ABSTRACT

FEMAL, MARK E.:Non-Uniform Power Distribution in Data Centers for Safely Overprovisioning Circuit Capacity and Boosting Throughput. (Under the direction of Assistant Professor Vincent W. Freeh).

Management of power in data centers is driven by the need to not exceed circuit capacity. Such techniques are evolving from ad hoc methods based on maximum node power usage to systematic methods that employ power-scalable components. These components allow for dynamically controlling power consumption with an accompanying effect on performance. Because the incremental performance gain from operating in a higher performance state is less than the increase in power, it is possible to overprovision the hardware infrastructure to increase throughput and yet still remain below an aggregate power limit. In overprovisioning, if each component operates at maximum power the limit would be exceeded with disastrous results. However, safe overprovisioning regulates power consumption locally to meet the global power budget. This research work presents PICLE, the Power Infrastructure Controller for Limited Environments. This framework is designed for boosting throughput through intelligent monitoring of server clusters by load-balancing available aggregate power under a set of operating constraints. The solution is useful for data centers that cannot expand the number of power circuits or seek effective usage of the available power budget due to power fluctuations. The framework is also ideally suited for environments with a heterogeneous workload and hence, a non-uniform power allocation requirement. Synthetic benchmarks indicate overprovisioning throughput gains of nearly 6% from a staticly assigned, power managed environment and over 30% from an unmanaged environment. In addition, based on a representative workload for a two minute period, a non-uniform power allocation scheme is shown to increase throughput by over 16% versus a uniform power allocation mechanism.

Non-Uniform Power Distribution in Data Centers for Safely Overprovisioning Circuit Capacity and Boosting Throughput

by

Mark Edward Femal

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2005

Approved By:

Dr. Frank Mueller

Dr. Eric Rotenberg

Dr. Vincent Freeh
Chair of Advisory Committee

This work is dedicated to Lauree Lucille Duginski.
There are some people in life that lack its appreciation;
there are some that do not but aren't afforded its luxury.

Biography

Mark Femal was born on March 13th, 1971 and is originally from the State of Wisconsin, USA. After a five year commitment in the military, he received his Bachelor of Science in Computer Science from the University of Wisconsin at Oshkosh in 1998. He then worked for several years in the insurance and investment, internet service provider, and telecommunications industries before co-founding Beantree, Inc., an E-Commerce Application Service Provider. He began graduate study in the Fall of 2003 and with the defense of this thesis, will receive his Master of Science in Computer Science from North Carolina State University in the Fall of 2005.

Acknowledgements

I would like to thank Dr. Vincent Freeh for the opportunity to work with him to accomplish this research under flexible constraints. I would also like to thank Dr. Frank Mueller and Dr. Eric Rotenberg for agreeing to be on my committee. I wish to thank Chad Rosier for initial assistance on performance counter collection and Daniel Smith for his help in serial communication techniques. Special appreciation is extended to the Autonomic Computing Staff at IBM Research, namely, Patricia Rago and David Ogle for both the funding received and their insights as the work progressed. Finally, I'd like to recognize and thank the anonymous reviewers from both the Workshop on Power-Aware Computer Systems and the International Conference on Autonomic Computing.

Contents

List of Tables

List of Figures

Chapter 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" [48]. This does not imply speed is not important for computing centers, but more and more installations find themselves operating within power limitations. Such limits exist due to either a limited power supply or a heat dissipation and removal capability. 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 limits performance.

1.1 Motivation

The primary motivation of this research work is to increase throughput given defined power limits by increasing parallelism. High performance clusters such as BlueGene/L [1] make use of low-power, modest clock rate processors to provide more efficient performance with respect to energy consumption. A similar approach is taken with frequency scalable CPUs and off-the-shelf hardware in the creation of PICLE, the Power Infrastructure Controller for Limited Environments. Because the CPU is a dominant power consumer in many servers, it is consequently the initial focus in PICLE. In addition to the CPU's power consumption characteristics, there is a manufacturer commitment towards power conservation as exhibited in the ACPI Specification [37] and through Dynamic Voltage Scaling (DVS).

With DVS, it is well established the relationship of CPU frequency (F) and voltage (V) is given by [52]:

$$CPU\ \ Power = A \cdot C \cdot V^2 \cdot F$$

Where A is the activity factor for how frequently gates switch and C is the total capacitance at the gate outputs. Using this relationship and knowing $F\ \alpha\ V$, decreasing the CPU frequency causes a quadratic reduction in power. If performance is proportional to power, a reduction in performance has an accompanying effect on power. When transitioning from the highest to the lowest performance state, the power reduction typically grows more quickly than the performance reduction. If the aggregate power limit in a cluster does not change, one can increase the number of nodes when the cumulative power reduction on all nodes is sufficient to support another node. This power and performance relationship is further explored in Chapter 2.

1.2 Power and Overprovisioning

Many large data centers have a goal of managing instantaneous power consumption. Although energy conservation is important, the physical infrastructure of data centers is typically partitioned into a set of circuits. These circuits, allocated per rack, provide a maximium quantity of available power. As a result, data center personnel assign equipment to racks (provisioning) conservatively. Such management methods under utilize available power; however, exceeding the circuit power capacity causes disruption in service. To avoid this negative effect, managing instantaneous power consumption is often more important than reducing energy consumption [43].

Complicating the delicate balance of maintaining a safe upper bound on power consumption, significant variation occurs based on the state of connected equipment. A server that boots needs near its maximum rated amount of power. In contrast, lightly loaded and idle systems draw significantly less. An analysis of work performed and power consumed facilitates safe overprovisioning (adding more equipment to racks than otherwise possible given power circuit capacity). This benefit offers sites the ability to maintain peak performance within defined power limits.

The general concept of overprovisioning is not new in industry. For instance, equipment exists to monitor aggregate power usage on a circuit. In addition, such equipment handles disruptions by ensuring all connected devices are not powered on simultaneously [7]. This hardware provides a means of reacting to power failures and monitoring circuit health. By monitoring aggregate usage over time, additional nodes can be added to a circuit by administrators. However, this equipment is not an effective solution to ensure the power consumption of each node is adequately controlled with respect to the aggregate limit. This requirement warrants the design of a framework for controlling power in an intelligent, concerted fashion.

1.3 Contributions

This research makes several 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 non-uniformly in order to achieve efficient application throughput. Third, it creates an implementation to control power consumption within defined limits. Finally, it substantiates fine-grain task throttling as a means of reducing power consumption with no hardware support. All of these contributions are shown to be an efficient means of achieving power allocation for data centers with minimal operating system modification. The system is effective for both homogeneous and heterogenous architectures and it operates automatically, within proscribed constraints, with minimal tuning.

With PICLE, a measured overprovisioning result reflects at least a 6% gain in throughput while still remaining below an aggregate power limit. This performance gain accounts for a static analysis done to ensure cluster nodes operate at the best gear (*i.e.*, manual human intervention). In unmanaged environments, using the same test, the benefit is in excess of 30%. In addition, utilizing a representative workload for a two minute period, a non-uniform power allocation scheme is shown to increase throughput by over 16% versus a uniform power allocation mechanism. A uniform power allocation solution is only applicable for environments with workloads that are equally distributed (or can be made so with task migration).

The remainder of this thesis is organized as follows. Chapter 1 contains an introduction to PICLE and the problems it solves. In Chapter 2, the power allocation problem and the strategy for overprovisioning is further refined. Chapter 3 presents related work and differentiates this thesis from prior knowledge. Chapter 4 presents the PICLE design overview and policy to solve the power allocation problem. Chapter 5 discusses the mechanism and how it accomplishes the goals discussed in the allocation strategy. Chapter 6 illustrates the measured results and Chapter 7 outlines conclusions and future work.

Chapter 2

Power Limits

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

One can more aggressively deploy nodes; however, unsupervised overprovisioning 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 such as requiring other manual intervention in the recovery process. Both of these situations are undesirable, but are typically the normal plan of action in many environments.

There are several high-level goals to power limit management and overprovisioning. First, any solution must function transparently with existing applications. Additional application hints, as demonstrated in [6, 67] to save energy, might provide additional cluster throughput gains or better power limit management. A solution that requires modifications to applications to function is too restrictive. Second, any solution must also impose minimal processing overhead. Operating system modification and stability must be considered with the additional intent to provide flexible policies outside the kernel as needed. Finally, any solution must be easy to deploy, administer, and operate transparently with minimal configuration or runtime adjustment. The runtime characteristics of available power, aggregate usage, and other circuit metrics should be exposed to provide an administrative view useful for provisioning or other decisions. For instance, a history of power utilization can be used to determine the best time to perform maintenance to power infrastructure.

The aforementioned goals are complementary to energy conservation techniques. The need to manage energy in data centers is important and can play a role in assigning node power limits. However, in order to increase throughput a different policy is employed and explored in this thesis. The same flexible PICLE mechanism, as discussed in Chapter 5, is useful in environments with a greater emphasis on energy conservation.

2.1 Allocation Strategy

Power allocation is approached from a central notion of two power limits. First, a global limit exists as a result of technical or business reasons. The technical restriction is due to either a limited circuit capacity or an inability to dissipate heat. The business justification stems from reducing energy and infrastructure expenditures or as a result of site limits (*i.e.*, inability to add circuits in a high-rise office complex). The second power limit, the local limit, is a node specific limit. These power limit abstractions are not specific to servers, but cover any network enabled device connected to the same power circuit.

Figure 2.1 depicts the general strategy to overprovision circuits using power limits. In a given interval, the total power available to all nodes is constrained by a limit of 450 watts. Each node on the left executes unconstrained and consumes 150 watts while completing 1/3 of the total work target (each node operates at 100% of its capacity). In contrast, the overprovisioned nodes on the right are restricted to 1/4 of the power limit so consume 25% less power per node. If these nodes execute with a 10% penalty in original work contribution, a reasonable expectation, these nodes provide a realized throughput benefit of 20%. This example is presented for illustrative purposes, but it provides the incentive to manage limits and overprovision power circuits.

If L denotes the power limit (expressed in watts) of a node in a given performance

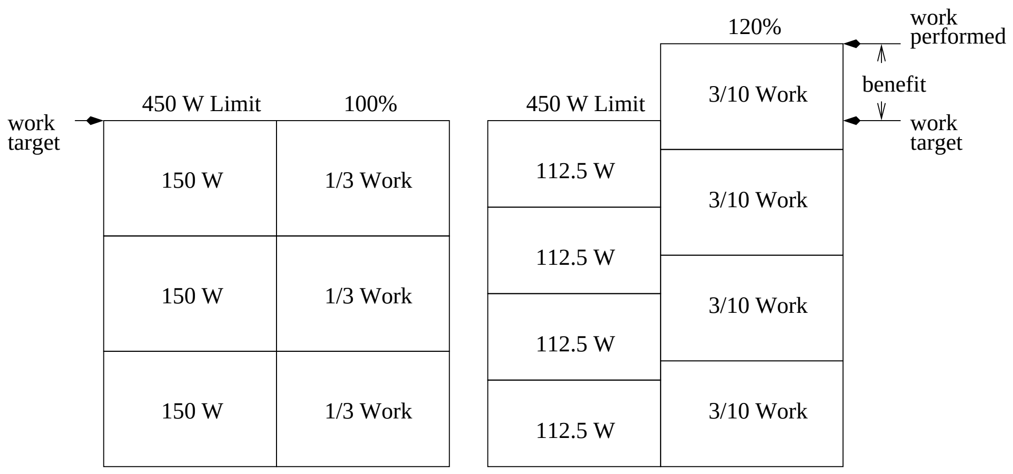

Figure 2.1: Managing power to a 450 watt limit based on a 10% reduction in original work and 25% reduction in node power.

gear, the total number of nodes on a power circuit with all nodes in the same gear is $n = \lfloor \frac{G}{L} \rfloor$. If a node uses all of the power limit L in time t, the energy in joules is $J = L \cdot t$. The work performed in t, as a function of energy consumed, is then represented by the function $W(J)$. As a consequence, total throughput of all nodes is $W_{total} = W(J) \cdot n$. The energy efficiency per node, *i.e.*, the useful work per unit watt over time t is, $E = \frac{W(J)}{J}$. The overall efficiency of n nodes is $E_{total} = \frac{W_{total}}{J \cdot n} = E$. Thus, the efficiency for a single node is useful to model a similarly configured cluster.

Modeling theoretical power usage with L is simplified above for clarity. In practice, with varying workloads and heterogeneous components the maximum, minimum, and average system power usage differs significantly. In addition, with multiple nodes and a non-uniform power allocation requirement it is possible for these nodes to be in different power states. However, a general relationship between L and $W(J)$ provides the impetus for increasing overall throughput in t. In general, as different gears are chosen to affect L, there is a work benefit when $\Delta L > \Delta W(J)$. With migratable workloads, the most effective solution is to evenly distribute G dynamically (power fluctuations change G), as indicated above, using an intelligent controller designed to not exceed L. Such a controller might also turn off some nodes for greater efficiency. In other environments that cannot migrate work, task demand and resulting power consumption is irregular so an adaptive solution to intelligently control power is needed.

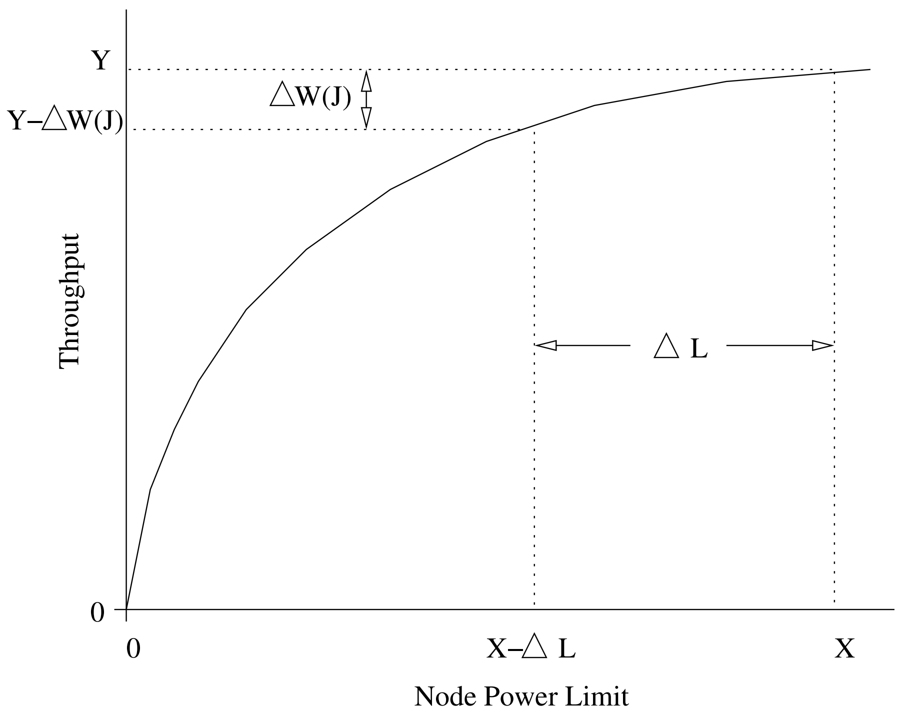

Figure 2.2: Ideal relationship of work performed with a provided power limit in a fixed time interval.

The previous relationship between work and the local power limit is idealized in Figure 2.2. As L increases from 0 to X, the associated work performed also increases from 0 to Y. However, this increase in throughput at a higher power consumption state is typically less than the change in energy needed. This figure also implies the overprovisioning relationship is only relevant when $\frac{d}{dJ}\ W(J) < 1$ when considering the slope of the tangent at point (X, Y).

