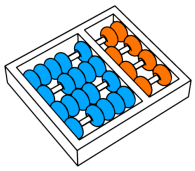
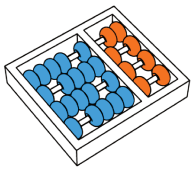


Leonardo de Paula Rosa Piga

"Modeling, Characterization, and Optimization of Web Server Power in Data Centers"

"*Modelagem, Caracterização e Otimização de Potência em Centro de Dados*"

CAMPINAS
2013



University of Campinas
Institute of Computing

Universidade Estadual de Campinas
Instituto de Computação

Leonardo de Paula Rosa Piga

"Modeling, Characterization, and Optimization of Web Server Power in Data Centers"

Supervisor: **Prof. Dr. Sandro Rigo**
Orientador(a):

Co-Supervisor: **Dr. Reinaldo Alvarenga Bergamaschi**
Co-orientador(a):

"Modelagem, Caracterização e Otimização de Potência em Centro de Dados"

PhD Thesis presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a PhD degree in Computer Science.

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.

This volume corresponds to the final version of the Thesis defended by Leonardo de Paula Rosa Piga, under the supervision of Prof. Dr. Sandro Rigo.

Este exemplar corresponde à versão final da Tese defendida por Leonardo de Paula Rosa Piga, sob orientação de Prof. Dr. Sandro Rigo.

Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

P622m

Piga, Leonardo de Paula Rosa, 1985-
Modeling, characterization, and optimization of web server power in data centers / Leonardo de Paula Rosa Piga. – Campinas, SP : [s.n.], 2013.

Orientador: Sandro Rigo.
Coorientador: Reinaldo Alvarenga Bergamaschi.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Arquitetura de computador. 2. Centros de processamento de dados. 3. Macromodelagem de potência. 4. World Wide Web - Servidores. 5. Energia elétrica - Conservação. I. Rigo, Sandro,1975-. II. Bergamaschi, Reinaldo Alvarenga. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

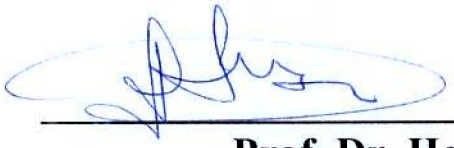
Informações para Biblioteca Digital

Título em outro idioma: Modelagem, caracterização e otimização de potência em centro de dados
Palavras-chave em inglês:
Computer architecture
Data processing service centers
Power macromodeling
Web servers
Electric power conservation
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Sandro Rigo [Orientador]
Philippe Olivier Alexandre Navaux
Hermes Senger
Edmundo Roberto Mauro Madeira
Rodolfo Jardim de Azevedo
Data de defesa: 08-11-2013
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

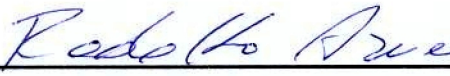
Tese Defendida e Aprovada em 08 de novembro de 2013, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Philippe Olivier Alexandre Navaux
INF / UFRGS

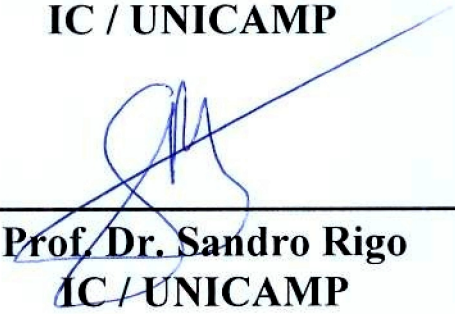


Prof. Dr. Hermes Senger
DC / UFSCAR

Prof. Dr. Edmundo Roberto Mauro Madeira
IC / UNICAMP



Prof. Dr. Rodolfo Jardim de Azevedo
IC / UNICAMP



Prof. Dr. Sandro Rigo
IC / UNICAMP

Institute of Computing /*Instituto de Computação*
University of Campinas /*Universidade Estadual de Campinas*

Modeling, Characterization, and Optimization of Web Server Power in Data Centers

Leonardo de Paula Rosa Piga[1]

November 08, 2013

Examiner Board/*Banca Examinadora*:

- Prof. Dr. Sandro Rigo (Supervisor/*Orientador*)
- Prof. Dr. Philippe Olivier Alexandre Navaux
  INF - UFRGS
- Prof. Dr. Hermes Senger
  DC - UFSCAR
- Prof. Dr. Edmundo Roberto Mauro Madeira
  IC - Unicamp
- Prof. Dr. Rodolfo Jardim de Azevedo
  IC - Unicamp

[1]Financial support: FAPESP scholarship (process 2010/05389-5) 2009–2011; AMD Research 2012-2013

Abstract

To keep up with an increasing demand for computational resources, IT companies need to build facilities that host hundreds of thousands of computers, the data centers. This environment is highly dependent on electrical energy, a resource that is becoming expensive and limited. In this context, this thesis develops a global data center-level power and performance optimization approach for Web Server data centers. It presents a power measurement framework for commodity servers, develops empirical models for estimating the power consumed by Web servers, and implements one of the global power optimization heuristics on a state-of-the-art, high-density SeaMicro SM15k cluster by AMD.

The power measuring framework is composed of a custom made board, which is able to capture the power consumption; a data acquisition device that samples the measured values; and a piece of software that manages the framework.

We show a novel method for developing full system Web server power models that prunes model parameters and reduces non-linear relationships among performance measurements and system power. The Web server power models use as parameters performance indicators read from the machine internal performance counters. We evaluate our approach on an AMD Opteron-based Web server and on an Intel i7-based Web server. Our best model displays an average absolute error of 1.92% for the Intel i7 server and 1.46% for AMD Opteron as compared to actual measurements, and 90^{th} percentile for the absolute percent error equals to 2.66% for Intel i7 and 2.08% for AMD Opteron.

We deploy the global power management system in a state-of-the-art SeaMicro SM15k cluster. The implementation relies on the concept of Virtual Power States, a combination of CPU utilization rate to the P/C power states available in modern processors, and on our global optimization algorithm called *Slack Recovery*. We also propose and implement a novel mechanism to control utilization rates in each server, a key aspect of our power/performance optimization system. Experimental results show that our *Slack Recovery*-based system can reduce up to 16% of the power consumption when compared to the Linux *performance* governor and 6.7% when compared to the Linux *ondemand* governor.

Resumo

Para acompanhar uma demanda crescente pelos recursos computacionais, empresas de TI precisaram construir instalações que comportam centenas de milhares de computadores chamadas centro de dados. Este ambiente é altamente dependente de energia elétrica, um recurso que é cada vez mais caro e escasso. Neste contexto, esta tese apresenta uma abordagem para otimizar potência e desempenho em centro de dados Web. Para isto, apresentamos uma infraestrutura para medir a potência dissipada por computadores de prateleiras, desenvolvemos modelos empíricos que estimam a potência de servidores Web e, por fim, implementamos uma de nossas heurísticas de otimização de potência global em um aglomerado de nós de processamento chamado AMD SeaMicro SM15k.

A infraestrutura de medição de potência é composta por: uma placa personalizada, que é capaz de medir potência e é instalada em computadores de prateleira; um conversor de dados analógico/digital que amostra os valores de potência; e um *software* controlador.

Mostramos uma nova metodologia para o desenvolvimento de modelos de potência para servidores Web que diminuem a quantidade de parâmetros dos modelos e reduzem as relações não lineares entre medidas de desempenho e potência do sistema. Avaliamos a nossa metodologia em dois servidores Web, um constituído por um processador AMD Opteron e outro por processador Intel i7. Nossos melhores modelos tem erro médio absoluto de 1,92% e noventa percentil para o erro absoluto de 2,66% para o sistema com processador Intel i7. O erro médio para o sistema composto pelo processador AMD Opteron é de 1,46% e o noventa percentil para o erro absoluto é igual a 2,08%.

