effectiveness adherring to the local power limit.

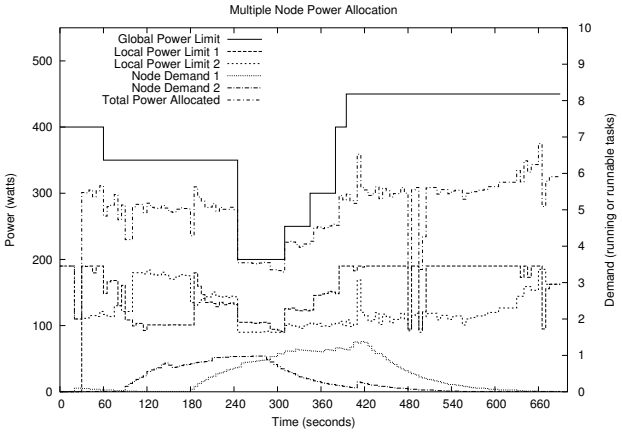

Figure 6. Multiple node power allocation with varying workloads and global power fluctuations.

Gear	Count	Time (msec)	%
0	29	578,863	58.88
1	42	92,344	9.39
2	19	51,213	5.21
3	11	70,152	7.14
4	9	19,046	1.94
5	8	39,152	3.99
6	4	132,275	13.46

Table 3. LPA state utilization performance.

second.

In Figure 5, notice that as recovery in G occurred near $t = 430$, the node power limit subsequently increased for the next interval. Near $t = 500$ and $t = 600$, the global power supply again forced node power reductions for both T and L. Because demand was high, we still try to boost throughput even with little difference between the global and local power limits. In this and all other results, global power reserve is $R = 5$. A site with a more conservative requirement could make this higher.

To quantify the results with multiple servers, two cluster nodes were used with $G = 400$ watts. The results, shown in Figure 6, reflect that even under high load the resultant aggregate local power limits of both nodes follows any degredation in the global power limit. When Node 1 first enters the cluster, at $t = 20$, it receives nearly the maximum allocation. Its limit subsequently decreases given its workload and no aggregate work to be performed. Notice that from nearly $t = 100$ to $t = 180$, the power limit for Node 2 is at L_{max} while Node 1 is idle with a limit of L_{min}. As demand on both nodes rises to approximately the same level, the node local power limits are balanced appropriately. This situation is later reversed when the demand on Node 2 falls while Node 1 continues processing. Later, as both nodes become idle and the global power supply increases, L on both nodes subsequently increases to slightly below L_{max}, with the aggregate limit well below G. This allows either node to quickly make use of available power as needed and rapidly react to additional processing requirements.

With non-uniform power allocation, it is possible to leverage the available power to increase throughput. To illustrate this benefit, two cluster nodes were configured with $L = 120$ watts each. The value for G was fixed at 245 watts (includes a 5 watt global reserve) so each node had an equal power allocation. Next, the standard load was applied to one of the nodes and the LPA was used to manage to the assigned local power limit. On both nodes, the GPA was configured to not change L and the results are shown in Figure 7. In addition, this same graph shows the result of running the GPA to manipulate the local node power limit according to our model. Notice the total power allocated, when the load is applied to one node, never exceeds G. In addition, the non-uniform

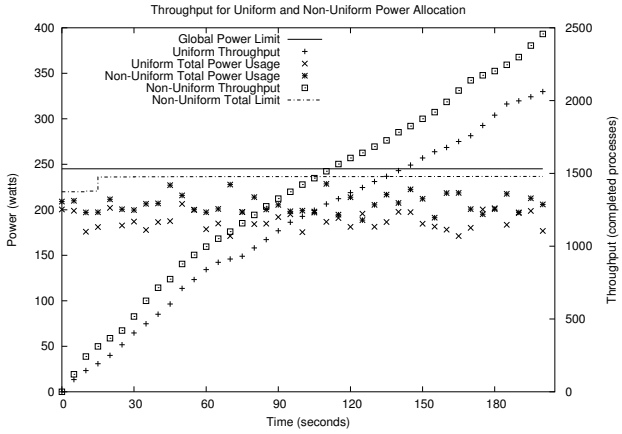

Figure 7. Throughput benefit using non-uniform power allocation.

power allocation scheme uses more than or equal to the power of the uniform method. However, from the start of the run, the node with load can easily transition to higher gears and increase throughput due to the higher local limit. The local power limit for the other idle node was forced to L_{min}. After just 200 seconds, the workload was terminated and the resultant throughput, in number of completed processes, shows a 16% gain using the dynamic, non-uniform power allocation method.

The throughput gain in the previous result is dependent on the particular application and the length of time needed to service normal demand. Long running processes, such as web or database servers, would experience a boost in throughput based on the relative workloads on each server (or multiple servers in the whole cluster). A data center with a hetereogenuos mix of applications, with non-migratable workloads, would experience gains if one server is busier than others during different time periods. This is the typical configuration at many hosting centers that provide managed, dedicated services for customers. Such configurations are typically used to address security, performance, reliability, or other customer requirements.