2.1.1 Local Power Limit

The locally assigned power limit for each cluster participant (*i.e.*, server, router, switch, etc.) is regarded as a mutual decision based on global constraints. Given the global limit, each node is responsible for the suballocation of power at a fine-grain level. There are several goals to local power allocation. First, as the demand for power exceeds the local limit, no task starves as those with higher priority become favored. Second, operating system changes must be simple and flexible so they are viable to implement across multiple architectures. In addition, service constraints should be enforced locally to ensure adequate task progress is made as access to resources becomes limited. Finally, a solution to manage consumption can utilize the inherent power and performance scaling features available in devices. In addition, a robust solution might use these capabilities to complement a general strategy to manage to a limit even without such features.

Thus, each participant operates within a derived maximum local power limit and attempts to never exceed this limit within the defined time interval, t. It should be free to choose where to set its own target usage, which differs from actual measured usage, based on need, priority, or other relevant measures. If all participants know the target usage of all nodes, the ability exists to account for power need in a global allocation mechanism and reassign local limits for $t + 1$.

2.1.2 Global Power Limit

As each cluster node manages power consumption with respect to its target, it is possible for a global power allocation mechanism to account for each node's power need based on the relative difference between its target usage and current local power limit. This allows flexible policies to be deployed and facilitates a simple interface for global power allocation. To allocate power globally, the global limit itself is either assigned by administrative personnel or is dynamically provided by intelligent devices on the network. For instance, smart racks or uninterruptible power supplies might supply the information for autonomic operation.

There are several additional goals to managing power globally. The first goal is to ensure that no node starves. Second, nodes must release and receive power according to the defined mechanism fairly. The notion of fairness implies all nodes are well behaved. The final goal is to ensure that as global power decreases below aggregate need, the mechanism allocates available power effectively. Such allocation necessitates ensuring no node power allocation falls beneath the minimum needed to handle its processing requirements. This involves ensuring that excess power not utilized, yet potentially allocated to other nodes, is redistributed fairly to power deficient nodes.

2.2 Measurement Fluctuation

Managing a local power limit is subject to measurement capability and the error due to minute fluctuations. As instantaneous fluctuations occur in a node due to device power consumption, an optimistic strategy in software is to manage average power consumption for a small time interval. In addition, cost-effective measurement hardware is typically limited to a response time often in the range of one second. High-end measurement solutions are cost prohibitive and onboard sensors do not exist on most nonmobile computing devices. Such sensors, if added to servers and other devices, would likely still provide a limited resolution.

As power fluctuations occur due to both resource demand and measurement limitations, it is constructive to set bounds to ensure power limits are not exceeded. It is also not useful to utilize true power as a point in time indicator. To this end, hardware measurement solutions use root mean square (RMS) power to determine true power (*i.e.*, measured power using watt meters) in subsecond granularity. The measurement then reported by hardware reflects several thousand samples taken over the course of a second. So an internal means in hardware exists to remove a degree of fluctuation in very small time intervals. This abstraction is also useful to extend into upper layers of a power limit management solution and is further discussed in in Chapter 4.

Chapter 3

Related Work

The case for a closer relationship between the operating system and power management is explored in [64, 24]. Flinn and Satyanarayanan use goal-oriented feedback to manage battery life in portable systems [31, 29, 30]. A similar approach of using feedback is useful in PICLE to manage power consumption below a local power limit. Dynamic voltage scaling to reduce power consumption was explored in [28, 33, 56, 59, 39, 9]. In [41], forecast methods are used on a single node to minimize power consumption. Unlike this work, PICLE automatically changes the forecast model based on the efficiency of past predictions and seeks to boost throughput given available power.

A vast amount of research has been done for energy conservation (EC) on mobile platforms [69, 5, 36, 23, 44]. In addition, while analyzing energy efficiency and operating points in [50], the most energy efficient gear is determined to not always be the lower performance point. The role of EC is consistent with the general goal of efficient power allocation. Power represents the instantaneous energy used at a specific point in time and energy is power usage in a defined interval. The PICLE strategy uses a notion of a node power target. This power target could be derived from an EC policy; however, it is not a necessary condition.

The case for a Super Dense Server (SDS) was explored in [27]. Dense servers offer higher deployment density and lower power consumption. Using industry benchmarks, it is shown that such dense computing techniques outperform CPU-bound electronic commerce workloads by a factor of 2 for nearly the same space and power budget as traditional servers. In addition, results are similarly favorable compared to virtualizing a high-end server to handle scaled-down workloads.

As tasks execute and power usage subsequently increases, a solution must exist to throttle tasks to avoid exceeding the limit. Several different options exist for enforcing restrictions on tasks. First, all tasks can be slowed equally, but this negatively impacts aggregate performance. Second, groups of tasks can be slowed, but even this technique can penalize some tasks. Finally, each process can be slowed to reduce power consumption. This latter option offers the most flexibility and is a design choice in PICLE (in addition to using DVS). Resource containers are a common approach to limiting access to devices [10]. In addition, the concept of resource principals are shareable across all cluster nodes using resource containers on each node and a resource manager, as in [8]. Energy containers are proposed and used in [12, 66] to regulate temperature. Task monitoring is also proposed in [68, 47] to analyze utilization and perform a device shutdown when idle for both disks and network cards. Workload monitoring and the observer architecture are proposed in [13]. This framework is also used to estimate idle periods for dynamic power management. Additional work in [21] detects idle periods in disks using an adaptive learning tree and idle period clustering algorithm. All of this research complements the strategy in PICLE to perform task restrictions to manage power usage below a local power limit.

In server farms, both disk and memory energy consumption is important. DRAM based power management is explored in [22] using memory bank reference tracking in the operating system scheduler. Those banks not being used are subsequently transitioned to lower power states. For disk management, one study of four energy conservation schemes concludes by stating reducing spindle speed is the only option for clusters [17]. DRPM is a scheme to modulate the speed of the disk dynamically to save energy [34, 35] rather than stopping disk rotation (the latter is useful in mobile systems). Both memory and disk power management should be included in a comprehensive solution to manage power limits.

Power management in commercial servers is important for web servers [15, 43]. Much of this work relies on load balancers to distribute work or simulators to quantify the benefit. An investigation of load balancing was done in [57, 58] to turn cluster nodes on or off based on load. Additional research has also been done by Elnozahy *et al.* [26, 25] for developing mechanisms for energy-efficient clusters using combinations of Individual Voltage Scaling (IVS), Coordinated Voltage Scaling (CVS), and Vary-On/Vary-Off (VOVO) policies. Although the VOVO policy is not considered in PICLE, its importance is less significant given an inability to migrate work. In [19], an economic approach is chosen to determine the minimal number of servers required to handle a workload. Unlike [19], PICLE uses a decentralized algorithm and seeks to increase throughput. In [63], 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 policy in PICLE once a power limit is defined.

Research such as in [16] focus on uniform workload distribution with migratable loads. This work has a contribution towards long and medium-term trends in the management of power limits for PICLE. The approach of estimating power consumption using performance counters is taken in [11, 40, 32] and is complementary to the PICLE notion of target power assignment. In addition, using program counters to lower a node's power need is useful in the context 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.

Besides general program counter data, a correlation of instructions per cycle (IPC) and power is shown effective in [45, 46]. In a high performance processor, a dominant part of power consumption is due to exploiting instruction level parallelism (ILP). As the ILP increases, the processor becomes busier and there is a subsequent increase in power. However, as the main pipeline stalls this leads to a lower IPC and clock-gating, which reduces power consumption. A similar exploration of IPC and performance is done in [20, 65, 55, 53, 54, 60]. In addition, IPC is utilized in [38] for an architectural simulator for both dynamic energy efficiency and temperature management. In [42], Kotla *et al.* uses IPC as a means of predicting future performance to schedule frequency changes in a server cluster. A workload forecasting method is employed in PICLE; however, IPC is used solely to identify higher power consuming tasks. In addition, rather than seeking to just satisfy workload for the next predicted period, PICLE advocates increasing power to nodes to boost aggregate throughput.

Chapter 4

Theory and Design Overview

The general design approach to power distribution is encapsulated in the producer-consumer model. The power producers are regarded as the suppliers of power to various components. The producer at the local level, such as a server power supply, may in fact be a consumer at the global level but the approach is conveniently generalized in a hierarchical manner. Thus, each device in the overall system fits in a hierarchy. The policies for each power-managed device must be flexible enough to adapt to the unique demands for services from that device. The interface between devices is through its budget and aggregated at upper levels as applicable. The units of budget allocation align with measurement methodology (watts).

4.1 Power Limit Policy

Global power allocation must be able to identify two main events. First, if a node needs less power than currently allocated a means must exist to reclaim power it was previously allocated. Second, as a node becomes power deficient, an additional signal is needed to ensure more power is allocated given the global limit. Knowledge of the last local power limit assigned for a node is not sufficient to recognize these events.

To ensure adequate power allocation during the two previously mentioned events, the power limit management policy presented in this thesis expects a node to provide an idea of how much power it believes it needs to all other nodes. This is not its local limit, as it is controlled with respect to the global power available. The power need is determined by a node based on whatever policy it deems appropriate to include overall demand, delay characteristics in an application, or other measures. Utilizing power usage directly to quantify need would not be as effective as usage tends to fluctuate to such a degree that decisions become unnecessarily more complex in global allocation. It is better to ensure global allocation allows for coarse-grain adjustments if needed.

This notion of local node power need is easily quantifiable based on information useful to manage power limits. As discussed earlier, in the context of measurement fluctuation, utilizing average power in small time intervals is an adequate solution to managing instantaneous power, P. To ensure such fluctuations can track a desired average usage, a power target, T, is used as a means of codifying a local node goal for interval t. To ensure an adequate bound on T, quantifying error on a per node basis is needed and is subsequently termed as burst, B.

Intuitively, B is the additional quantity of power above T a node can utilize. However, it is not regarded as the long-term average power consumption target. The burst value might be changed by a node to reflect a trend in usage. For instance, on an idle system, B might be very small (little deviation exists) yet on a very busy system with highly variable power usage, this value might be large. Since both target and burst are useful in managing actual power usage to a local limit, need is easily expressed as $T + B$ without any additional knowledge. If all nodes provide these values, the global allocation policy can determine a desired minimum power need for the entire circuit.

Figure 4.1 represents the power usage of a single node over two time intervals. The instantaneous usage fluctuates for both intervals, but for $t + 1$ a local limit policy decreases the power target. As a result, this is the signal in global reassignment to reclaim all or a portion of the excess power this node had in interval t. To ensure a local node power limit increases, the policy in reassignment is to always allocate available power to ensure sufficient global reserve, but also allocate a portion above the need for each node. Thus, power need can increase up to the point it reaches the assigned local limit. In this case, in the next time interval this condition is detected in global allocation and a subsequent increase in the local power limit is done if sufficient reserve exists. Maintaining a clear separation between $T + B$ and L provides additional flexibility in a power management mechanism. For instance, it might be desirable to broadcast an immediate new burst value to a subset of hosts based

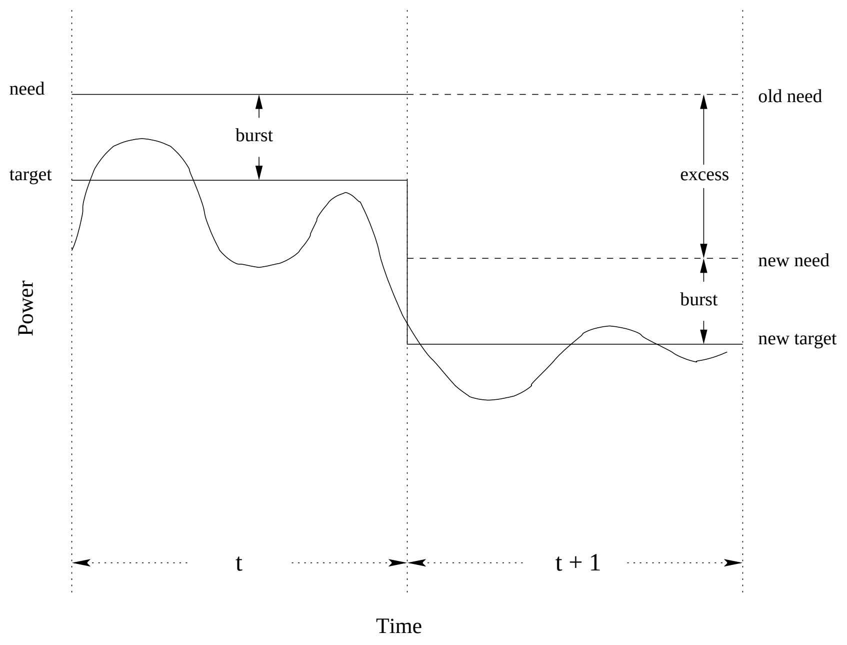

Figure 4.1: Policy for managing local limits by dividing time into discrete inverals and inferring local node need from its target and burst.

on their current values. By maintaining separation, such a condition is globally detected and enforced.

This strategy in limit assignment accomodates a variety of different policies. For instance, if energy conservation or controlling heat is a priority, the derivation of the power target could reflect an assignment indicative of such a requirement. However, for simplicity, the remaining discussion is restricted to managing instantaneous power and increasing throughput. 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 dealt with in a similar manner.

4.2 Component Introduction

There are two primary autonomic managers within PICLE. First, the Local Power Agent (LPA) uses feedback about power consumption to ensure the power of an individual node remains below a defined local limit. The LPA regulates system power consumption of a cluster node by changing the frequency and voltage setting of the microprocessor. In addition, for those systems without this capability and to complement the strategy of using DVS, a fine-grain task throttling component exists to ensure consumption can be restricted. This Task Governor (TG) utilizes runtime behavior to throttle tasks according to priority and power consumption. Through the usage of TG and DVS, adjustments at runtime can instantaneously reduce power consumption, albeit with some reduction in performance.

The second autonomic manager in PICLE is referred to as the Global Power Agent (GPA). Its duty is to assign the local node power limit, based on the known global limit, in order to achieve efficient utilization across all nodes. It collects data, projects future performance, then in a non-uniform fashion allocates the available power in order to balance system throughput. This allocation mechanism accounts for the service constraints that exist across all nodes. In addition, a sufficient and administratively controlled reserve exists for activities such as adding additional cluster nodes.

4.3 Managing Power to a Local 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 (*i.e.*, power reserve). Using its assigned limit, a node is responsible for the suballocation of power at a fine-grain level. A node attempts to manage its actual power usage given L, an output from the GPA. To ensure any 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 can use due to technical limitations (*i.e.*, maximum node consumption possible given connected devices) or to limit over consumption for business reasons. Thus, the local node power relationship is governed by:

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

At the node architectural level, each device in a node has an interface to relay or provide information related to power draw, available gears (*i.e.*, DVS performance states), as well as minimum and maximum power required directly to the power management mechanism. Although elements of this are available in laptop systems using the ACPI specification [37], future development of consistent interfaces to hardware should promote similar hardware sensors and functionality for servers using the same methods.

4.3.1 Feedback Control

A system level power feedback controller ensures a node manages consumption with respect to its power target, T. The local power limit is regarded as an upper bound on instantaneous power usage. For energy constrained sources such as batteries in laptops, feedback is a useful technique to set a target and subsequently manage consumption [49]. For other devices that seek to manage average consumption within small intervals, it is also useful by utilizing a circular sampling window of size w. As measurements are taken of usage, P, the error from the target is $\epsilon = T - P$.