A implantação do sistema de otimização de potência global foi feita em um aglomerado de nós de processamento SeaMicro SM15k. A implementação se baseia no conceito de *Virtual Power States*, uma combinação de taxa de utilização de CPU com os estados de potência P e C disponíveis em processadores modernos, e no nosso algoritmo de otimização chamado *Slack Recovery*. Propomos e implementamos também um novo mecanismo capaz de controlar a utilização da CPU. Nossos resultados experimentais mostram que o nosso sistema de otimização pode reduzir o consumo de potência em até 16% quando comparado com o governador de potência do Linux chamado *performance* e em até 6,7% quando comparado com outro governador de potência do Linux chamado *ondemand*.

Agradecimentos

Eu gostaria de agradecer primeiramente aos meus pais, que sempre me apoiaram em toda a minha vida. Ao meu pai Sérgio pelo exemplo de dedicação ao trabalho, busca pelos objetivos pessoais, e pelos primeiros ensinamentos na área das exatas e em computação. À minha mãe Érica por toda a paciência do mundo, todo o suporte em todos os momentos, quando estava perto ou mesmo quando estava longe, e principalmente por todo amor que um filho pode ter.

Aos meus avós Acary e Glória. Avô Acary, que sempre me estimulou a buscar conhecimento e demonstrava orgulho a cada novo objetivo alcançado, mas que infelizmente partiu alguns meses antes de poder ver o fim dessa jornada de perto. À minha avó Glória pelas longas conversas e pelo incentivo ao estudo.

À minha namorada Junielles, que esteve ao meu lado no final deste trabalho, que teve toda a paciência do mundo em todos os momentos e que me incentivou até o último segundo.

Aos meus colegas de turma de graduação: Luis e Matheus. Aos meus amigos de laboratório: Auler, Baldassin, Cardoso, Ecco, Gabriel, George, João, Klein, Nicácio, Portavales, Raoni e Yang. Este trabalho tem um pouco de cada um de vocês e tenho muito orgulho de ter convivido com todos vocês.

Agradeço a todos da AMD, em especial a Alan, Andy, Jay, Keith, Patryk. Mas principalmente ao Mauricio que confiou em meu trabalho e abriu as portas para que eu pudesse fazer um bom trabalho na AMD Research. A experiência adquirida no meu estágio foi fundamental para o meu crescimento como pesquisador.

Agradeço aos professores que passaram por minha vida. Carrego parte de cada um deles comigo, em especial aos professores do LSC: Borin, Côrtes, Guido, Ducatte, Rodolfo.

Ao meu coorientador Reinaldo, por ter colaborado muito com as ideias deste projeto e por ter me passado muito da sua experiência na escrita de artigos.

Finalmente, gostaria de agradecer ao meu orientador Sandro Rigo, que me orienta desde 2006 durante a iniciação científica e que contribuiu de maneira fundamental para o meu desenvolvimento como pesquisador.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The shift towards increasing demand in computational resources has forced companies to build facilities hosting hundreds of thousands of computers called data centers. A variety of services, from search (e.g. Google and Yahoo), to e-commerce (e.g. Amazon and e-Bay), to stock trading (e.g. Fidelity and e-Trade) and media streaming (e.g. YouTube) rely on the computing capabilities of data centers. This comes at a price of higher operational costs which include the management of these installations, the electricity costs, and environmental impacts due to the increased power consumption.

The power associated with the IT equipment in a data center includes the power of the servers, the power required by the cooling, and auxiliary equipment (e.g. power distribution units, switching, back up power). The power consumption of the actual server drives the power needs for auxiliary equipment and cooling; thus, reducing server power has a direct effect on reducing data center power as a whole. Futhermore, saving energy implies in reducing costs because the total cost of ownership of a data center is proportional to its power consumption.

In 2005, the energy consumption of total servers corresponded to about 0.6% of total electricity consumed in the USA. If auxiliary equipment was taken into account, this share increases to 1.2% [36]. In 2010, the electricity used in US data centers accounted for between 1.7 and 2.2% of total electricity use [37]. In this context, power-aware computing has emerged as a concern in data centers.

In 2009, Schulz claimed that 50% of electrical power was spent on cooling in a typical data center [62]. However, on modern data centers, this problem is solved by building the facilities in cold places such as arctic region or close to rivers where they can use water to cool servers. Data-centers efficiency can be measured using a metric called Power Usage Effectiveness (PUE), which is calculated by dividing the total electric power used in a facility by the power brought to the computing units (e.g. computers, networking equipment). By 2006 the PUE of 85% of data centers was about 2, that is, for each

watt used for computation other additional two watts were spent by cooling and auxiliary equipments [46]. By the end of 2010, Google's data centers achieved an overall PUE of 1.1 [61] a huge improvement.

The share of computation power to total is increasing while the contribution of auxiliary equipment for total power is being reduced. Therefore, two important ways to reduce computation power are as follows: building energy efficiency into the design of system and components; and adaptively controlling the power consumption of systems in response to fluctuations in environment conditions or workload. The former method includes circuit techniques to reduce dynamic power such as clock gating, where the clock signal to inactive processor parts is disable; and adding multiple voltage and frequency levels in system components. The latter approach requires intelligent methods to switch servers on and off and to choose the operating voltage and frequency levels in order to optimize power and performance. This thesis is inserted in the last class as it develops and implements power optimizations on Web Server clusters.

Current server CPUs rely on a hierarchy of states controlled by the operating system, middleware, and hardware, in order to trade off power and performance at runtime. These states are typically called S-States, C-States and P-States [13,49]. S-States determine the overall system power state, such as sleep, or hibernate. C-States are power-saving states, ranging from C_0 to C_n, with C_0 being the active state, and the others being idle states with increasing power savings and increasing time overhead in switching among them. P-States are different stages of processor frequency and voltage which can be used to trade-off performance and power consumption within active states. The ACPI [2] standard provides platform-independent interfaces for accessing the state tables and for applying power management policies as determined by the OS or upper software layers. For example, according to ACPI tables found in operating systems the Intel i7 860 processor cores have 14 P-states, or 14 different levels of frequency and voltage. While these internal control mechanisms are very useful to optimize power on a single CPU, they result in a local optimization.

Figure 1.1 illustrates the high-level functionality of our system. It consists of two parts: a data center and a power/performance manager. The data center has a monitoring system that captures the sensor data, that is, readings of the power and performance values of each core in the data center. The manager takes into account any external policy and the input workload, and determines on-the-fly the best configuration of power states for all cores (i.e., which core needs to be shut down, or brought up, or have its power state or utilization rate changed). It then passes this new configuration and the new workload distribution back to the data center, which is then reconfigured while the workload is running. Our methodology contrasts to prior works as it is the first to relate the concept of utilization rate to the P/C power states to develop a global optimization

approach to accomplish a global data center-level optimization of power and performance.