6. Summary

This paper investigates a non-uniform, automatic distribution of power for server clusters based on forecasted workload. Because the speed and performance of servers continues to increase, the additional power such servers consume must be accounted for in both the provisioning and financial planning processes. Power can be an automatically controlled resource within defined limits. Effective control yields greater utilization within the global limit as well as boosts application throughput. It is not always necessary to run a server at maximum performance. As a result, safe overprovisioning can occur by managing a cluster of servers to meet power limits.

A number of improvements are planned for the current implementation. First, a more robust local power mechanism based on multiple power scalable components is being developed. This controller will account for the power usage and performance of multiple devices. The second enhancement is to ensure a tight bound on the local limit. This is possible by forcing CPU idle time at a per-task level with the objective to increase local throughput. These throughput gains are possible by slowing down higher power consuming tasks. The final planned improvements are to synchronize simultaneous local limit changes and supplement the short-term forecast model with additional medium or long-term models.

We have presented a global power allocation mechanism based on both local and global power limits. The additional throughput gains possible from a strategy to manage power limits can increase the computational effectiveness of data centers that have non-migratable workloads, suffer from the inability to expand their power infrastructure, or seek an effective solution to transition from ad-hoc management methods. Our distributed and autonomic power control policy not only yields gains in throughput but also handles balancing the load across a heterogeneous mix of applications and architectures.

References

[1] AMD Athlon 64 processor data sheet. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs%/24659.PDF`, February 2004.

[2] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, September 2000.

[3] M. Berkelaar. Mixed integer programming solver. `http://groups.yahoo.com/lp_solve/`, January 2005.

[4] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The case of power management in web servers. In R. Graybill and R. Melham, editors, *Power Aware Computing*. Kluwer/Plenum, 2002.

[5] D. Bradley, R. Harper, and S. Hunter. Workload-based power management for parallel computer systems. *IBM Journal of Research and Development*, 47(5):703–718, September 2003.

[6] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *Proceedings of International Conference on Supercomputing*, pages 86–97, San Fransisco, CA, 2003.

[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[8] C. Ellis. The case for higher-level power management. *Proceedings of the 7th Workshop on Hot Topics in Operating Systems*, March 1999.

[9] E. M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Workshop on Mobile Computing Systems and Applications*, February 2002.

[10] M. E. Femal and V. W. Freeh. Safe overprovisioning: Using power limits to increase aggregate throughput. In *Workshop on Power-Aware Computer Systems*, Dec. 2004.

[11] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. In *Proceedings of the 7th Conference on Mobile Computing and Networking MOBICOM '01*, July 2001.

[12] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, pages 48–63, 1999.

[13] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, February 1999.

[14] C. Gniady, Y. C. Hu, and Y.-H. Lu. Program counter based techniques for dynamic power management. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture*, Feb. 2004.

[15] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.

[16] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Dynamic speed control for power management in server class disks. In *Proceedings of International Symposium on Computer Architecture*, pages 169–179, June 2003.

[17] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Reducing disk power consumption in servers with DRPM. *IEEE Computer*, pages 41–48, Dec. 2003.

[18] http://www.acpi.info. *Advanced Configuration and Power Interface Specification, Revision 3.0*. Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation, September 2004.

[19] C. Im, H. Kim, and S. Ha. Dynamic voltage scheduling technique for low-power multimedia applications using buffers. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.

[20] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.

[21] N. Kandasamy, S. Abdelwahed, and J. P. Hayes. Self-optimization in computer systems via on-line control: Application to power management. In *Proceedings of the 1st IEEE International Conference on Autonomic Computing (ICAC '04)*, pages 54–61, May 2004.

[22] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commerical servers. *IEEE Computer*, pages 39–48, Dec. 2003.

[23] J. Markoff and S. Lohr. Intel's huge bet turns iffy. New York Times Technology Section, September 29, 2002. Section 3, Page 1, Coumn 2.

[24] R. J. Minerick, V. W. Freeh, and P. M. Kogge. Dynamic power management using feedback. In *Workshop on Compilers and Operating Systems for Low Power*, pages 6–1–6–10, Charlottesville, Va, September 2002.

[25] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, pages 35–44, 2002.

[26] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *ISLPED 1998*, Aug. 1998.

[27] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proceedings of the Workshop on Compilers and Operating Systems*, September 2001.

[28] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In *Compilers and Operating Systems for Low Power*, September 2001.

[29] J. Pouwelse, K. LangenDoen, and H. Sips. Energy priority scheduling for variable voltage processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.

[30] R. Redelmeier. *cpuburn*. http://pages.sbcglobal.net/redelm/, June 2001.

[31] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron. Power-aware QoS management in web servers. In *24th Annual IEEE Real-Time Systems Symposium*, Cancun, Mexico, Dec. 2003.

[32] A. Vahdat, A. Lebeck, and C. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 31–36, 2000.