The value for ϵ does not properly capture the notion of time. A deficit of 10 watts for a 3 second interval has a larger impact than the same deficit in just 1 second. Thus, the controller calculates energy, in joules j, based on the power usage measurement interval, $j = t \cdot \epsilon$. The previous error is then used in a *Proportional-Integral-Derivative* (PID) controller [62]. In a PID algorithm, the proportional control provides the uplift, the derivative helps eliminate overshoot from the target, and the integral corrects steady-state error (*i.e.* ϵ does not change from t to $t+1$). However, as a result of eliminating steady-state error, the justification for w is observed as the need arises to bound the integral from excessive aggregation (*i.e.*, T is consistently larger than P). Thus, with instantaneous error known and gain constants G_1, G_2, and G_3, the energy surplus or deficit per second is

$$e = G_1 \cdot j_t + G_2 \cdot \int_{t-w}^{t} j\ dt + G_3 \cdot (j_t - j_{t-1}).$$

It is now useful to use e as an indicator to control the overall performance of the system. Based on a predefined threshold, if it is exceeded and $e < 0$ (implies excessive usage with respect to T) the system performance can be decreased. The inverse condition, $e > 0$, implies system performance can be increased as power usage differs from the target significantly enough to warrant additional power consumption. Smaller thresholds are preferred when comparing e and the deviation from T. This facilitates a tight bound on T, but there are inherent limitations in hardware that prevent excessive state changing from having a measurable effect. In addition, although L is the upper bound there are delays to which new measurements provide effective information in future decisions. This is discussed in greater detail in measurement fluctuation and may manifest itself in increased and undesired utilization above L before local performance (and power) can be decreased.

4.3.2 Resource Consumption Throttling

It is important to recall an assigned power limit accounts for a margin of error above the target. The impact to an LPA is to ensure $T + B$ does not exceed L. To accomplish this objective, access to power consuming devices must be limited to ensure consumption is managed appropriately. As the local power limit constrains performance, access to a resource must also preserve task constraints. PICLE accomplishes this in the LPA for the CPU using the Task Governor (TG). The objective of the TG is to ensure that as power need exceeds L, local allocation is managed effectively by reducing the CPU time that some tasks desire. This objective for the TG is complementary to using DVS; however, the current PICLE implementation only uses the TG in systems without DVS or clock throttling.

Without a TG, it is less likely power usage can be managed as close to, but less than L. Program behavior can be very bursty with regards to computation. This activity is likely to cause a similar effect in power consumption. An overly aggressive policy of never changing to a higher performance gear, if L might be exceeded, does not favor overprovisioning methodology. The approach with the TG is optimistic, as appropriate reaction is taken after first determining $P > L$. There are certainly means to mitigate this risk for more conservative environments. One such policy in PICLE is to increase the per node B, as applicable, to provide a larger safety margin. A similar methodology for all cluster nodes supports increases to the global reserve, R.

To keep in line with the original goals of transparency and no modifications to applications, the TG makes use of two important concepts, task interval i and task skip s. This allows tasks to be temporarily descheduled from the CPU run queue after a period of execution, i, and later rescheduled after a specified quantity of time, s. This is done in the operating system dynamically as tasks execute, but the assignment of i and s is left to an external component from the LPA.

In the LPA, where power usage data is collected, it is possible to use the TG to throttle tasks to a ratio of their original CPU utilization. To illustrate this feature, in a simple scenario with one runnable task (*i.e.*, it utilizes a majority of the CPU discounting operating system overhead) in time period t, if $i = s = 1$ in $t + 1$ this task is throttled to 50% of its original utilization. The ratio of s to $i + s$ is subsequently referred to as the reducibility factor f. In this example of $f = 0.50$, this reduction in CPU time may or may not correspond to an exact 50% reduction in CPU power. It depends on the task instruction mix and the work performed in $t + 1$. If the solution were static, such an assignment may hold and never warrant future adjustment. However, tasks go through phases in their execution so this necessitates a dynamic solution to ensure a tight bound on L is enforced. The preceeding example is oversimplified, as it is rarely the case just one task executes. Thus, the estimate needed for task throttling is a bit more complex. In addition, having an infinite number of values for f further complicates an assignment for any given task (both in degrading and recoverying from different values of f needed to ensure L). These two problems are further addressed in chapter 5.

For this to work in practice, a mechanism must also exist to prioritize tasks for throttling. In addition, it is important to provide a task CPU utilization minimum to ensure its service level constraint is met. The first precondition in TG is met by utilizing the inherent, user or administratively defined task priorities rather than adding an additional abstraction to what already works well within the operating system. In addition to inter-priority classification, intra-priority distinction is made to first throttle those tasks shown to be higher power consumers. This identification is handled by calculating instructions per cycle (IPC) for all tasks and throttling those with a higher IPC first. If the work a task performs does not change, even after task throttling it is not likely to reduce its IPC. This is a result of an equal reduction in both the instructions that execute and the cycles allocated to a task if the work is constant. Thus, a task with the higher power consumption (with no change in demand) will continue to receive less CPU resource time until it reaches its minimum. In addition, if a higher priority task is the cause for higher power usage, low priority tasks suffer first but do not starve due to a practical lower bound used in the absence of a task specific utilization minimum. A final point to consider is task time slice times can differ signficantly, so just using IPC and any static encoding for reduction is not satisfactory to determining the effect on power usage. Accounting for run lengths for each task and the impact of f in $t + 1$ is necessary to ensure total reduction in CPU time is accomplished.

For the TG to be effective, the degree of aggregate throttling must be indicative of the error from L. This error, expressed in units of time t, is calculated as $\omega = (\frac{P-L}{P}) \cdot t$. The goal of the TG is then to reduce task resource time to ensure the per task time reduction, k, does not cause the utilization minimum for that task to be violated and the $\sum_{i=1}^{m} k_i = \omega$ where $m \leq n$. PICLE currently has no mechanism to ensure this local condition holds;

however, the global allocation scheme does provide a mechanism to ensure L_{min} holds. Thus, a sample policy is to dynamically calculate L_{min} for $t+1$ based on determining all per task resource utilization minimums (or the global starvation threshold minimum). The global allocation mechanism does not regard L_{min} as a static quantity for each iteration of local power limit assignment.

4.4 Global Limit Allocation for Clusters

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. PICLE dynamically and non-uniformly allocates power among nodes in order to increase the aggregate performance of all nodes given irregular workloads. To make this work in practice, time must be divided into discrete processing intervals and synchronization is needed. 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.

The global power limit itself is known and quantifiable based on circuit capacity. Considering a given quantity of power to keep in reserve as $R \geq 0$, the allocation problem is to calculate L for each node i such that

$$\sum_{i=1}^{n} L_i + R \leq G.$$

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 cluster demand, $D = \sum_{i=1}^{n} W_i(L_i)$. This problem is not solvable because work performed for a given power limit changes dynamically and is not known a priori. Therefore, PICLE first allocates 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 contribution of each node with respect to satisfying D_{t+1}. This estimate ensures nodes with greater need have a higher priority. Finally, a reassignment of all node local power limits is done for $t+1$.

On successive iterations of the global power mechanism, the node surplus or deficit is given by $S+L-(T+B)$. The value S reflects the aggressiveness of global redistribution and

L is a calculated output of the global allocation algorithm. A higher S value manifests itself in assigning more of the available power after first assigning all node L values. Intuitively, if $L-(T+B)=0$, the node is effectively utilizing its allocation; likewise, if $L-(T+B)<0$ the node is using less power than desired so excess node power is available for reallocation. The quantity S provides the ability for a node to increase consumption in subsequent iterations if T or B increases.

Based on the estimate of aggregate global power need, if it is less than but close to G, the predicted aggregate power need is used 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 D_{t+1}, $\frac{\Delta W_i(L_i)}{\Delta G}$. In addition to insufficient allocation, if less power is needed than available, nodes receive an additional allocation of power as a result of $S>0$.

4.4.1 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. This is accomplished using DVS or clock throttling. Second, a means of controlling power locally at each node must exist. This condition is met in PICLE using an LPA. 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 with an impact on power allocation performed. Finally, any available excess power must be allocated to ensure the global limit is not exceeded, yet nodes receive an appropriate value of additional power indicative of demand. Previous chapters have provided detail to the first two points, the next two sections discuss the latter two requirements in greater detail.

4.4.2 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 demand historical data. Using this same information, a predicted value of demand, $W_{predicted}$, can also be calculated and utilized to forecast demand for the next interval

$$d=\frac{W_{predicted}}{W_{average}}$$

Given current node power need, an approximation is now possible for the next interval using:

$$A = (T + B) \cdot d + S$$

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 lower 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 should experience increased local demand if its need increases faster with respect to other nodes in future intervals. This, in turn, may subsequently cause additional increases in its local limit as the power allocation method begins to balance aggregate power based on demand throughout the whole cluster.

4.4.3 Allocating Available Power

As previously discussed, to provide an upward momentum for nodes to continue to increase their power limit a quantity of global power may exist due to meeting all estimated obligations for all participants. In the previous discussion, S is simplified to represent the additional per node allocation. However, this additional power is not a fixed quantity allocated uniformly across all nodes. Instead, S is regarded as an administratively defined per node upper bound whose actual calculated value is dependent on $W_{predicted}$ and the aggregate workload, $W_{total} = \sum_{i=1}^{n} W_{predicted_i}$. If available power, after allocating all new local limits, is denoted as p, then

$$p = G - R - \sum_{i=1}^{n} (T_i + B_i) \cdot d_i.$$

With p known, a node specific allocation for S can be found based on its work contribution, $c = \frac{W_{predicted}}{W_{total}}$. It then follows that the per node additional allocation is $S = p \cdot c$ and this value is used in the approximation A (S is bounded by the administrative limit). 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 LPA for the next interval.

Chapter 5

PICLE Implementation

The PICLE implementation presents a framework that allocates power globally and manages local node power consumption based on an assigned limit. In this chapter, the measurement prerequisites are discussed that enable PICLE to function and the two core components are covered in greater depth.

5.1 Measurement Requisites

A hardware solution to measure power must exist for any power control mechanism to function. This is accomplished in PICLE using both digital multimeters and watt meters. Custom software in PICLE allows core components and other tools (the command-line tool *mclient*) to obtain current power readings from a user-level daemon process (*mlogger*). In *mlogger*, client reporting and measurement device reading are in separate threads. In addition, *mlogger* starts a new thread to handle each client and once connected, a client issues as many requests (without reconnecting) as it needs to obtain power usage information. The *mlogger* utilizes a vendor independent core API and it supports querying four different meters. There are no timing delays imposed by *mlogger* so data is read by it, and able to queried by clients, at the rate the connected meter can provide.

5.2 Core Components

Two autonomic managers comprise the core framework. Starting at the lowest level, interface drivers provide the intelligence to determine and manipulate the state of devices in a cluster node. Each node aggregates multiple drivers into a cohesive entity referred to as the Local Power Agent or LPA (*lpagent*). It is the inter-device mediator responsible for determining the power consumption target, T, given a locally assigned power limit, L. The LPA selects device gears to meet the power consumption target or uses the Task Governor (TG) to perform task throttling. A message queue is used by the LPA as a logical bridge to device drivers as well as for other external requestors, as further discussed below. This message queue facilitates loosely coupled communication between each device and the general LPA controller through standardized functions (*i.e.*, shift_up(), shift_down(), etc.).

The second major software component in PICLE is the Global Power Agent or also referred to as the GPA (*gpagent*). The GPA is responsible for the coordination and interchange of relavent messages between nodes. It analyzes the state of the cluster and makes appropriate requests to the LPA using the message queue and shared memory. Communication between multiple GPAs is done using a group string identifier specified on startup. This identifier is subsequently referred to as a Power Management Group (PMG). The GPA learns the state of all other nodes by broadcasting to and receiving data from all nodes in its PMG. There is not a one-to-one correspondence between a PMG and subnet; however, PICLE limits PMG nodes to the same broadcast network. In addition to receiving state information of all other nodes, the GPA responds to other general requests. These messages include administrative control requests (*i.e.*, assigning a new global limit) as well as other message types. Both the GPA and LPA are implemented as separate, non-priviledged daemon processes.

The interaction of both the LPA and GPA is depicted in Figure 5.1. The GPA calculates and assigns the local power limit based on external information provided by all nodes in the PMG (along with knowledge of G). The LPA is responsible for ensuring the target power of an individual node is managed, subject to its GPA-assigned local limit. Each of these components must run on each node in the cluster.

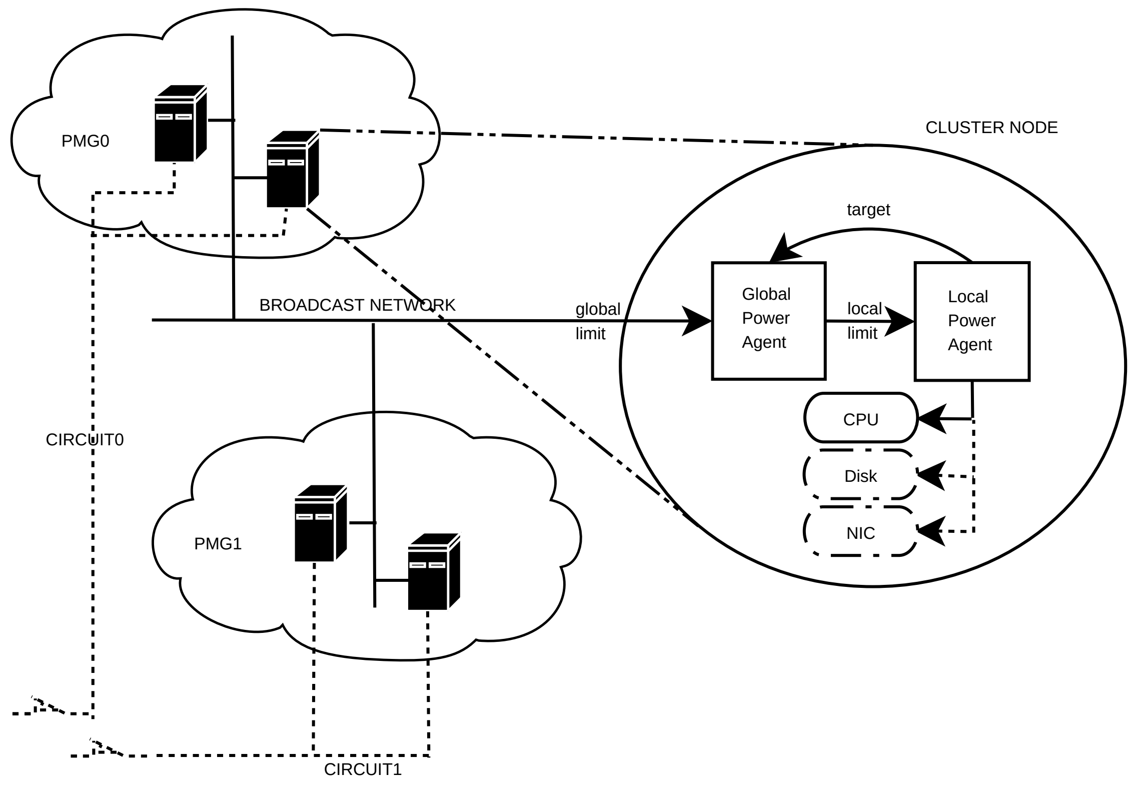

Figure 5.1: Relationship of power management groups, agents, and power-managed devices.