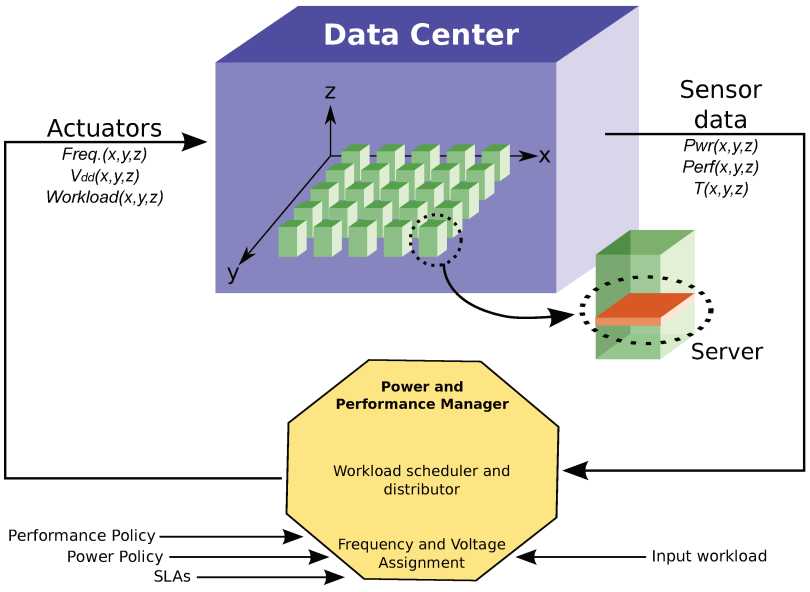


Figure 1.1: Model and simulation environment that evaluates the load-balancing algorithms

The algorithms to explore power and performance trade-offs are implemented in the manager. We used an algorithm based on ILP (integer linear programming) and another based on heuristics for selecting the states to each core, in order to optimize the power and/or performance. Both algorithms were tested in a simulation environment and the one based on heuristics, the *Slack Recovery*, was implemented on a state-of-the-art, highly dense SeaMicro SM15k cluster by AMD [3] along with a novel mechanism to control the CPU utilization rate of each server as part of the *Slack Recovery* implementation.

Most of the power-aware policies require measuring the system power consumption. However, power monitoring tools may not be available on commodity servers because it would increase the price of the servers. In this way, we present a power measuring infrastructure that can be used to measure commodity server power.

A usual approach to mitigate the lack of power monitoring tools is to derive linear power models based on system performance measurements [5, 9, 20, 31, 32]. However, as part of this work we found that Web Server applications, which tend to be I/O-bound, cause non-linear effects in the relationship between power and system architectural measurements. We present Web Server power modeling techniques to overcome this issue.

1.1 Contributions

This thesis presents a custom-made power measuring infrastructure that is able to measure power on commodity servers; it investigates Web Server power modeling techniques and develops accurate empirical models for estimating the power consumed by Web Servers considering all their major parts, such as processors, disk, memory, network, and other motherboard components. The power of processors is characterized for their different *P-states* and models are developed for each *P-state*. It applies correlation-based feature selection (CFS) [27] to prune correlated model parameters and k-means clustering to soften non-linear relationship among server measurements and power consumption on linear power models improving model accuracy. We developed a global data center-level optimization approach for Web Server clusters by looking at the performance and power, simultaneously, of all CPU cores and then deciding what should be the P-state and utilization rate of each core. As part of the implementation of our optimization algorithm, we present an original mechanism to control the CPU utilization rate of each server.

In summary, the contributions of this thesis are as follows:

- A custom-made power measuring infrastructure that can be installed on commodity servers.
- Accurate empirical models for each *P-state* that are able to estimate the power consumed by Web Servers considering all their major parts.
- A Web Server power characterization for their different *P-states*.
- Application of a correlation-based feature selection (CFS) algorithm to prune correlated model parameters.
- k-means clustering to soften non-linear relationship among server measurements and power consumption on linear power models.
- Deployment the *Slack Recovery* algorithm [6], on a real cluster, composed of 25 SeaMicro nodes to minimize power under a minimum performance requirement.
- A novel mechanism to control CPU utilization rates in each server, a key aspect on our power/performance optimization system.

This thesis resulted in the following publications:

- Leonardo Piga, Reinaldo A. Bergamaschi, Rodolfo Azevedo, and Sandro Rigo. Power Measuring Infrastructure for Computing Systems. Technical report, Institute of Computing, University of Campinas, 2011.

- Leonardo Piga, Reinaldo A. Bergamaschi, Felipe Klein, Rodolfo Azevedo, and Sandro Rigo. Empirical Web Server Power Modeling and Characterization. In *IEEE International Symposium on Workload Characterization (IISWC), 2011*, 2011 [50]

- Reinaldo A. Bergamaschi, Leonardo Piga, Sandro Rigo, Rodolfo Azevedo, and Guido Araujo. Data Center Power and Performance Optimization through Global Selection of P-States and Utilization Rates. *Sustainable Computing: Informatics and Systems*, 2012 [6]

We have two papers submitted to Journals:

- Leonardo Piga, Reinaldo A. Bergamaschi, and Sandro Rigo. Empirical and Analytical Approaches for Web Server Power Modeling. In ‘‘`Jornal of Cluster Computing`’’.

- Leonardo Piga, Reinaldo A. Bergamaschi, Mauricio Breternitz, and Sandro Rigo. Adaptive Global Power Optimization for Web Servers. In ‘‘`Jornal of Supercomputing`’’.

In addition, during the development of this thesis, two other side-related papers were published:

- M. Breternitz, K. Lowery, A. Charnoff, P. Kaminski, and L. Piga. Cloud Workload Analysis with SWAT. In *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2012 [12]

- Leonardo Piga, Gabriel F. T. Gomes, Rafael Auler, Bruno Rosa, Sandro Rigo, and Edson Borin. Assessing Computer Performance with SToCS. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE ’13, 2013 [52]

1.2 Organization

This thesis developed power and performance policies to reduce the energy consumption on Web data centers. In order to study this problem, we first needed to measure server power. Hence, we studied alternatives to measure server power and developed a custom-made power measuring infrastructure, presented in Chapter 3. Since the availability of power measuring infrastructure was limited, Chapter 4 presents techniques to estimate Web Server power by using available system level statistics. These models were used on our simulation environment to develop and test the power and performance policy

algorithms. They can also be used by servers that do not have embedded power meters. Finally, Chapter5 describes one of our heuristics on a real cluster environment. To study these topics, we divided this thesis in eleven tasks illustrated in Figure 1.2.

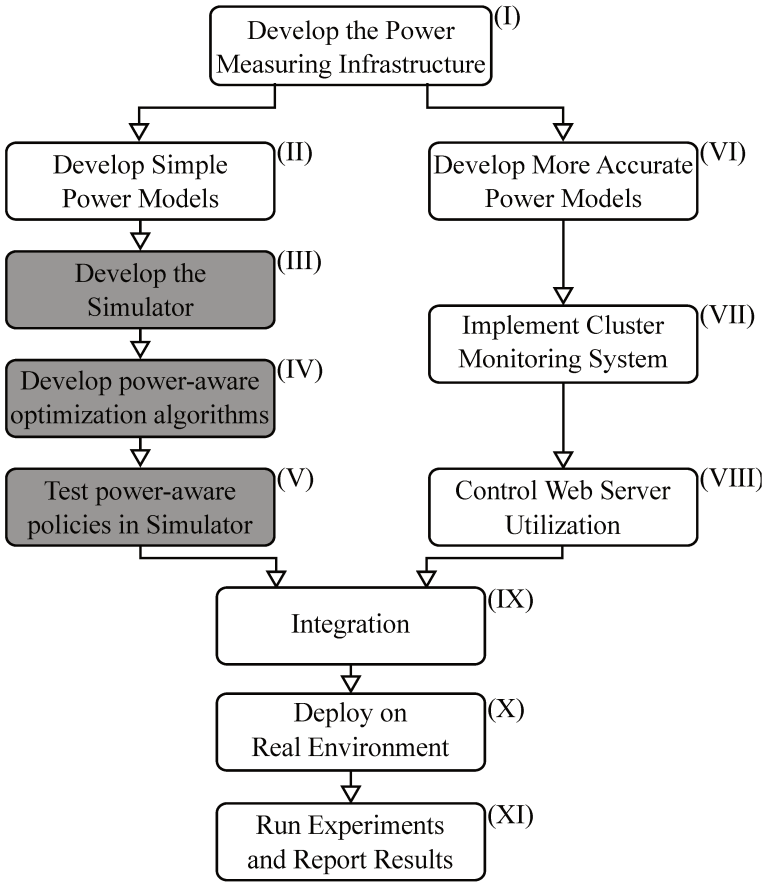


Figure 1.2: Major project task workflow. The rounded white rectangles correspond to tasks developed by the student. The rounded gray rectangles corresponds to tasks performed by the research group.