5.3 Local Power Agent

The LPA may not have, nor be cognizant of, power-scalable devices. It simply controls power consumption through the device interface layer given the exposed device's functionality. It is also the entity responsible for obtaining power usage data available from external sources (*i.e.*, directly from a meter or through ACPI). The long-term goal of the LPA is to manage all devices, but the remaining discussion is confined to just the CPU given the current progress in PICLE. 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, this performance state is denoted 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. Logical LPA components are shown in Figure 5.2. Each of these is discussed in greater detail in the following sections.

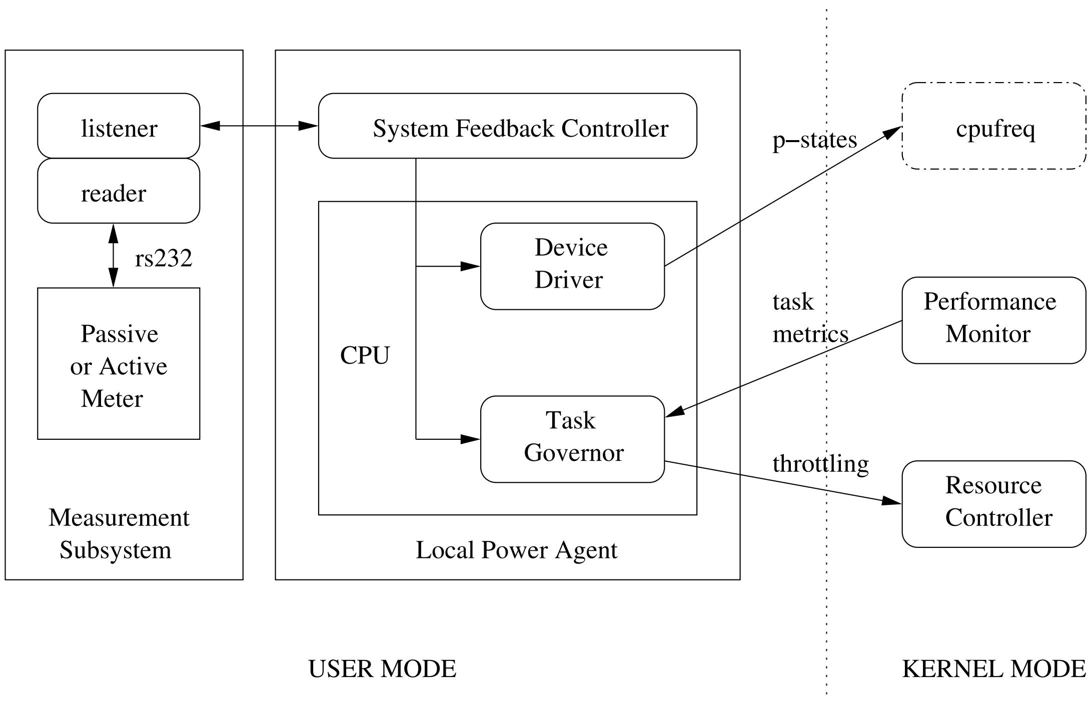

Figure 5.2: Logical entities and communication with the Local Power Agent.

5.3.1 Target Power Determination

The LPA derivation of T is flexible enough for different policies to be implemented without disruption to the power limit mechanism, as discussed in Chapter 4. In PICLE, the target power policy advocates overprovisioning so $T = L - B$. This is not always the best option. For instance, a robust mechanism could reduce T, with potentially no loss in overall performance, if task progress is limited by memory or I/O bound tasks. In such a case a reduction in CPU performance is possible, with no likely performance impact, if measured power usage, P, is consistently less than L. In addition, overprovisioning has the potentially negative impact of always maximizing power, subject to other constraints (*i.e.*, power reserve R, all node L_{max} values, etc.). However, as the number of nodes increases the global power allocation policy will make intelligent power adjustments as discussed in Chapter 4.

The treatment of the local power limit is regarded as a *soft* upper bound on instantaneous power usage. For instance, as the global power limit changes due to underlying power disruption, it might take several seconds for the new local limits to become stable given the magnitude of change. Given the design of the feedback controller discussed in Chapter 4, the sampling window size is 30 seconds. To prevent excessive gear switching and allow stabilization, a minimum of 5 seconds between changes is enforced in the feedback controller. This delay also helps mitigate the differing capabilities of CPUs (their subsequent ability to transition to different gears). In addition, the feedback threshold is set to 20 joules and is the value compared to e, the energy surplus or deficit that triggers gear changes. These values are tuned according to empirical observation.

5.3.2 Feedback and CPU Task Throttling

As stated previously, the Task Governor (TG) could work in connection with, or in isolation of, an DVS or clock throttling mechanism. The current implementation is restricted to using the TG in the absence of any other CPU control capability. Control of both DVS and task throttling is accomplished via startup options to *lpagent*.

5.4 Task Governor

For the TG to operate, task-level kernel data is required and an additional component is needed to assign and regulate task CPU time. To support these two requirements, two separate kernel modules are implemented whose interfaces are through /proc file system entries. The first module facilitates the collection of performance counter data in a Performance Monitoring Counters Module. The second module enacts task restrictions and is referred to as the Resource Controller Module. Each of these modules exists and functions in isolation of the other. The TG aggregates their respective functionality into a cohesive entity for the LPA. The TG is discussed in greater detail after first providing the details for the two aforementioned kernel modules.

5.4.1 Performance Monitoring Counters Module

To support the collection of task data, modifications to the linux scheduler are needed in *schedule()*. These changes include calls into the Performance Monitoring Counters Module (*pmc*) to support pre-task and post-task context switching functions. The *pmc* module manages an internal memory buffer to store data. This data is retrieved by the TG based on the power usage sampling interval. The only processor event configured by the TG is to monitor instructions (only AMD events [2] are currently supported). However, the *pmc* buffer facilitates the collection of all relevant task information. This data includes the processor ID, command name, process ID, user ID, task priority, time stamp counter, and the task runtime. Task runtime is obtained using the Power Management (PM) Timer, which provides good precision during frequency and voltage changes. As this kernel transient buffer fills, if a threshold below its maximum size is reached the *pmc* module will send a signal to the LPA to force a buffer read. In *pmc*, performance counters are reset prior to starting a new task and read right after that task completes. Thus, kernel overhead is not counted in task metrics; however, kernel threads assigned a process ID are dealt with just like any other task at this layer.

5.4.2 Resource Controller Module

The Resource Controller Module (*rcontrol*) requires changes to the kernel task structure to incorporate two additional fields to throttle tasks. In addition, scheduler changes are done in *sched_fork()* to initialize these values and in *sched_exit()* to perform cleanup. An additional kernel modification to support *rcontrol* is needed within the timer core code in the function *update_process_times()*. This function is called from the timer interrupt handler to normally charge one tick to the current process. Due to this modification, *rcontrol* facilitates fine-grain throttling given its per tick check to determine if a task should continue to execute.

The interface of *rcontrol* to TG is in millisecond precision. To limit processor time, a task skip, s, and interval, i, are used to reduce the cpu time a task is eligible to execute. Given available precision in *rcontrol*, $s = 1$ and $i = 1$ correspond to a 1 millisecond skip and interval. These values are converted to jiffy [1] measurements to provide appropriate reduction and accounting in the kernel scheduling process. As task restrictions are fed to *rcontrol* from TG, a hash table is updated based on process ID. This table only contains restrictions for throttled tasks so lookup overhead is minimal. In addition, the lookup is only needed if a task has no currently assigned i value.

To throttle tasks while executing, a wait queue is utilized when a given task

[1]Basic measure of kernel time in the scheduler. It is related to HZ, the basic resolution of the operating system.

i expires (*i.e.*, it is decremented per tick). Once this occurs, the task status is set to TASK_STOPPED and the scheduler is notified this task should be descheduled (it cannot be rescheduled directly due to handling the timer interrupt). In order to wake a stopped task, its task interval is set to the future time it is eligible to be runnable (*i.e.*, current time plus skip). A background kernel thread executes periodically whose execution delay is the minimum of the smallest task skip not woken in the last processing interval or a default time-out. As tasks are determined runnable by this background thread (*i.e.*, by noting their time stamp in i), their status is reset to TASK_INTERRUPTIBLE. After all tasks in the wait queue are examined, the standard kernel wait queue function *wake_up_interruptible_all()* is used to wakeup those tasks designated as interruptible. In PICLE, there are no special provisions for real-time or other tasks that might require special handling.

5.4.3 Details of Operation

The *pmc* buffer is normally read based on the power usage polling interval in the LPA. Given that a task is sometimes scheduled in excess of a hundred times per second (number of context switches), measured data is accumulated efficiently using hashing by the TG (the number of tasks is orders of magnitude smaller). With this aggregate data, appropriate task throttling is now performed by creating a sorted array ordered first on priority, then instructions per cycle (IPC). Next, the CPU time reduction, ω, is calculated and if $\omega > 0$, deficit handling is done. Otherwise, a time surplus exists and existing task restrictions are removed or adjusted as needed. Since CPU time recovery is the inverse of any task time restrictions put in place (*i.e.*, proceeding in the opposite order with a similar algorithm), only the reduction algorithm details are further discussed below.

Using ω and the sorted list of tasks, a step down algorithm is implemented until $\omega \leq 0$. If a new task is about to be throttled (*i.e.*, it has no current f), the value for f is calculated based on the ratio of ω and its runtime, r, with respect to the power polling interval (*i.e.*, $f = MAX(1, \frac{\omega}{r})$). With f, a precomputed lookup table is accessed to determine s and i. From this, a rate of throttling change, h, ($h = f$ in the first iteration) is found and used to project the amount of task drop (milliseconds) it is predicted to experience, $d = r \cdot h$. Thus, $\omega = \omega - d$ and this process continues until ω is zero (with this task or other tasks). If a sorted task is already throttled, its existing throttling level is stepped down one reducibility factor at a time ($h = \Delta f$). Note, as this step down algorithm iterates with the same task, its runtime is adjusted $r = r - d$. In addition, the step algorithm facilitates a tight bound on any task utilization minimum constraint. For simplicity, TG maintains a global minimum to prevent task starvation.

In the algorithm used by the TG, certain tasks may be excluded. For the purposes of PICLE and obtaining results, those tasks owned by root along with PICLE tasks are excluded. The relationship between i and s is important. Having an i too long might not break a task into small enough increments to have an associated power reduction (if it is, or part of, the cause for high power usage). Thus, this assignment is not as easy as simply incrementing s or decreasing i independently. The dependence of these values and the associated f is further discussed in Appendix A.

5.5 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.

The logical components of the GPA are shown in Figure 5.3. 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

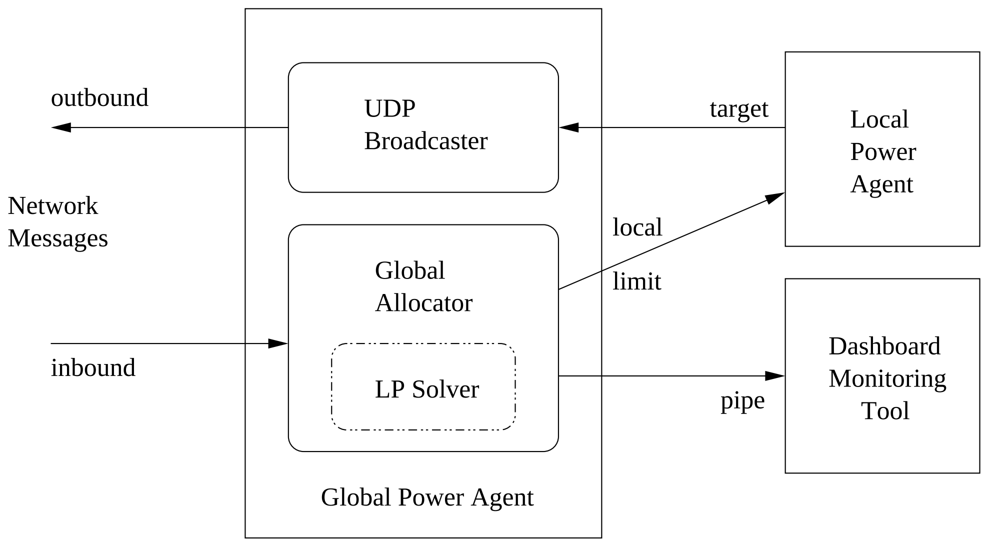

Figure 5.3: Logical entities and communication with the Global Power Agent.

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.

5.5.1 Power Management Groups

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 PICLE, 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 be balanced to reflect an appropriate site policy.

5.5.2 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, as noted below.

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.

5.5.3 Linear Programming Solution to Global Allocation

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 node power limit, bound its value by the maximium approximate node power value, A. This value accounts for all cluster demand and provides additional power above $T + B$.
5. Finally, ensure a sufficient reserve, R, is kept and the global limit, G, is not exceeded.

The informal steps outlined above are translated into the LP model shown in Figure 5.4. 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. In PICLE, such an instance is not inherently addressed in the implementation.

The outputs from the LP model include the current total power consumption, $O \leq G$, as the solution to the objective function. In addition, the amount of power after allocations are made and the minimum power needed to meet all node constraints is determined, $\sum_{i=1}^{n} L_{min_i}$. The critical output though is new local power limit for the LPA in the next time interval. This is found by saving the offset for the node within the objective function as the model is built at runtime. It is important to notice that A accounts for maximize:

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

subject to:

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

Figure 5.4: LP model to maximize G and assign L_i $\forall i \in [1, n]$.

the relative power need for each node (either surplus or deficit). Thus, each nodes LPA 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. For ease of reference, all notation in Figure 5.4 coincides with variables discussed in Chapter 4.4.

Given the model in Figure 5.4, a linear programming solver [14] calculates the outputs using the simplex method. All values are considered to be real, continuous values within their respective bounds. The solver is embedded directly in the GPA (runs in-process) and the 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 of 1 second is used for generating suboptimal solutions.

5.6 Support Tools

Although the long-term intent of PICLE is to be totally autonomic in operation, it currently 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.

5.6.1 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). The RPC interfaces are platform independent and handle argument marshalling and unmarshalling (*i.e.*, host-to-network and network-to-host conversions). 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).

5.6.2 Cluster Monitor

Similar to the handling of messages by the GPA, the distributed allocation algorithm is also utlized in a console, curses-based 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. If it executes on the same node as a GPA, it automatically connects locally to that GPA to receive broadcast data via a named pipe. Thus, the GPA monitors the state of connected *dashboard* clients in addition to waiting for network requests in the same thread. As broadcast data is received by the GPA, it is routed to every connected *dashboard* client. If *dashboard* is run on a non-cluster server (*i.e.*, a non-GPA managed node), it simply listens on the broadcast port for data analogous to a GPA. This tool displays the current state of the entire cluster and can ensure all nodes properly adhere to the power allocation strategy.

Chapter 6

Results

6.1 Architecture

To evalute PICLE, a cluster of ten servers was built using frequency scaling processors. They were assembled using the following hardware: 40 GB Maxtor EIDE 7200 RPM disk drives, ASUS K8V motherboards (on-board 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, along with the original settings, are illustrated in Table 6.1. The CPU power usage in this table is from [3]. Measured system power at each of these gears is discussed in greater detail in Appendix C.

6.2 Microbenchmarks

To quantify the efficiency and performance of PICLE, a series of low-level benchmarks was performed. The first reflects the implementation cost of switching to different gears. This cost was measured by first constructing a kernel module to utilize the *cpufreq* notification mechanism to determine the internal kernel cost for state changes. This kernel cost is discussed in greater detail in Appendix B. To measure the implementation cost,

Gear	Frequency (MHz)	Voltage	CPU Power
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 6.1: Assignment of gear to frequency and voltage values for the CPU used in DVS. All gears added to the default BIOS table are identified in the power column.

agentctl was used to control gear changes and the kernel time is subtracted from the total time. The measured time values from the LPA ranged from 23 to 363 microseconds. A change to a gear either either one above or below from the current gear was the most efficient. This result supports the incremental model (discussed in Chapter 5) used to maintain the power target in the LPA with DVS. Nevertheless, 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*. The *cpufreq* module handles the low-level architectural details to control the frequency and voltage in the CPU.

There is additional overhead to connect and disconnect from shared memory, as measured using *agentctl*. This ranged from 164 to 242 microseconds and the GPA only incurs this cost once when first started. For the GPA, 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 low-level payload details can be verified for integrity and unpacked from the buffer area. These packets incorporate all the relevant metrics discussed in Chapter 5. In addition, current power usage and local power limit are also sent and provide a means for the *dashboard* monitoring tool to maintain compliance and track trends.

The overall CPU time requirements for the system are minimal. Over the course of a 10 minute period on a single node (samples taken every second), the CPU utilization and memory used for the LPA and GPA was logged using *top*. Using this tool, measured CPU utilization for both agents was negligible. In addition, the average virtual memory required for the LPA was 170 kb and the GPA required 1.84 MB. The latter value is largely due to the size of the linear programming solver library (1.39 MB).

The kernel memory costs for the Resource Controller Module (*rcontrol*) and Performance Monitoring Counters Module (*pmc*) were 396kb and 450kb, respectively. The *pmc* module includes an internal 128kb buffer for logging task data. Using the LPA, the time to transfer and process this buffer in the Task Governor was measured every second for a 10 minute period on an idle system. The average time calculated for this period was 9.8 milliseconds. As this result and all previous microbenchmarks show, the processing overhead to run PICLE is minimal. The next topic of exploration is an analysis of overprovisioning.

6.3 Overprovisioning Benefit

To examine the nature of the trade-off between throughput and power, a series of synthetic CGI [18] programs were constructed and run using an Apache 2.0 web server. The *httperf* [51] tool was used to generate a sustained workload using four cluster nodes configured as request clients. A single web server was setup to handle requests and clients were configured to overload the server based on a request timeout of five seconds. When the aggregate number of errors for all clients exceeded five percent, the throughput was considered maximized for that performance gear. For the time alloted for a benchmark (30 seconds), the average system power consumption was calculated using multimeters. Extrapolations based on single node measurements are used to show the benefit to overprovisioning.

An additional cluster node controlled the request clients and collected power readings. All results notation depicts the highest performance gear (and highest power consumption gear) as zero. A consistent theme emerges from the synthesized benchmarks. The highest power consumption gear does not have the highest performance (*i.e.*, throughput) per unit power. To illustrate this difference in throughput with respect to power usage, consider Figure 6.1. It shows the resultant increase in power from gear 1 to 0 is approximately 12.57%; however, the throughput gain is only 6.25%. In contrast, the increase in throughput from gear 6 to 5 is 25% with only a 6.19% increase in average power usage. Table 6.2 depicts the raw data collected in this benchmark.

Using the data in the previous result, it is possible to show a reduction in the CPU performance gear and an increase in the number of nodes can decrease aggregate power consumption. This is shown by a simple example and the data in Table 6.2. If the service requirement states the desired throughput is 36000 connections in 30 seconds with

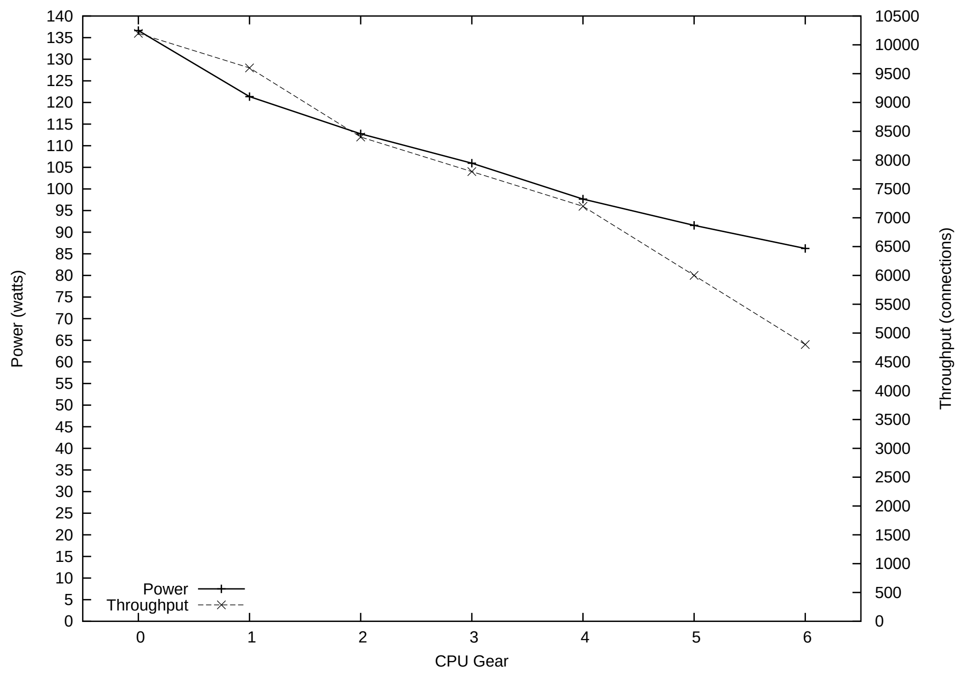

Figure 6.1: Power and performance trade-off in a CGI synthetic benchmark program (using merge sort) run by Apache.

a client request timeout of 5 seconds, 5 cluster nodes meet this requirement running in gear 4. Assuming this cluster has work evenly distributed to each node, the total power usage is approximately $5 \cdot 97.66 = 488$ watts. Four similarly configured servers (without managing the power proactively) could also service this same load; however, the total power usage is $4 \cdot 136.63 = 547$ watts. This example illustrates a decrease of about 11% in total power needed.

The preceding example uses an increase in nodes to show the benefit in conforming to a lower power limit. In addition, this same data is used in Table 6.3 to reflect additional gains in throughput. Extrapolation begins with four nodes and the gear represents a static assignment done on all nodes. The indicated aggregate throughput and power values are calculated using the respective values from Table 6.2 multiplied by the number of nodes. Notice the greatest gain in work performed is at gear 3 and is nearly 13%. This gain is despite the 70.25 watts of unused power due to under utilizing the global limit of 600 watts. This unused power represents a loss in throughput without a more proactive means of power allocation (*i.e.*, PICLE).

The prior example helps motivate overprovisioning, but it does not reflect the

Gear	Throughput	Change	Power	Change
0	10200	6.25%	136.63	12.57%
1	9600	14.29%	121.37	7.65%
2	8400	7.69%	112.74	6.41%
3	7800	8.33%	105.95	8.49%
4	7200	20.00%	97.66	6.66%
5	6000	25.00%	91.56	6.19%
6	4800		86.22	

Table 6.2: Power and throughput details for the merge sort CGI. Power is measured in watts and throughput is the total number of connections.

Nodes	Gear	Total Throughput	Change	Total Power	Error
4	0	40800		546.52	53.48
4	1	38400	-0.06%	485.48	114.52
5	2	42000	0.03%	563.70	36.30
6	3	46800	12.82%	529.75	70.25
6	4	43200	5.56%	585.96	14.04
6	5	36000	-11.76%	549.36	50.64
6	6	28800	-41.67%	517.32	82.68

Table 6.3: Throughput gains with a 600 watt power limit and static gear assignment. Power and throughput are extrapolated using the individual results from the proceeding table.

strict representation of all applications. Of the 10 benchmarks created, 9 exhibit the general trend depicted in Figure 6.1. An exception to the high difference of power and throughput at gear 0 was exhibited in one benchmark. The resulting power and throughput relationship in Table 6.4 shows an even larger gain in throughput at lower power gears (*i.e.*, 6 to 5) than the preceeding example. However, the benefit of the reduced power gear 1 (from 0) is not as pronounced [1]. In this example, the benefit in throughput and power is nearly equal until the lower gears starting at gear 4.

Another important result emerged from one benchmark and is shown in Figure 6.2. Note the minimal difference in throughput from gear 3 to 2. Unfortunately, there is a 6.21% increase in power usage for this same transition (the raw data is not shown for brevity). The data does reveal the lower bound overload threshold was just exceeded for gear 3, but the upper bound prevented gear 2 from additional gains. With knowledge of

[1]Sort-based benchmarks used the same quantity and distribution of items in all algorithms.

Gear	Throughput	Change	Power	Change
0	10800	12.5%	140.87	11.40%
1	9600	6.67%	126.45	6.87%
2	9000	7.14%	118.32	6.06%
3	8400	7.69%	111.56	8.23%
4	7800	18.18%	103.08	7.84%
5	6600	37.50%	95.59	6.91%
6	4800		89.41	

Table 6.4: Power and throughput details for the insertion sort CGI. Power is measured in watts and throughput is the total number of connections.

this result a power management policy should select the lower power gear.

6.4 Local Power Agent

The central idea of the LPA is to manage average power consumption for small time intervals. This is done to show an aggregate throughput gain is possible by increasing the number of nodes and ensuring average power is well utilized beneath the local power limit. To verify the efficacy of the LPA, the merge sort CGI was used as a representative benchmark to calculate two data sets. In the first, a 1500 watt limit is established and extrapolation starts with 10 nodes. Starting with 10 nodes allows an interpolated point to occur between each static gear assignment. These interpolated points are calculated by finding a power target for the local controller such that $T = \frac{G}{n}$. With the 1500 watt limit, each increase in n ensures a new controller power target exists between two consecutive gears. In the second data set, a 550 watt limit is used and extrapolation starts at 3 nodes. Table 6.5 depicts these two calculated data sets. Total throughput for this table is found by subtracting the number of errors from the total number of connections. This was not done in the previous overprovisioning examples for simplicity (PICLE was not used).

With the two data sets just constructed, throughput and power measurements were taken while the LPA managed power according to the calculated power target values from Table 6.5. A sustained load was applied for 30 seconds and throughput was considered maximized based on a 5 second timeout using 4 cluster nodes as request clients. The measured results of the LPA managing usage to the calculated power target are shown in

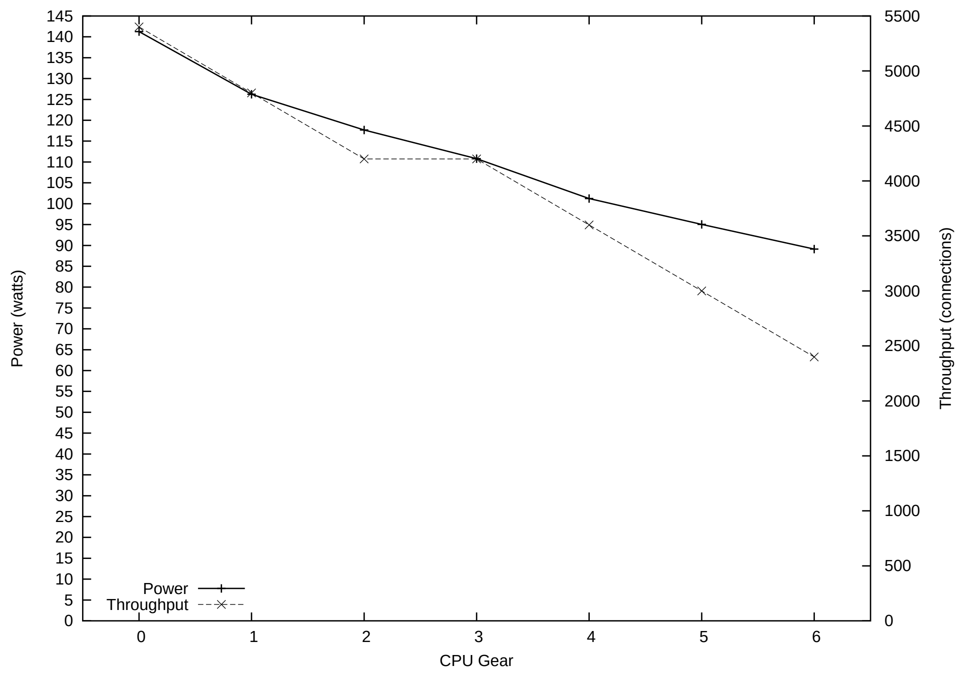

Figure 6.2: Power and performance trade-off in a CGI synthetic benchmark program (using a prime number generator) run by Apache.

Table 6.6. The last column labeled error is the node measured average power subtracted from the ideal power target in Table 6.5. The total power and throughput are calculated based on the single node value multiplied by the number of nodes.

Figures 6.3 and 6.4 illustrate the benefits of overprovisioning using the LPA. These figures are created based on the data from Tables 6.5 and 6.6. There are two lines depicting throughput, one for a single node and the other for the extrapolated aggregate value. As expected, the individual throughput decreases as available power is reduced. The aggregate throughput curve shows the performance of the cluster. As the power decrease crosses a vertical bar, another node is supported within the power limit (this is the ideal power target in Table 6.5). It is important to notice that in both Figure 6.3 and 6.4, the aggregate throughput increases at some points even though the individual node performance decreases. In Figure 6.3, throughput is highest with 12 nodes with an average power consumption of about 136 watts. With Figure 6.4, throughput is highest with 5 nodes and an average consumption of about 110 watts.

The raw data in Tables 6.5 and 6.6 show there is a 7.15% gain in throughput between gears 0 and 1 for 11 nodes. In addition, there is a 6.69% gain in throughput

Gear	Nodes	Total Throughput	Total Power	$T = \frac{G}{n}$
0	10	209150	1418.10	150.00
1	11	212256	1396.78	136.36
2	12	215184	1464.32	125.00
3	13	208169	1446.12	115.38
4	14	201306	1440.32	107.14
5	15	173415	1440.30	100.00
6	16	143072	1443.68	93.75
0	3	62745	425.43	183.33
1	4	77184	507.92	137.50
2	4	71728	476.96	137.50
3	4	64052	444.96	137.50
4	5	71895	514.40	110.00
5	5	57805	480.10	110.00
6	6	53652	541.38	91.67

Table 6.5: Fixed gear throughput, aggregate power, and target node power usage, $\frac{G}{n}$. The first data set is with a global power limit of 1500 watts; the second set is with a limit of 550 watts.

between a staticly assigned gear 2 and the LPA using 12 nodes (after gear 2 there is no resultant benefit to increasing the number of nodes). Additional consideration must be taken for environments that do no proactive management of power. In such a case, only 10 nodes can exist under the limit (all nodes run at maximum power) so the throughput benefit of the LPA using 12 nodes is 9.77%. Clearly, the benefits just mentioned are dependent on the global power limit and the number of nodes. For instance, using the measured data starting with 3 nodes the benefit from a statically assigned solution is approximately 5.91% (4 nodes at gear 1). In addition, the gain from an unmanaged 3 node solution is 30.28% (5 LPA nodes).

The previous benefits on overprovisioning must be put into perspective. The prior mention of an unmanaged environment still implies a minimal amount of intervention and monitoring. As mentioned in Chapter 1, many data centers utilize ad hoc methods to manage power. As a result, the true benefit to deploying a solution such as PICLE might be much larger for those environments using simple techniques to provision rack space (*i.e.* provision a rack such that $\frac{3}{4}$ of the global limit is allocated).