The first major task was to develop a power and performance measuring infrastructure [(I) in Figure 1.2]. Hence, we were able to correlate power and performance statistics to measure the power consumption of the servers. Chapter 3 describes it. The full description of this infrastructure can be found in Piga et al. [51].

After having developed the power measuring infrastructure, we worked on developing power models. First a simple power model was developed [(II) in Figure 1.2] to be used in the simulator. This power model used only CPU utilization as parameter. We calculated the average CPU power when the CPU was in idle and the average CPU power when the CPU was operating on 100%. Then, the power model was derived by linear interpolation on these two points.

The next three tasks were to develop the data center simulator, develop the power-aware optimization algorithms, and test the power-aware policies in a simulation environment [(III,IV,V) in Figure 1.2]. These tasks were developed jointly with other members of our research group. We published a paper on "Sustainable Computing: Informatics and Systems Journal" describing the simulator and the results [6].

Chapter 4 describes the development of accurate power models [(VI) in Figure 1.2]. These models can be used to estimate the power of the system on-the-fly on clusters that do not have embedded power meter mechanisms. A previous version of this work was published as a short paper in the "IISWC 2011 Conference" [50]. An updated version of this work is under review for the "Jornal of Cluster Computing".

Tasks VII to XI are described in Chapter 5. A version of this work is under review for the "Jornal of Supercomputing". The cluster monitoring system [(VII) in Figure 1.2] is a client-server system that collects performance counters and system level statistics and assigns frequency and utilization rate to the Web Server to implement the "Virtual Power States" (frequency states plus a utilization level) see [6]. The system architecture overview is described in Section 5.2.

To perform the control of the Web Server CPU utilization [(VIII) in Figure 1.2] we developed a controller mechanism based on modern control theory that used a feedback control loop. This mechanism adjusted the weights of HAProxy, an open-source proxy implementation. The description of the utilization controller is found in Section 5.4.

The "Integration" [(IX) in Figure 1.2] task consisted of combining the power-aware load-balancing heuristics with the cluster monitoring system and the mechanism for controling the Web Server utilization.

The next task was to deploy the system on a real environment [(X) in Figure 1.2]. We ran our infrastructure on a SeaMicro SM15k cluster. This cluster virtualizes the I/O and has special fabric to interconnect computing nodes. This results in a high density cluster that is target for cloud computing applications. Finally, we ran the experiments and report the results.

In summary, the remainder of this thesis is organized as follows:

- Chapter 2 presents the related works. They are divided into five categories. We give special attention to the works that presents power models and power optimization under performance requirements.

- Chapter 3 presents our custom-made power measuring infrastructure used to measure power during the power modeling phase of this thesis. It also describes the the Intel Running Average Power Limit (RAPL) interfaces [29, 59] used to measure CPU power on the SeaMicro cluster.

- Chapter 4 describes the Web Server power modeling techniques. The models are developed incrementally starting from a linear-regression-based power model and ending with a model that uses K-means clustering.
- Chapter 5 presents the implementation of the *Slack Recovery* algorithm on a SeaMicro cluster along with a CPU utilization control mechanism.
- Chapter 6 summarizes the results of this thesis and presents possible future works.

Chapter 2

Related Work

The thesis related works can be divided in five major areas. The first area is related to data center provision. The second area describes the studies in Web Server characterization. The third area presents the papers that optimize performance under a given power cap. The fourth area shows the studies that optimize power under performance constraints, which could be either at the microarchitectural level or system level. The fifth area presents the power modeling related works. This thesis focuses on the last two areas as it studies Web Server power modeling techniques that can be used to derive Web Server power models on systems that do not have embedded power meter facilities and implements a power optimization heuristic on a modern Web Server cluster.

2.1 Data Center Provision

There are some works that focused on resource planning. Fan et al. [23] characterized the power consumption of a data center by breaking up the power consumption into components such as computing units and cooling. They developed a model to improve the data center power provision.

Filani et al. [25] focused on increasing the compute density by changing the server provision policy. They used a closed-loop control algorithm to perform power management, by acting on processor states and memory in order to comply with certain policy directives dictated by the upper level software.

2.2 Web Server Characterization

The work described by Bohrer et al. [11] measured and analyzed the power in a single Web Server using a specific processor type and developed a power simulator for it using static

workloads. They did power characterization of Web Servers considering a hypothetical support to DVFS (Dynamic Voltage and Frequency Scaling). They developed a Web Server simulation tool which could predict the power consumption based on Web requests and CPU cycles. The adopted workload was generated from LOG files and static Web content. Such workload was not compatible with modern Web content, which heavily rely on dynamic content for rendering the pages. In this thesis, we also do power and performance characterization of Web Servers; however, our workload has dynamic content. We used SPECweb2009 [64] and the CloudSuite Web Server benchmark, which runs a Web 2.0 web-based social calendar called Olio [24].

2.3 Performance Optimization under a Power Cap

Other works intend to maximize performance given a power cap. Rajamani and Lefurgy [53] investigated the problem of scheduling service requests among servers in a cluster in order to minimize energy consumption known as Power-Aware Energy Distribution (PARD). The authors evaluated the influence of the system-workload context on energy-saving schemes using a simple on/off model for estimating the energy consumption of the cluster. In Chen et al. [18], on/off node optimizations were applied in a multiple-application data center. The goal was to determine how many servers and their operation frequencies should be used for each application. All servers ran the same application at the same frequency.

Kant et al. [33] developed a simple task model based on QoS requirements and presented Willow, a simple adaptative control scheme for energy-adaptive computing (EAC) that considered power and thermal constraints simultaneously. They discussed three scenarios for applying their model: 1) Cluster EAC, where clients submit requests that required significant computation on the cluster side; 2) Client-Server EAC, where observing QoS was an important requirement; and 3) Peer-to-Peer EAC, where devices changed information through a network.

Winter et al. [66] presented scheduling and power management algorithms for heteregeneous many-core architectures. Cochran et al. [19] presented a control techinique to choose CPU voltage and frequency states in an optimal way in order to maximize performance under a power budget. Shen et al. [63] developed an operating system feature that enabled request level power management. Meisner et al. [48] evaluated the effects of switching CPU states to idle states for a short period of time on online data-intensive services (e.g. web search). The problem was more complicated than other Web workloads, since it required fast response time and the data were distributed across the nodes unabling nodes shutdown.

2.4 Power Optimization under Performance Threshold

There are works focused on microarchitectural power optimization. The in-core power management algorithms in Isci et al. [30] dynamically controled the processor parameters (frequency, voltage and fetch throttling) in order to optimize power and/or performance according to on-the-fly workloads. They reported up to 38% of power savings with 17.7% in performance degradation.