Nodes	Total Throughput	Total Power	Node Power	Error
11	224103	1498.09	136.19	1.91
12	229584	1498.68	124.89	1.32
13	216112	1464.32	112.64	35.68
14	210826	1477.84	105.56	22.16
15	182700	1500.00	100.00	0.00
16	141232	1478.88	92.43	21.12
4	81600	546.96	136.74	3.04
5	81745	547.70	109.54	2.30
6	63204	544.86	90.81	5.14

Table 6.6: Dynamic gear throughput and aggregate power using the LPA. The indicated error is the difference between the measured average power and the power target.

6.5 Task Governor

The previous LPA results depict the use of different CPU gears to manage power consumption. PICLE also utilizes the LPA's Task Governor to manage power consumption in the absence of gears. To evaluate this functionality, the *cpuburn* [61] program is used. For the first test, a single program instance was executed at gear 0 for 750 seconds. In each second, the power usage, local power limit, and task reducibility data was collected. The local power limit was changed during the test with *agentctl*. The results of this test are shown in Figure 6.5. The power limit and usage are shown using the Y axis on the left side. In addition, the reducibility of the program is shown on the Y axis on the right side. As reducibility is a measure between zero and one, this axis is scaled to show this measure as well as power on the same graph.

The results in Figure 6.5 show a fairly tight bound between measured power usage and the assigned local power limit. Within the Task Governor, a power usage reading 2% above the assigned limit or 3% below was considered unacceptable. In such a condition, the controller would either reduce or increase CPU time as needed. These percentage values were arbitrarily chosen and would normally reflect a site-specific policy. The three apparent power surges around 120, 340, and 650 seconds are not a result of incorrect task throttling. The data shows a periodic task run by root executed briefly during those time periods. As discussed in Chapter 5, root tasks are specifically excluded. This is partially due to the absence of a more robust exclusion mechanism in the current implementation.

In Figure 6.5, even with the most drastic power change (*cpuburn* power usage is

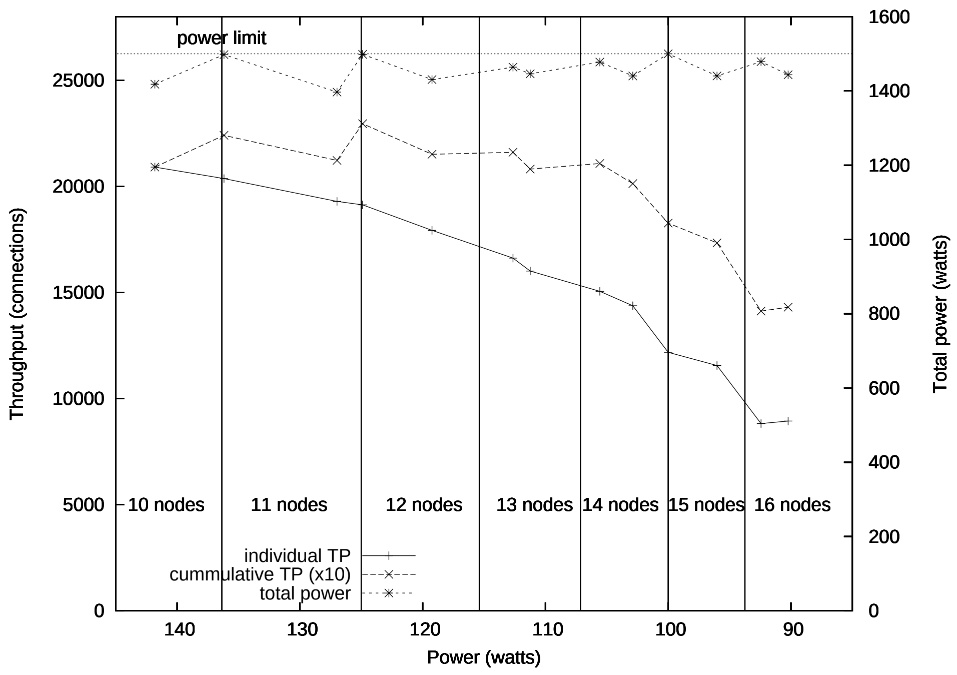

Figure 6.3: LPA performance with $T = \frac{G}{n}$ and a global power limit of 1500 watts. Extrapolation starts with 10 nodes using single node power and throughput measured values.

provided in Appendix C) it still only takes a few seconds for usage and the reducibility factor to stabilize. In normal operation, as the power limit fluctuates 10 to 20 watts, the power usage stabilizes in 1 to 3 seconds. The high variation that is experienced around 190 and 520 seconds is a response to the interval and skip values used in the construction of the reducibility lookup table (Appendix A). The degree of fluctuation exhibited is only due to the TG changing between 3 different reducibility factors.

The previous result utilized one task to show the capability of the TG in maintaining usage to a local node power limit. Figure 6.6 shows two *cpuburn* tasks at different priorities. The priority was set on task startup using the *nice* command. The graph layout is similar to the prior result. The higher priority task is started first. After about 40 seconds, the lower priority task is then started. Shortly thereafter, the local power limit is changed to a lower value. The lower priority is immediately throttled (2 seconds) by a reducibility factor of 0.5. The other higher priority task time is not impacted. Subsequent power limit reductions occur with little impact on reducibility. As the limit reaches 120 watts, both the high and low priority tasks are simultaneously throttled but the higher priority task gains CPU time as the power limit is shown to be stable. Around 180 seconds

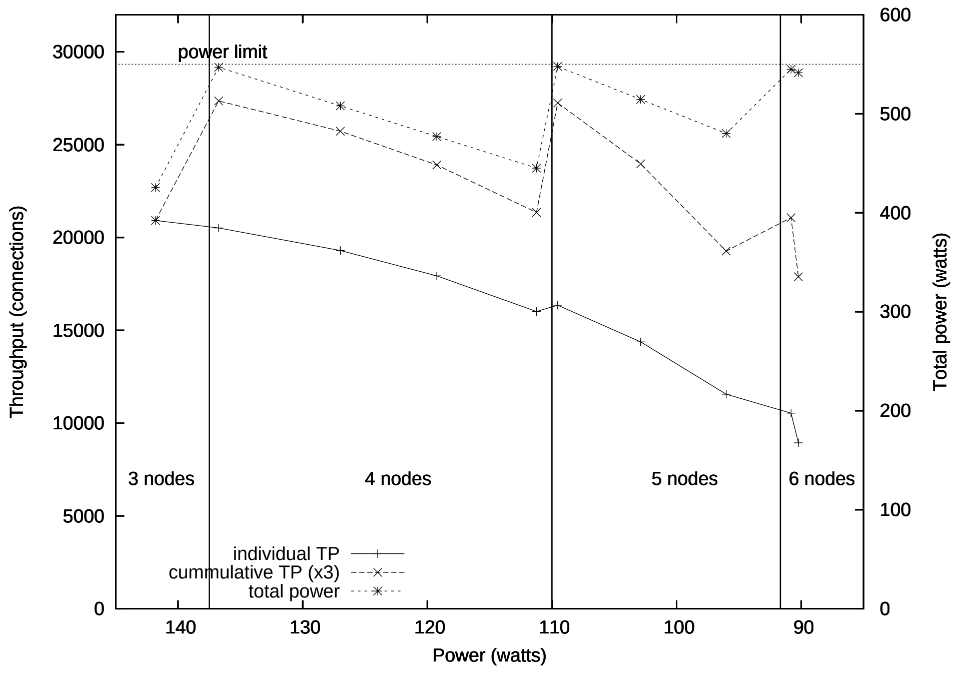

Figure 6.4: LPA performance with $T = \frac{G}{n}$ and a global power limit of 550 watts. Extrapolation starts with 3 nodes using single node power and throughput measured values.

the local power limit is reduced in such a fashion that the high priority task is throttled to about 0.80. However, as subsequent increases in the power limit occur it is also the first task to receive the benefit of increased CPU time.

The preceeding two examples avoid two deficiencies in the current implementation of the Task Governor. First, task restrictions only take effect based on the last measurement cycle that occurred. As a result, if a workload consists of numerous CPU intensive tasks whose execution duration is on the order of several seconds the resulting fluctuation in power can be significant. This is due to waiting up to 2 seconds for the reducibility impact to be measured.

The second deficiency in the TG is due to a lack of a more robust facility for exclusions. This problem is exhibited in the previous graphs as the spikes in power usage occur due to a root-owned periodic task. A more comprehensive strategy for exclusions helps mitigate this problem.

The next result quantifies the first deficiency mentioned previously in the TG. To determine how the TG performs with a highly diverse set of tasks, the first 20 minutes of a linux kernel compile was conducted twice (using gear 0). The first run was unconstrained

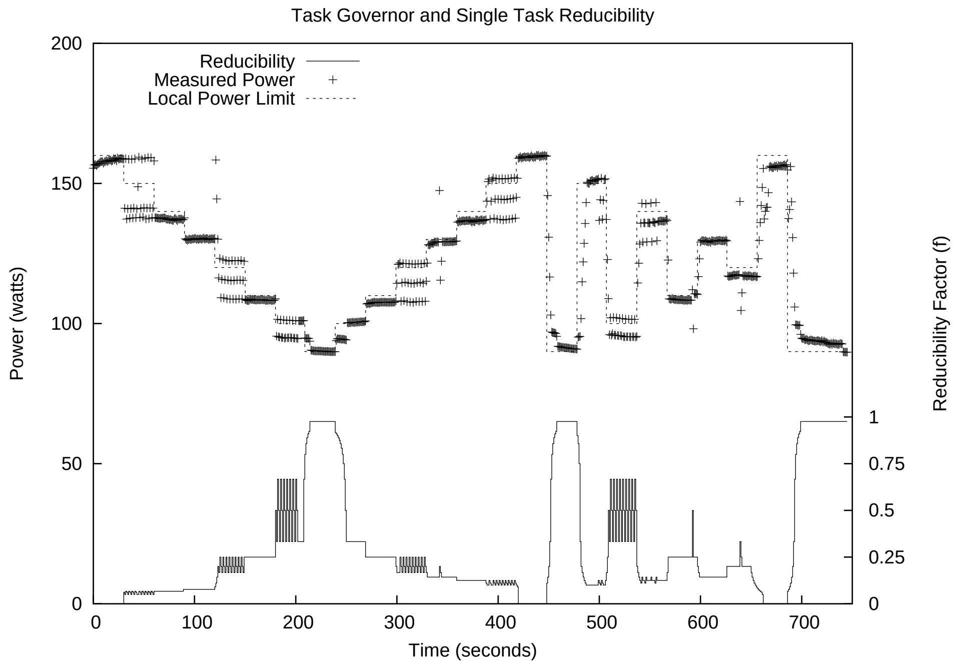

Figure 6.5: Reducibility with one task during local power limit changes. The right side Y axis is not to scale.

and the calculated average power was found to be about 114 watts. The second run used the LPA and the TG with a local power limit set to 115 watts. The calculated average power in this second run was found to be 112 watts. There were a significant number of power usage points above the assigned limit (over 50%). This result highlights the inability of the TG to effectively limit short, bursty tasks. Possible remedies for this deficiency are discussed in Chapter 7.

Although the problems in the TG are worthy to recognize, the initial results in task throttling based on CPU time are encouraging. A similar approach might also be warranted for limiting access to other devices.

6.6 Global Power Agent

There is a corresonding cost, in time and increased node workload, to running the global allocation algorithm. Measurements were taken on a single node to determine the minimum, maximum, and average time to solve the allocation problem during a two minute

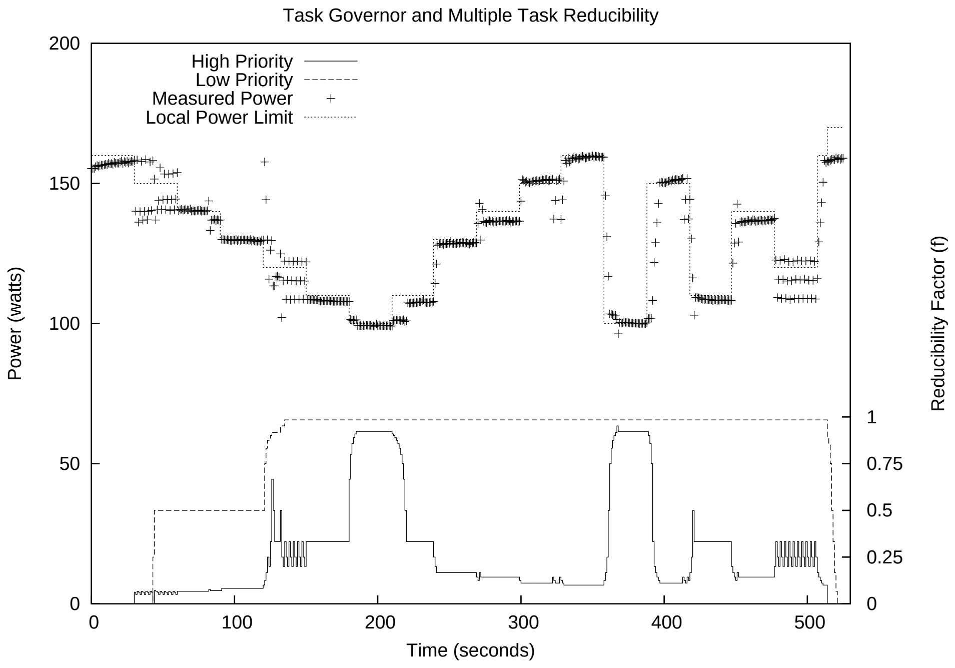

Figure 6.6: Reducibility with two tasks during local power limit changes. The right side Y axis is not to scale.

period with different time intervals (t divides 120). The single node results are shown in Table 6.7. The table column headings reflect the time, in seconds, between processing runs of the algorithm. The more often the algorithm executes, the busier a node becomes. However, the execution runtime decreases due to cache behavior. Running the algorithm more often allows quicker adjustments to node power limits to meet short-term demand. Thus, shorter execution intervals help balance power more efficiently.

Intuitively, the general problem size grows as the number of nodes increases due to the additional constraints. In Figure 6.7, the same measurement method as the prior result was used but in addition, multiple nodes were 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 supports lowering the interval to make the allocation mechanism responsive.

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 forecast model variants, one type relied on a weighted average and the other used

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

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 simultaneously utilized for each inbound broadcast update for each node. To determine the actual forecast for a node, a quick sort was performed to find the model with the smallest error. This model was then used when calculating the new node local power limit. Figure 6.8 shows Model 5, a weighted average with $\alpha = 0.9$ (applied to the most recent data point), was the most effective to classify prior workload and make a prediction for the next interval. Three other models were selected during this benchmark, although the total time they were used was small relative to Model 5. The workload used in this test, and remaining results, was a gcc compiler run of the linux kernel. It is expected that changing the global allocation interval impacts the estimation model; however, the same model selection process still chooses the most applicable model dynamically.

Utilizing this same data, the effectiveness of the LPA in maintaining usage below the local limit, L, on a single node is shown in Figure 6.9. For this and all other results, $L_{min} = 90$ and $L_{max} = 190$. Notice as the global power limit, G, decreases and node workload exists, L subsequently decreases between 220 and 360 seconds. The measured power usage exceeded L within three intervals due to a high power target, T, 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 for the LPA (*i.e.*, high demand with low global and local power limits). It is also important to recognize L represents the next interval limit while usage is a measure of the prior interval. In this and remaining tests the power burst, B, was set to 5 watts. A more conservative policy might choose to increase this value or adjust it automatically based on power usage trends.

In Figure 6.9, recovery in G occurred near $t = 430$ and the node power limit subsequently increased for the next interval. This increase occurred very swiftly and continued

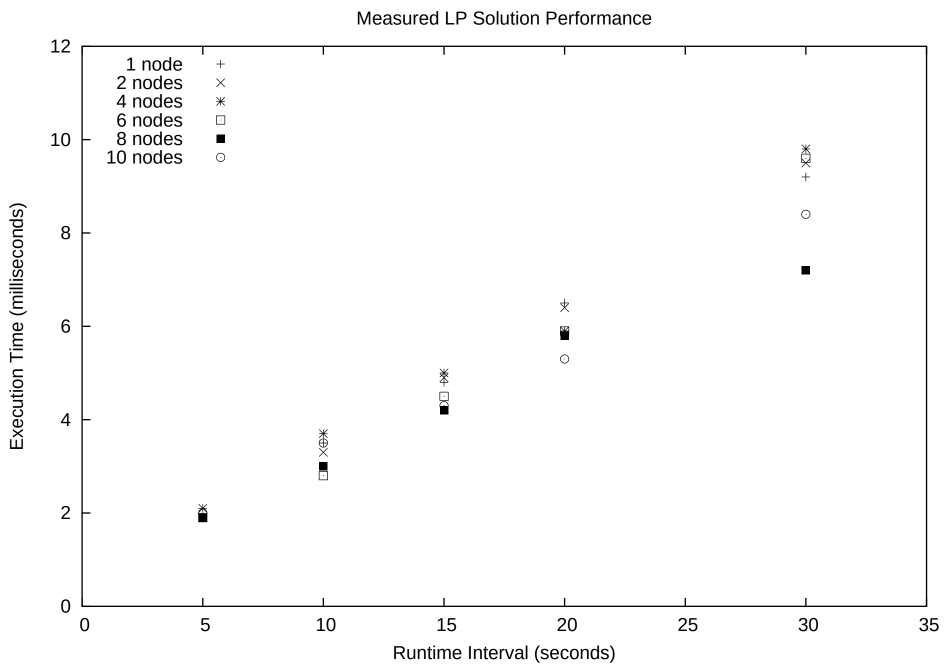

Figure 6.7: Execution run times (microseconds) for solving the global allocation problem with multiple nodes. The execution interval, t, varies from 5 to 30 seconds and divides 120.

to increase while the limit increased for the next few seconds. The global power limit remained well above the local power limit until between $t = 500$ and $t = 600$. At about 600 seconds, a decrease in the global power limit again forced node power reductions for both T and L. Because demand was high, an attempt is still made to boost throughput even with little difference between G and L. Such a condition causes the global power limit to be exceeded. In this and all other results, the global power reserve was $R = 5$. A site with a more conservative requirement could make this higher and provide a larger buffer to account for such conditions.

To maintain the power target, T, in the previous result, the CPU gear utilization is shown in Table 6.8. The full spectrum of available gears was utilized with lower gears used while idle. As previously discussed in Chapter 5, there was a minimum of 5 seconds imposed between gear changes except if the current node usage temporarily exceeded L. In such a condition, the controller was configured to immediately reduce the gear. As can be seen from the data, the system rarely utilized gears 4 and 5. In addition, gear 1 was utilized quite often due to switching between it and both 0 and 2.

To quantify the PICLE result with multiple servers, two cluster nodes were used

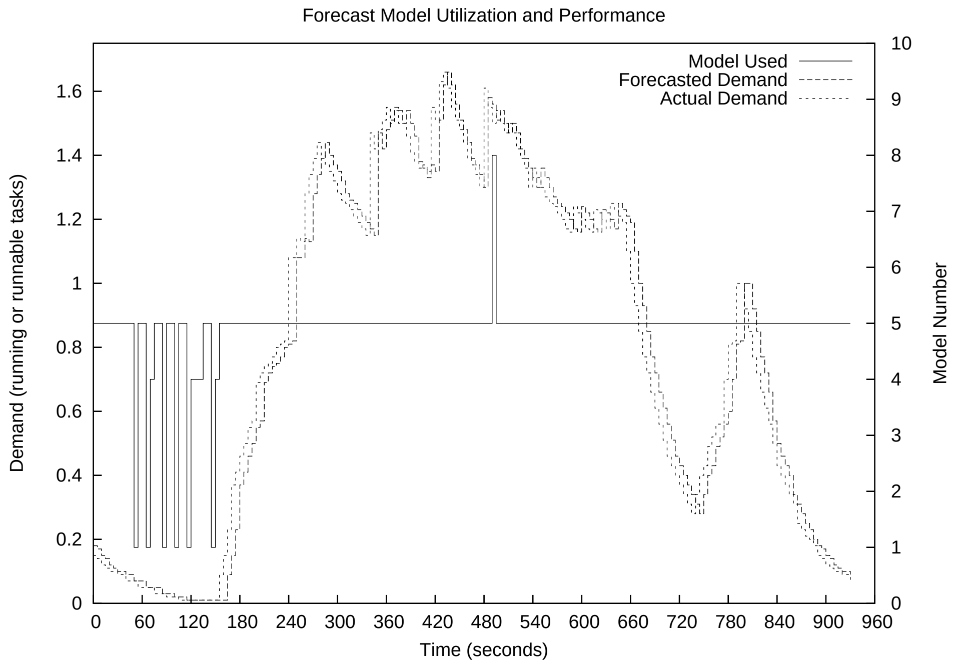

Figure 6.8: Forecast model effectiveness during fluctuations in workload.

starting with $G = 400$ watts (only two multimeters were available). This data from this test is shown in Figure 6.10. The two lower curves reflect the workload demand on each node. Both nodes start out idle, but as one node becomes busy its demand is closely followed by the second node. Eventually, node workload demand is equal and then the node previously busy first, has its demand reduced while the second node remains busy. Finally, demand on both nodes decreases and both eventually become idle. During these changes in demand, global power fluctuates, but the total power allocated never exceeds G. The Y scale on the right for demand is purposefully scaled to ensure node demand is below all other lines for clarity.

In Figure 6.10, 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 even though it is idle and there is little demand on the other node. Notice from about $t = 100$ 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

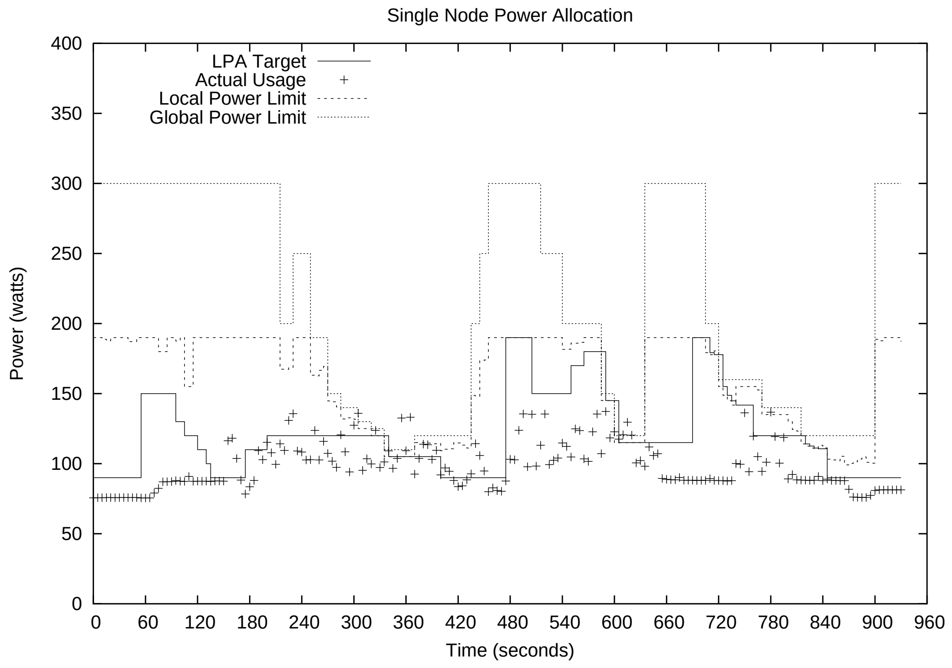

Figure 6.9: Single node effectiveness adherring to the local power limit during global power changes.

node local power limits are balanced appropriately. This situation is later reversed when the demand on Node 2 falls while Node 1 remains busy. Later, as both nodes become idle and the global power supply increases, the local power limit on both nodes subsequently increases to slightly below L_{max}. The cumulative local power limit allocation remains well below G, but this increase allows either node to quickly use power as needed and rapidly react to processing requirements.

With non-uniform power allocation, it is possible to leverage the available power to increase aggregate throughput. To illustrate this benefit, two cluster nodes were configured with a fixed local power limit of $L = 120$ watts each. The value for G was also fixed at 245 watts (includes the 5 watt spare allocation) 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 6.11. In addition, this same graph shows the result of using the GPA to manipulate the local node power limit for non-uniform allocation. The total power allocated in both instances never exceeds G. In addition, the non-uniform power allocation scheme uses more than or equal to the power of the uniform method. However,

Gear	Reference Count	Total Time	Time %
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 6.8: LPA performance state utilization in both references and total time spent (milliseconds) in each gear.

from the start of the run during non-uniform allocation, the busier node transitions to higher gears and increases throughput as a result of being assigned a higher local power limit. The local power limit for the other idle node was also forced to L_{min} (which can't occur in a uniform allocation scheme). After 200 seconds for both uniform and non-uniform allocation runs, the workload was terminated and the resultant throughput found. Throughput was measured based on the number of completed processes completed. Based on the final values, the non-uniform power allocation method provided a 16% gain in throughput.

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. Data centers typically contain a hetereogenuos mix of applications and hardware architectures. As certain cluster nodes become busy, it is often the case that others are lightly loaded or idle. These inequities in workload can be leveraged to overprovision circuits safely. If the workload is inherently uniform or made so with task migration, PICLE is still useful in balancing available power uniformly across all nodes. A data center could also include a mixture of servers (only some that are load balanced) on the same circuit. This situation is also handled in the global allocation strategy proposed in this thesis.

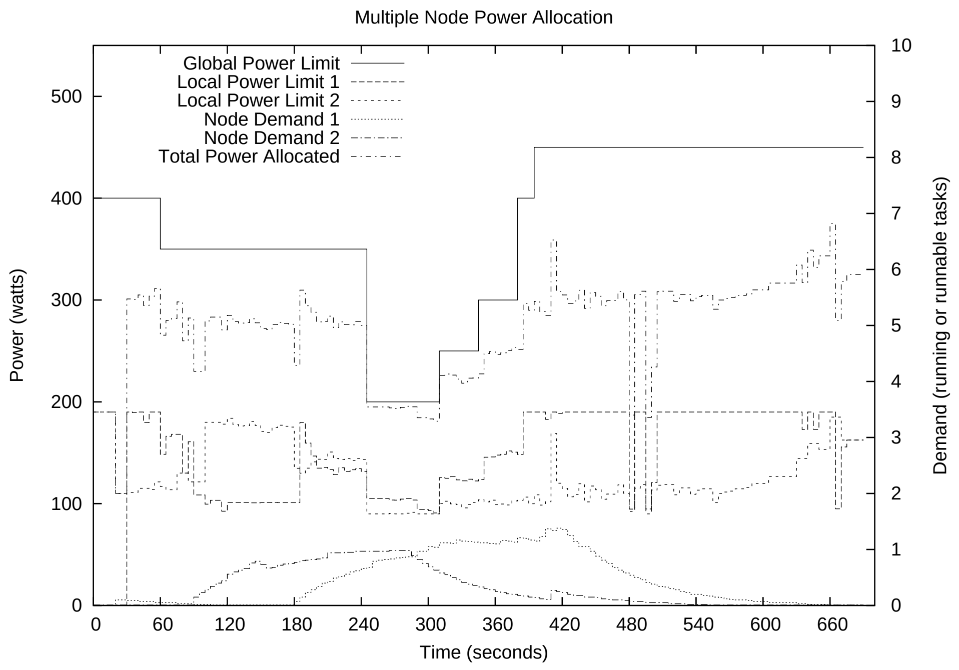

Figure 6.10: Multiple node power allocation with varying workloads and global power fluctuations. Node demand curves are purposefully scaled to be beneath all other lines.

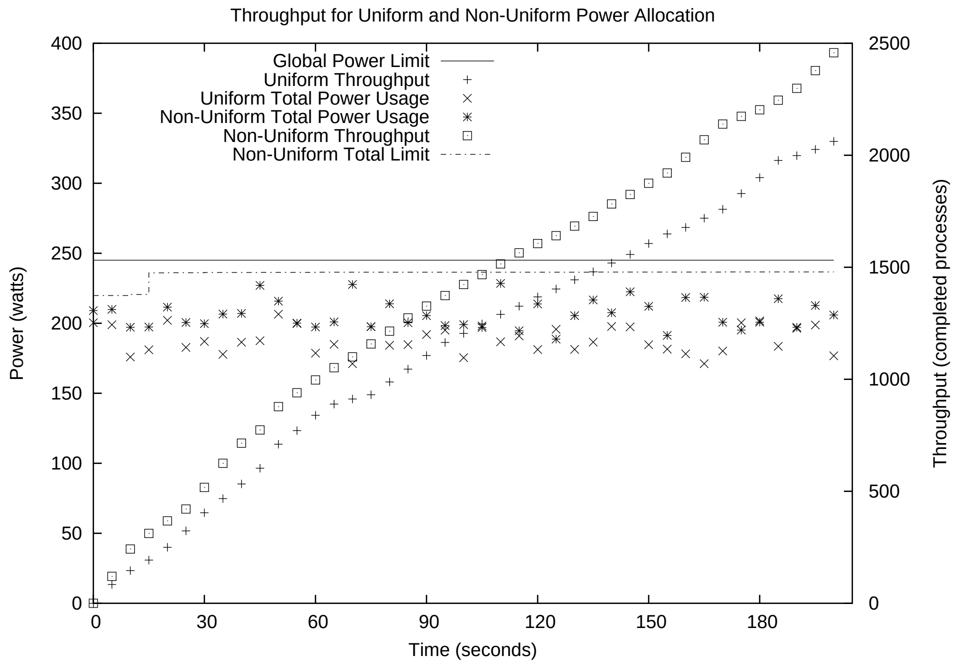

Figure 6.11: Throughput benefit when using non-uniform power allocation and two cluster nodes.

Chapter 7

Future Work

This thesis investigates the potential of overprovisioning in data centers with a focus on increasing throughput based on a defined global power limit. In addition, it proposes 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 provisioning. Power can be automatically controlled within defined limits. Effective control yields greater utilization under a global limit as well as increases in 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.

PICLE is only a preliminary framework and represents an initial foray into the realm of distributed and local power control. Consistent with the approach to make the overall local mechanism perform in a “hands free” operation, additional adjustments to the feedback controller might tune it more efficiently to adapt to the workload. In addition, a more robust local power solution based on multiple power-scalable components should be developed. This controller must account for the power usage and performance of multiple devices (*i.e.* disk, cpu, etc.) and perform the appropriate changes as needed.

Additional work is also needed to ensure short-lived tasks are accounted for in a more efficient implementation of the Resource Controller Module. For instance, it might be useful to throttle all tasks initially or to provide an inheritance mechanism in the throttling implementation. It may also be useful to modify this component to support differentiated or class-based service. This module should also be designed to throttle more than the CPU as power consumption of all resources is further developed in the local controller.