A survey of power management techniques developed to explore and optimize the power–performance trade-off in data centers can be found in Bianchini and Rajamony [10].

There are works that tried to reduce the power consumption maintaining a certain performance level. The work in Chase et al. [16] developed an agent-based approach, implemented in the middleware, to turn off servers under low-load conditions while maintaining the expected (or contracted) Service Level Agreements (SLAs). Their approach was based on turning servers on and off and distributing the work among the powered-up servers. They could reduce energy in up to 29% for a Web workload from the 2000s.

In Kusic et al. [38], dynamic resource provisioning was used in a virtualized computing environment to reduce power consumption while maintaining SLA. This work accounts for the switching costs incurred while provisioning (turning on/off) virtual machines and explicitly encoded the corresponding risk in the optimization problem. They reported 26% of power savings.

In Elnozahy et al. [21], five cluster-wide power management policies were evaluated. The authors applied DVFS and node on/off techniques to reduce power consumption during periods of reduced workload. The policies assumed that the workload was balanced among cluster nodes. Policies include: independent voltage scaling, where each node managed its own power consumption; coordinated nodes' voltage scaling actions; turning nodes on/off so the minimum number of servers required by the workload was kept active; and combinations of these techniques. The authors concluded that between 33% and 50% of cluster energy could be saved by applying the combined policy, when compared to a cluster that is not power managed.

In their follow up research [22], the authors used DVFS and request batching (where the servicing of incoming packets was delayed until a specified batching timeout was reached) management mechanisms to propose three policies to reduce energy consumption in Web Servers. They showed that DVFS was better suited for moderately intense workloads, while batching was better for low-intensity workloads. They also proposed a combined policy that reached 17% to 42% energy savings in all workloads, compared to a base model with no optimization. A feedback-driven control framework was used to adjust the policiy parameters.

Bertini et al. [8] presented an optimal linear programming-based solution to the problem of dynamic cluster configuration combined with the use of feedback control theory to control the QoS (Quality of Service) and dynamically select which servers turn on/off or their operating frequencies. Two control theory schemes were compared, single-input single-output (SISO) controller and single-input multiple-output (SIMO) controller. The authors showed that the SISO approach did not scale as an online solution, so they applied a table-based offline solution. On the other hand, the SIMO approach ran with N-independent controllers, at a cost of loss of optimality. They reported 40% in power reduction. Afterwards, the authors revisited the problem of energy consumption minimization with QoS guarantees, but this time focused on an e-commerce application and its control loop, presenting a new metric for QoS [7].

Abbasi et al. [1] proposed TACOMA (two-tier architecture for cooling-computing energy managements), a two-tier Internet data center scheme. The first tier adjusted the number of active servers; the second tier predicted the workload arrival rate. The algorithm evaluation was based on web traces and they reported energy savings of up to 40% considering compute and cooling power.

Several of these works reported significant savings (30% or more). Their contributions were undoubtedly relevant for the CPU architectures and data center power management techniques at the time. However, the state-of-the-art today is very different. CPU power (both active and idle) in modern architectures (e.g., Intel Ivy Bridge) is considerably lower than it was 5 years ago; thus, reducing significantly the gains that simple On/Off or DVFS techniques can achieve. In addition, the power/performance management techniques in today's data centers (e.g., Linux *ondemand* governor [14]) are much more effective than the baseline data center approaches used in these previous works. In this thesis, we investigate power and performance trade-offs for Web Servers on a state-of-the-art, high-density, power-efficient SeaMicro SM15k cluster by AMD and we were able to reduce power consumption in up to 16% even in such a power efficient cluster.

2.5 Power Modeling

Applying power optimization requires, maintaining an average power budget, asserting peak-power constraints, or optimizing performance under a power cap require monitoring the power consumption. Previous works developed linear power models based on system performance measurements, which have shown suitable for CPU-bound workloads [5,9,20, 31,32]. They have used Performance Monitoring Counters (PMC) as proxies to estimate CPU power.

Bellosa [5] showed that CPU power correlates to floating point operations, L2 cache references, and memory references. His work was one of the first to propose the use of

PMCs to create an energy-aware scheduler.

Joseph et al. [32] introduced a model capable of estimating the power consumption for the processor and its sub-components by using PMCs and some heuristics based on capacitance models. Though accurate, such information might not be available for all processors. Isci et al. [31] introduced a model for the Pentium IV processor that did not rely upon circuit-level information. However, their model required more then 15 PMCs, even though such a large number of counters is not usually available at the same time on most modern processors.

Contreras and Martonosi [20] used PMCs as inputs to a linear model for an Intel mobile processor at three voltage and frequency levels. Their model was validated with benchmarks representing embedded systems. Bertran et al. [9] used PMCs to build power models for contemporary Intel multi-core processors considering all voltage and frequency levels available.

Chen et al. [17] presented performance and power models in a multi-programmed multi-core enviroment by addressing the problem of time sharing. The performance model was based on the cache access pattern. They proposed a power model using neural network and another using linear regression on CPU performance counters. The reported prediction error was 3.2% for the former and 3.8% for the later. Though accurate, they only evaluated their models for CPU-intensive benchmarks (i.e. SPECcpu 2000).

Lewis et al. [41] also proposed power models using PMCs to enable dynamic control of thermal footprint. They modeled the computer using system of deterministics differential equation whose solution was estimated via time-series approximation. Since their model used a time series, it needed to keep previous samples to estimate future values. They reported prediction error between 1.6% and 3.3% for both systems that they evaluated.

In our best Web Server Power model, the prediction error is 1.92% for the Intel i7 server and 1.46% for the AMD Opteron. Therefore, our proposed technique is equivalent in terms of accuracy to the other alternatives [17, 41].

The works listed so far focused mainly on CPU benchmarks, whose applications fully utilize the processor. Because Web Servers typically do not run CPU-bound workloads, due to the high number of I/O operations performed by such servers, the shared resources are more likely to be used more often resulting in non-linear effects between power and system measurements. Koller et al. [35] tried to deal with this issue by aggregating application parameters, such as throughput, to the power model. They reported improvements of up to 10 times over a simple power model that only uses CPU utilization.

Rivoire et al. [56] created JouleSort a benchmark that characterized energy efficiency to evaluate systems. It was a full-system benchmark that specified a workload, a metric to compare systems, and rules for running it. They continued the research on metrics and models and compared several high-level full-system power models [57]. They used a

variety of workloads and architectures and also observed non-linear effects among CPU power and performance measurements due to bottlenecks on shared resources.

Another component that should be taken into account when dealing with Web Servers is the hard disk drive. Carrera and Bianchini [15] proposed the usage of disks with multiple rotation speeds to reduce the energy consumption in data centers. The authors used power values from data-sheets instead of actually measuring power. They showed that SCSI hard disks could account for up to 24% of the overall energy consumption of a server. Besides, if a server was built with a higher number of disks, this fraction could increase to 77%. In our experiments with Web Servers, we observed that hard disk power account for up to 20% but did not exhibit a wide variation allowing this power component to be modeled as a constant value.

Zedlewski et al. [67] developed a tool called *Dempsey*, which simulated disk operations in order to estimate the power consumption for a given workload. Their model was built with real power measurements. However, the chosen modeling parameters were based upon disk information that might not be available for all classes of hard disks.

2.6 What is different in this work

This section describes the contributions of this work to the state-of-the-art in in Web Server power modeling techniques and in the Web Server Power and Performance optimization field.

2.6.1 Web Server Power Modeling

Most contemporary high-end processors feature sensors for monitoring energy consumption [54]; however, in commodity processors, which are prevalent in Internet-based data centers, this is typically not the case. Nevertheless, these processors usually feature event counters that can be used to estimate power consumption. For example, the second-generation Intel Core microarchitecture uses performance events as proxies of power consumption [60]. The technique is based on reading hundreds of internal performance counters and on applying activity energy costs to each event to estimate power. Previous studies have used these probes to indirectly estimate power consumption [5, 9, 20, 31, 32]. Their usual approach is to derive linear power models based on the usage numbers collected for the processor sub-components (e.g. caches and branch predictor).

This work advances the state-of-the-art in this area by presenting power models for two systems considering all their major parts (i.e. processors, disks, network, memory, and other motherboard components) and all its software stack (when running as Web Servers) for their different CPU core voltage and frequency states (also known as *P-states*). These

models can be used for commodity systems for on-the-fly power-saving algorithms, and among other applications, they can also be used by simulators which evaluates the power behavior of workloads. Our models have been used in a data center simulator to guide the implementation of power/performance optimization algorithms through voltage and frequency state assignment [6].

The power models are developed by using different methods, such as, linear regression, cluster analysis, and machine learning techniques having hardware events (e.g. number of instructions, unhalted cycles, cache misses) and system level measurements (e.g. page-faults, number of context switches) as proxies for power. However, linear power models based on system measurements and performance counters exhibit issues that need to be investigated: excess of parameters and non-linear relation among power and the associated factors (as is the case for I/O bound workloads).

The fewer the number of parameters of a model the faster the power estimation; therefore, we prune model parameters by using a correlation-based feature selection (CFS) [27] algorithm for choosing a subset of them that is most correlated to the power measurements. This approach has reduced the number of parameters preserving accuracy and precision when compared to a model using all the parameters.

To allow the use of power models based on linear regression on these types of workloads, we use k-means clustering to group up the performance measurements. Linear regression is applied on each cluster to dismiss the non-linear relationship among server measurements and power consumption improving model precision when compared to the other models. From our knowledge, this is the first work that applies CFS and k-means clustering to improve linear regression-based power models for computing systems.

2.6.2 Web Server Power and Performance Optimization

Our approach differs from previous works as it is the first to combine the notion of utilization rate to the P/C power states available in modern processors to define what we called Virtual Power States. This enabled us to develop a global optimization approach by looking at the performance and power of all cores in a data center simultaneously, in order to achieve a global data center-level optimization of power and performance.

This work is part of a project that presents a framework for modeling and simulating the power and performance behavior of complete data centers composed hierarchically of racks, nodes (or blades), CPUs and cores. Based on the concept of Virtual Power States, two algorithms for selecting the states in each core in order to optimize the power and/or performance of the data center were developed. The first algorithm was based on Integer Linear Programming (ILP) and was able to find optimal solutions for a limited-size data center (due to runtime limitations). The second one was a *Slack Recovery* heuristic

algorithm which can find near optimal solutions for all cases (when compared to the ILP results).

In this work, we developed power optimization techniques that can show significant improvements over best-of-breed, production-level dense data centers, using the latest CPU architectures. We took the concepts of Virtual Power States (VPS) and the *Slack Recovery* algorithm and applied them to a modern cluster architecture composed of 25 nodes of SeaMicro servers [3], a state-of-the-art server architecture by AMD.

Several practical implementation aspects could be overlooked in a simulation environment. In this work, we implemented everything in the actual cluster; thus, demonstrating not only that the approach is scalable, but can achieve significant savings even considering the extremely power-efficient baseline starting point, such as the SeaMicro cluster using Intel's Xeon processors (Ivy Bridge architecture), running Linux *ondemand* governor power management.

As part of the power optimization algorithms, we developed a novel mechanism to control the CPU utilization rate of each server, a key aspect of our approach.

To the best of our knowledge, this is the first work to implement and evaluate power/performance optimization algorithms in such high-density cluster architectures (e.g., the SeaMicro SM15k cluster). We believe the results are general and applicable to a wide range of data center architectures.

Chapter 3

Power Measuring Infrastructure

There are several methods to measure the power delivered to a given component. Some approaches use expensive intelligent power supplies or motherboards that have embedded power meters [39]. In other works [11,67], multimeters connected in series with the circuit are used for current measurements. This approach might add noise into the circuit and not be suitable for high sampling rates. This Chapter presents our power measuring infrastructure used to develop the Web Server power models described in Chapter 4. It also presents details about the Running Average Power Limit (RAPL) interfaces [29, 59] used to measure the CPU power on SeaMicro SM15k cluster.

3.1 Custom-made board

To measure the power consumption of a server, we designed a custom-made measuring device similar to the infrastructure described in other works [45, 51] as shown in Figure 3.1. To measure the power consumption of a server, we connect sensors in series with the positive power lines that measures the current that flow across the wires. In a regular server that uses the ATX interface, the memory banks, the CPU, and the chipset are powered by the 3.3V, 5.0V and 12.0V rails. The hard-disk has an exclusive power connector. According to the ATX specification, 11 wires need to be monitor, plus two wires that feed the CPU, and two that feed the hard disk. Thus, 15 current transducers (LTS 25-NP [40]) in series with the power lines are used so as to convert current to measurable voltage. Given that the voltages of these wires are known upfront, by measuring the currents that flow through them, it is possible to calculate the power consumption, which is the product of voltage and current.

We used the LTS sensors because they present high accuracy ($\pm 0.2\%$) and linearity of less than 0.1%. The LTS 25-NP operation mode is described in the graph of Figure 3.2. For this sensor, I_{Pmax} is 80A and I_{PN} depends on the connection of the sensor sideline

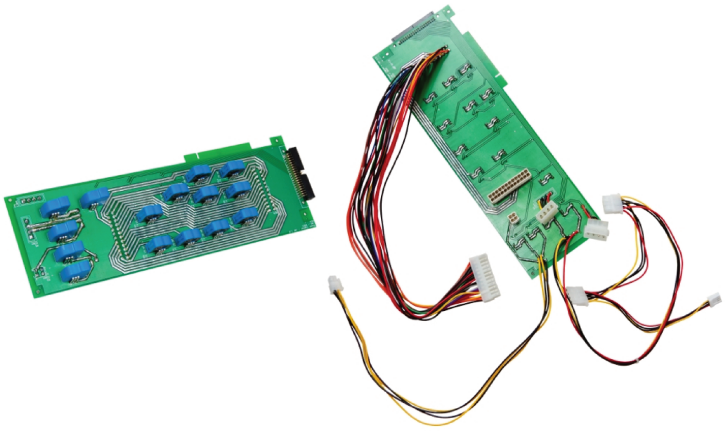


Figure 3.1: Picture of the Board: Front-side and Back-side

pins. There are three modes: in the first mode I_{PN} is 25A; in the second mode I_{PN} is 12A; and in the third I_{PN} is 8A. For further details on how the pins should be connected refer to the LTS 25-NP datasheet [40].