Additional modifications to the Task Governor offer the ability to increase local throughput by overprovisioning power at the local level. This is accomplished through intelligent task monitoring and throttling those tasks which are shown to be higher power consumers, subject to their performance minimums. This throttling is possible irregardless of whether the local power limit is exceeded. Throttling increases CPU time to allow more tasks to exist than otherwise possible, but must also be balanced with a potential increase in power consumption. Such modifications should also be consistent with the notion of utilizing existing operating system priorities.

The global allocation proposed in this thesis represents a novel solution to the distribution of power for heterogeneous architectures and workloads. However, greater focus is needed to address current shortfalls in initial node power on, synchronization of broadcast data and timing irregularities, and simultaneous cluster node adjustments. These changes will enable an even tighter bound on the global power limit. In addition, the short-term forecast model proposed should be supplemented with additional medium or long-term models. Such models could provide the ability to reserve power more efficiently than simply reacting to short-term demand.

PICLE is an effective framework to handle safe overprovisioning based on 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. The distributed and autonomic power control policy proposed in this thesis not only yields gains in throughput, but also balances power across a heterogeneous mix of applications and architectures.

Bibliography

[1] N.D. Adiga et al. An overview of the BlueGene/L supercomputer. In *Supercomputing 2002*, November 2002.

[2] AMD Athlon processor x86 code optimization guide. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs%/22007.PDF`, February 2002.

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

[4] BIOS and kernel developer's guide for the AMD athlon 64 and AMD opteron processors. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs%/26094.PDF`, April 2004.

[5] Manish Anand, Edmund Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Mobicom*, September 2003.

[6] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Ghosts in the machine: interfaces for better power management. In *Proceedings of the 2nd international conference on Mobile systems, applications and services*, pages 23–35, 2004.

[7] APC power distribution products. `http://www.apc.com/products/`, September 2004.

[8] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Measurement and Modeling of Computer Systems*, pages 90–101, 2000.

[9] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau.

Profile-based Dynamic Voltage Scheduling Using Program Checkpoints, page 168. IEEE Computer Society, 2002.

[10] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation OSDI'99*, February 1999.

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

[12] Frank Bellosa, Simon Kellner, Martin Waitz, and Andreas Weissel. Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'03)*, September 2003.

[13] L. Benini, A. Bogliolo, and G. De Micheli. Monitoring system activity of OS-directed dynamic power management. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '98*, 1998.

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

[15] Pat Bohrer, Elmootazbellah Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. The case of power management in web servers. In Robert Graybill and Rami Melham, editors, *Power Aware Computing*. Kluwer/Plenum, 2002.

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

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

[18] The common gateway interface. `http://hoohoo.ncsa.uiuc.edu/cgi/`, October 2004.

[19] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[20] W. Chen, M. Dubios, and P. Stenstrom. Integrating complete-system and user-level performance/power simulators: The simwattch approach. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, 2003.

[21] Eui-Young Chung, Luca Benini, and Giovanni De Micheli. Dynamic power management using adaptive learning tree. In *Proceedings of the 1999 IEEE/ACM international conference on computer-aided design*, pages 274–279, 1999.

[22] V. Delaluz, A. Sivasubramanian, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Scheduler-based DRAM energy management. In *Proceedings of the Design Automation Conference (DAC '02)*, Jun 2002.

[23] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Computing*, 1995.

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

[25] Elmootazbellah Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *USITS '03*, 2003.

[26] E.N. (Mootaz) Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Workshop on Mobile Computing Systems and Applications*, February 2002.

[27] W. M. Felter, T. W. Keller, M. D. Kistler, C. Lefurgy, K. Rajamani, R. Rajamony, F. L. Rawson, B. A. Smith, and E. Van Hensbergen. On the performance and use of dense servers. *IBM Journal of Research and Development*, 47(56):671–688, September 2003.

[28] 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.

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

[30] 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.

[31] Jason Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Transactions on Computer Systems (TOCS)*, 22(2):137–179, May 2004.

[32] Chris Gniady, Y. Charlie Hu, and Yung-Hsiang Lu. Program counter based techniques for dynamic power management. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture*, February 2004.

[33] 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.

[34] 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.

[35] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Reducing disk power consumption in servers with DRPM. *IEEE Computer*, pages 41–48, December 2003.

[36] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Mobile Computing and Networking*, pages 130–142, 1996.

[37] 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.

[38] M. Huang, J. Renau, and J. Torrellas. Profile-based energy reduction in high-performance processors. In *Proceedings of the 4th Workshop on Feedback-directed and Dynamic Optimizations*, December 2001.

[39] 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.

[40] 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.

[41] Nagarajan Kandasamy, Sherif Abdelwahed, and John 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.

[42] Ramakrishna Kotla, Soraya Ghiasi, Tom Keller, and Freeman Rawson. Scheduling processor voltage and frequency in server and cluster systems. In *Workshop on High-Performance, Power-Aware Computing (HPPAC)*, 2005.

[43] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commerical servers. *IEEE Computer*, 36(12):39–48, December 2004.

[44] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *USENIX Winter*, pages 279–291, 1994.

[45] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. In *Proceedings of the ACM SIGMETRICS 2003 Conference*, pages 160–171, June 2003.

[46] Yingmin Li, David Brooks, Zhigang Hu, Kevin Skadron, and Pradip Bose. Understanding the energy efficiency of simultaneous multithreading. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '04*, pages 44–49, 2004.

[47] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Power-aware operating systems for interactive systems. *IEEE Transactions on Very Large Scale Integration Systems*, 10(2), April 2002.

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

[49] Robert J. Minerick, Vincent W. Freeh, and Peter 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.

[50] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, pages 35–44, 2002.

[51] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *WISP*, pages 59–67, Madison, WI, June 1998.

[52] Trevor Mudge. Power: A first class architectural design constraint. *IEEE Computer*, 34(4):52–57, April 2001.

[53] Enric Musoll. Predicting the usefulness of a block result: a micro-architectural technique for high-performance low-power processors. In *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, pages 238–247, 1999.

[54] Enric Musoll. Speculating to reduce unnecessary power consumption. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(4):509–536, November 2003.

[55] Subbarao Palacharla, Norman P. Jouppi, and J.E. Smith. Complexity-effective superscalar processors. In *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*, pages 206–218. ACM Press, 1997.

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

[57] 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.

[58] 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.

[59] 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.

[60] Michael D. Powell, Mohamed Gomaa, and T. N. Vijaykumar. Heat-and-run: leverage smt and cmp to manage power density through the operating system. In *Proceedings of the 11th international conference on architectural support for programming languages and operating systems*, pages 260–270, 2004.

[61] Robert Redelmeier. cpuburn. `http://pages.sbcglobal.net/redelm/`, June 2001.

[62] Dale E. Seborg, Thomas F. Edgar, and Duncan A. Mellichamp. *Process Dynamics and Control*, chapter 8. John Wiley & Sons, Inc., second edition, 2004.

[63] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, and Kevin Skadron. Power-aware QoS management in web servers. In *24th Annual IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003.

[64] Amin Vahdat, Alvin Lebeck, and Carla 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.

[65] M. Valluri and L.K. John. Is compiling for performance = compiling for power? In *Proceedings of the 5th Annual Workshop on Interaction between Compilers and Computer Architectures*, 2001.

[66] Andreas Weissel and Frank Bellosa. Dynamic thermal management for distributed systems. In *Proceedings of the First Workshop on Temperature-Aware Computer Systems*, June 2004.

[67] Andreas Weissel, Bjorn Beutel, and Frank Bellosa. Cooperative io - a novel io semantics for energy-aware applications. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.

[68] Giovanni De Micheli Yung-Hsiang Lu, Luca Benini. Operating-system directed power reduction. In *International Symposium on Low Power Electronics and Design*, pages 37–42. Stanford University, July 2000.

[69] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Currentcy: Unifying policies for resource management. In *USENIX 2003 Annual Technical Conference*, June 2003.

Appendix A

Task Governor Reducibility

Reducibility factors are used in the Task Governor (TG) to limit CPU time for fine-grain task throttling. The following table is used in the determination of interval and skip in the step algorithm discussed in Chapters 4 and 5. An increasing interval implies a longer execution time before that task is interrupted and placed in the wait queue. In contrast, an increasing skip value implies a longer time in the wait queue. The resolution of both skip and interval is in milliseconds. Interval is accounted for across one or more context switches and skip is not a guarantee a task will be executed immediately after the specified value.

To codify an example using the table below, a task is running and it is determined by the TG an 18% reduction in task CPU time is needed to meet the local power limit. Based on this reduction, a lookup is performed and the first which exceeds this reduction is used. The rate of throttling change is represented with h (see Chapter 5.4.3). In this first lookup (no prior task reduction was done), $h = f = 0.2$, $s = 1$ and $i = 4$ so the impact of reduction error (between target value and f) is to allocate less CPU time. A further reduction to this task (*i.e.*, a decrease in interval to $i = 3$) would account for a task time impact change of $h = \Delta f = 0.20 - 0.25 = -0.05$. Any target value above the highest f assigns this top value s and i. In addition, any task time increase below the lowest f removes any restrictions.

Based on the calculated value for h, task time reduction is performed in the TG step algorithm. For additional details on the TG and the application of h, see Chapters 4.3.2 and 5.4.3.

Skip (s)	Interval (i)	Reducibility Factor (f)
60	1	0.983607
40	1	0.975610
20	1	0.952381
12	1	0.923077
11	1	0.916667
10	1	0.909091
9	1	0.900000
8	1	0.888889
7	1	0.875000
6	1	0.857143
5	1	0.833333
4	1	0.800000
3	1	0.750000
2	1	0.666667
1	1	0.500000
1	2	0.333333
1	3	0.250000
1	4	0.200000
1	5	0.166667
1	6	0.142857
1	7	0.125000
1	8	0.111111
1	9	0.100000
1	10	0.090909
1	11	0.083333
1	12	0.076923
1	13	0.071429
1	14	0.066667
1	15	0.062500
1	20	0.050000
1	40	0.025000
1	60	0.016667

Table 7.1: Task Governor reducibility factors and their associated interval and skip values.

Appendix B

AMD CPU Performance State Transitions

The AMD CPU performance state transition algorithm is defined in [4]. The system BIOS normally contains the tables used to lookup performance states based on the revision of a processor. The frequency-to-voltage relationships, designed for each processor, are used in connection with the limitation that no direct frequency change can occur greater than 200 MHz. In addition, portal core frequencies are defined based on a range of values from the chip minimum core frequency (*i.e.*, the first is double the minimum). A transition from the minimum core frequency to a portal core frequency can occur in one step. Any other changes utilize a step of 200 Mhz. The actual algorithm in the *cpufreq* software driver has three phases. During the first phase the voltage is transitioned to the level required to support frequency transitions. Second, the processor frequency is then changed to the frequency associated with the requested new state. Finally, the voltage is transitioned to the value specified with the requested performance state.

The first phase in the transition algorithm for the processor is further decomposed into additional steps. First, to determine the voltage needed to support the transition, the requested voltage is added to a ramp voltage offset (RVO). This is done if the requested frequency is greater than the current performance state. Otherwise, the voltage in the current performance state is added to the voltage indicated by the RVO. These values are further subjected to a maximum specified voltage step (MVS). Each step increases voltage to the minimum of MVS or the desired voltage plus the RVO. In addition, after each step a voltage stablization time (VST) is enforced.

Measured State Transition Times

Performance state transitioning costs were measured on one cluster node. For each transition, a new frequency was provided to the *cpufreq* kernel module. This frequency corresponds to an appropriate voltage and frequency pair in table 6.1. To measure the time for a transition, this module's notifier interface was used in an event timing kernel module. Thus, prior to the transition the CPUFREQ_PRECHANGE event was used to time stamp a variable later used during the CPUFREQ_POSTCHANGE event. The time in the prechange event is subtracted from the postchange event time and this value was reported to microsecond precision.

A total of 100 transitions for each performance state change were performed. The average, minimum, and maximum were then calculated across all measured values. Thus, approximately 4200 total transition times were collected and between each measurement, a 5 second delay was imposed to ensure general system idleness. To remove ambiguity in measurements, the PM Timer was used. This timing source is guaranteed to be independent from microprocessor (and core clock) performance state changes. To minimize the changes to the PowerNow *cpufreq* driver for exposing addition performance states, the portal frequencies were adjusted. These changes do add additional overhead to the standard algorithm; however, these costs are accounted for, as applicable, in the results presented in this thesis.

From (MHz)	To (MHz)	Average	Minimum	Maximum
2000	1800	355.55	340	375
2000	1600	554.66	536	573
2000	1400	850.63	836	868
2000	1200	1138.78	1122	1158
2000	1000	1954.74	1913	1990
2000	800	568.62	550	587
1800	2000	785.46	767	814
1800	1600	278.68	263	300
1800	1400	556.72	537	591
1800	1200	853.07	831	873
1800	1000	1522.51	1483	1556
1800	800	292.44	276	324
1600	2000	1436.75	1406	1470
1600	1800	566.70	551	588
1600	1400	279.47	264	295
1600	1200	555.84	538	574
1600	1000	1122.03	1087	1146
1600	800	289.90	274	310
1400	2000	2204.94	2185	2231
1400	1800	1218.49	1198	1263
1400	1600	279.32	262	299
1400	1200	279.61	264	302
1400	1000	746.02	725	783
1400	800	289.22	272	308
1200	2000	3415.02	3375	3524
1200	1800	2259.24	2242	2274
1200	1600	556.16	536	573
1200	1400	280.20	264	301
1200	1000	362.66	348	383
1200	800	575.44	556	593
1000	2000	5015.28	4969	5068
1000	1800	3597.30	3556	3645
1000	1600	853.51	826	881
1000	1400	564.01	542	593
1000	1200	280.72	265	302
1000	800	297.68	281	316
800	2000	5647.90	5602	5727
800	1800	3892.10	3870	3927
800	1600	289.13	273	316
800	1400	289.70	272	322
800	1200	1134.16	1092	1159
800	1000	2141.23	2107	2205

Table 7.2: Performance state kernel transition cost (microseconds) between different gears for an AMD 3000+ CPU.

Appendix C

AMD 3000+ CPU Power Consumption

System idle power was measured for a cluster node and is shown in the table below. In addition, maximum power was approximated using *cpuburn*, a tool designed to overheat and stress a CPU [61]. Although the aforementioned tool is useful to determine an upper bound for power measurements, it is not endorsed by any manufacturer and may not represent the true upper bound experienced across all applications.

For both system power measurement types (idle and with load), 100 readings were taken with a 1 second delay between successive readings. All average, maximum, and minimum values are in watts. The gear corresponds to the frequency and voltage settings given in Table 6.1. Measured system idle power at each gear reflects reduced power usage from the `HLT` instruction.

CPU	System Idle Power			System Maximum Power		
Gear	Average	Minimum	Maximum	Average	Minimum	Maximum
0	87.28	86.81	88.00	158.88	156.20	160.37
1	84.48	84.17	85.57	135.80	133.67	136.48
2	82.95	82.46	83.58	128.38	121.73	125.11
3	81.86	81.38	82.64	112.62	111.58	113.81
4	79.76	79.44	80.45	100.84	100.01	101.40
5	78.07	77.76	78.55	91.89	91.38	92.56
6	75.96	75.71	76.38	85.52	85.07	86.03

Table 7.3: Measured system power at idle and with a CPU-intensive load (*cpuburn*).