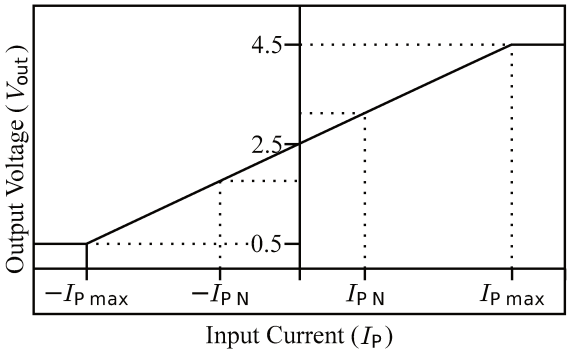


Figure 3.2: LTS 25NP: V-I Characteristic Curve

In our power measuring infrastructure, the sensors that measure the current operate on the third mode, since the maximum current that the connectors support is 6A. On the other hand, the sensors that measure hard-drive power operate on the second mode. The operating mode makes more precise the measurements in the region of $-I_{PN}$ and $+I_{PN}$, but does not limit the current to this interval. Moreover, values out of this interval are not observed during the experiments.

Figure 3.3 shows the complete infrastructure, our board is installed into the server (plugged into a PCI slot). ATX, HD, and CPU plugs from the power supply are connected

to the board's inputs. The output of the transducers are attached to a 16-bit data acquisition system (National Instruments NI USB–6212 [28]) that is capable of acquiring 400 k samples per second across all channels. Since 15 channels need to be sampled, the actual sampling rate is 25 k per second. Each ATX positive wire is connected in series to a transducer. Given that the voltages of these wires are known upfront, by measuring the current values that flow through them, it is possible to calculate power consumption, which is the product of voltage and current. Eventually, the data are read and stored by a monitoring computer from the acquisition device. Piga et al. [51] fully describes our power measurement infrastructure shown in Figure 3.3.

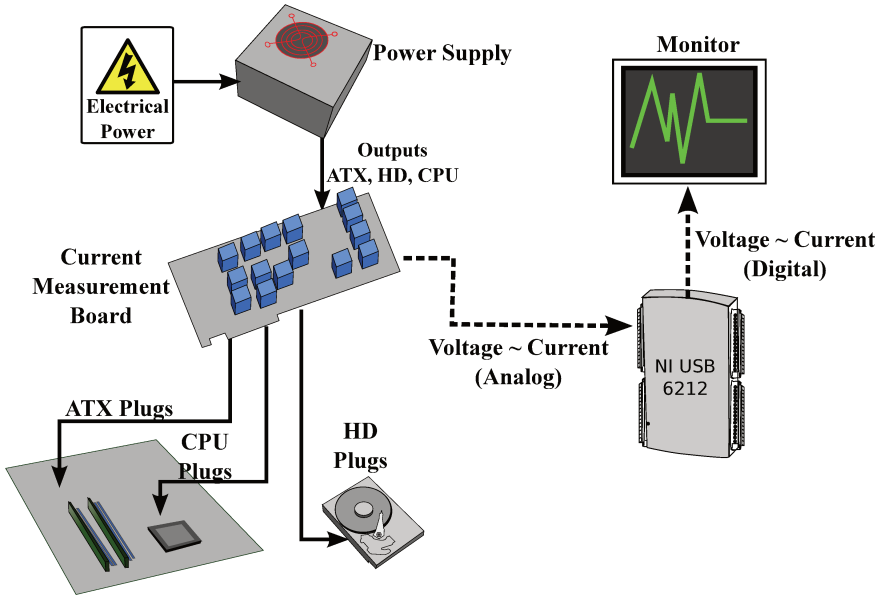


Figure 3.3: Custom-made power measuring infrastructure designed to collect power on commodity computer systems

3.2 Calibration process

The next step consists on calibrating the sensors in order to increase the accuracy of the measurements, it is done as following: insert into the sensors known values of currents; annotate the sensor output values; after annotating the values, fit 15 curves (one for each sensor). For this process, we used a precise current source (Minipa MPL 3303M), which is able to generate currents from 0A to 3A.

The curves are on the form $i_j = a_j * AI_j - b_j$. Table 3.1 shows the parameter for each sensor. The sensors support current values that may be in the intervals $[-I_{Pmax}, -I_{PN}]$

and $[I_{PN}, I_{Pmax}]$, however these values are not observed along a large set of experiments. Hence, the equations are for the interval $[-I_{PN}, I_{PN}]$.

j	a_j	b_j	j	a_j	b_j
0	20.151	50.130	8	20.187	50.209
1	13.399	33.390	9	13.341	33.175
2	13.340	33.259	10	13.264	33.015
3	13.369	33.305	11	13.320	33.123
4	13.317	33.101	12	13.396	33.293
5	13.329	33.165	13	13.302	33.073
6	13.493	33.586	14	13.435	33.476
7	13.456	33.553			

Table 3.1: Parameters of the Sensors

3.3 Monitoring software

This section presents the application software that runs on a remote computer identified as "Monitor" in Figure 3.3. The application that interacts with the data acquisition system works based on TCP/IP messages and is composed of seven main steps. Figure 3.4 illustrates the control flow. The first step is the initialization which opens a TCP/IP socket and listens for connections. Only one client is allowed at a time. The second step waits for `Start` messages. The third step is responsible for recognizing the measuring type mark, which determines the action taken when a sample is read. Then, the program flow is split up into two parts, one responsible for reading and processing the samples and other that waits for `Stop` messages. After receiving a *Stop* message, the reading process halts, the two parts are joint, and the application waits for the next message. If the next message is a `Start` the process restarts from the third step. If it is a `Quit` message, the application executes the termination steps and halts.

A starting message is followed by a measuring type mark. It determines how the samples should be stored. Some experiments need only a summarized result such as mean, minimum value, maximum value, others may need to store all samples. Currently, the application has four ways of processing the samples:

1. Read the samples, convert the sampled current values into power values, group them into CPU, HD, and miscellaneous components power, and write them into a binary file.

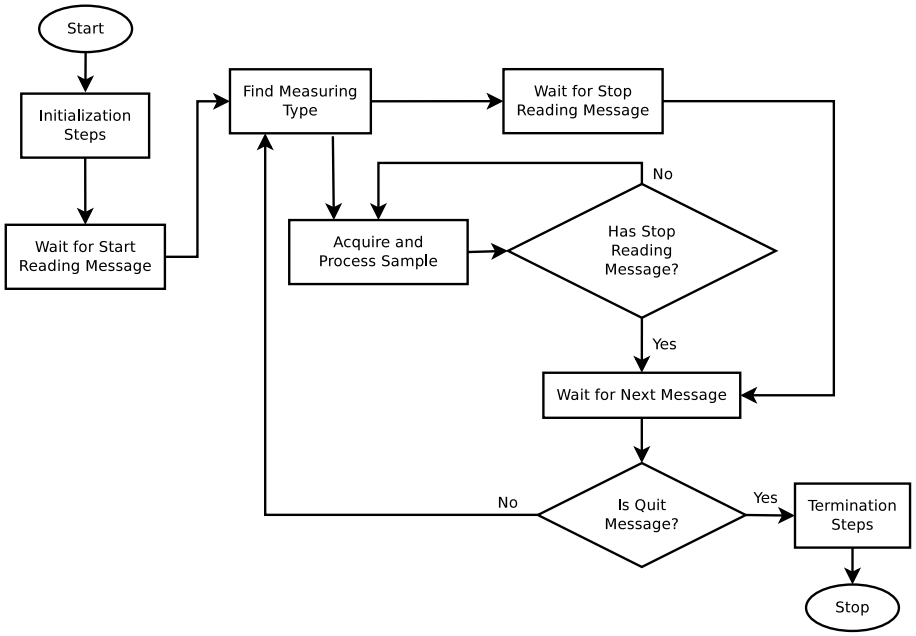


Figure 3.4: Application Control Flow

2. Read the samples, convert the sampled current values into power values, **group them into CPU, HD, and miscellaneous components power**, compute the average, checks for the minimum and the maximum values. After receiving a `Stop` message, these statistics are sent to the client over TCP/IP messages.

3. Read the samples, convert the sampled current values into power values, **for each sensor**, compute the average, checks for the minimum and the maximum values. After receiving a `Stop` message, these statistics are sent to the client over TCP/IP messages.

4. Read the samples, convert the sampled current values into power values, store the latter into a big buffer. After receiving a `Stop` message, the buffer is sent to the client over TCP/IP.

These operation modes are sufficient for most of the experiments that we need. For the last mode, we developed a converter that enables the signals to be displayed in a waveform. The program we used to do this operation is the GTKWave [26]. Figure 3.5 shows a period of time when the computer is in idle mode. If we observe the CPU signals, we can see that it displays bursts of power in intervals of 4ms. This corresponds to the system tick clock cycle which is 250Hz. This mode is interesting for using when seeking patterns along the time line.

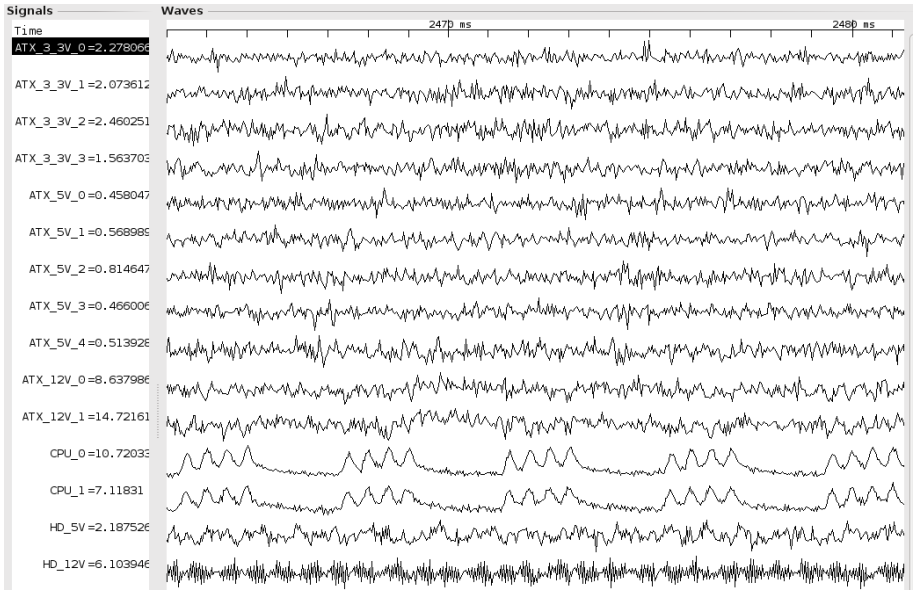


Figure 3.5: Waveform when the computer is in idle

3.4 Intel Running Average Power Limit Interfaces

Another approach used for measuring CPU power is the embedded power estimator counters available on modern architecture such as Intel Sandy Bridge and AMD Family 15h processors. Intel made available the *Running Average Power Limit* (RAPL) interfaces [29, 59] after the second-generation Intel Core microarchitecture. The technique is based on reading hundreds of internal performance counters and on applying activity energy costs to each event to estimate power with an accuracy of around of 10% [59].

The interface makes available special processor registers that estimates the power and energy consumption of CPU-level components such as CPU and memory controllers. It also can be used to limit the power consumption during a period of time. This could be explored in the implementation of maximize performance under a power cap of the *Slack Recovery* algorithm. However, in this thesis, we are not investigating this mode.

Chapter 5 presents how we used RAPL interfaces on SeaMicro cluster to optimize Web Server power. We used the RAPL interfaces on the SeaMicro SM15k cluster instead of our custom-made board experiments for four reasons: The first is availability, the cluster CPUs have the RAPL interface. The second is the geographic barrier, the servers were located in USA and we did not have physical access to it. The third reason is because the board interfaces would not be compatible with the SeaMicro fabric. The last reason is the fact that on SM15k cluster, the I/O components are virtualized and used more efficiently because the HD and network power component is divided among the nodes. Hence, the CPU power component dominates the system power.

The counters can only be accessed by kernel code (ring-0). Thus, we developed a Linux Kernel module that reads the counters and make the values available in the sys filesystem, a specicial directory that carries information about device drivers and hardware. This approach facilitates the readings of the counters as the counters can be accessed as regular system files. For our experiments on SM15k cluster, we measure the whole CPU package power.

Chapter 4

Web Server Power Models

This Chapter presents the techniques used to develop the Web Server power models. It describes the experimental methodology used to collect power and performance measurements, the characterization models and the experimental results. The models are developed incrementally by starting with a linear-regression-based model containing data points from SPECweb 2009 benchmark and SPECint 2006 benchmark for all P-states and ending with a model that softens non-linear relation between power and performance statistics.

4.1 Experimental Methodology for Web Server Power Modeling

This section introduces the power characterization methodology to measure the power consumption of commodity system computers. The workloads are SPECint2006 and SPECweb2009. Our main target is Web Server models, however, we use SPECint2006 to stimulate higher activity level on the CPU. We develop models for two different commodity systems: (1) A server with Intel i7 860 processor, 4 GB of memory, and an Western Digital serial ATA hard disk of 500 GB and 7200 rpm running Ubuntu 9.10 x86. (2) A server with an AMD Opteron 6168 processor containing 12 cores, 16 GB of memory, and a Seagate serial ATA hard disk of 1 TB and 7200 rpm with Ubuntu 10.04 x64 as operating system. All servers run the Apache 2.2.16 Web Server configured to use threads and PHP 5.3.3 for serving SPECweb2009 [64].

The experimental setup environment is depicted in Figure 4.1. SPECweb2009 [64] *(i)* is a benchmark suite that generates server requests based on real Web applications, characterizing by different workloads. In order to synchronize the power numbers collected from our measuring device with the performance statistics from the Web Server, we im-

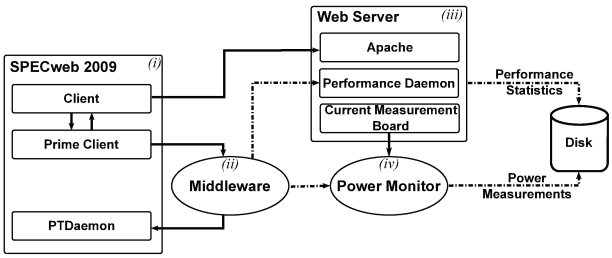


Figure 4.1: Diagram for the power characterization methodology using the SPECweb 2009 benchmark

plemented a middleware *(ii)* layer. When a given workload execution is completed, the resulting power values and performance statistics are stored on disk for model generation. This information is then used to build power predictors by using conventional regression analysis methods (e.g., least mean squares). This data-set is split up randomly into training set (30 percent of the points) for building the models and testing set (70 percent of the points) for model validation. By comparing the measurements of the testing set to the estimates from our model, its accuracy can be determined.

4.1.1 SPECweb2009 Benchmark

This section gives an overview of SPECweb2009. The benchmark was constructed based on real Internet services' LOGs from companies in Texas. It is composed of 3 different Web applications [64].

The first is a typical Internet banking application where all requests are performed through the SSL (Secure Sockets Layer) protocol. It has 95.06% of its requests done by the GET method and 4.93% by the POST method. 30% and 70% of the requests use the protocols HTTP 1.0 and HTTP 1.1, respectively. The files are divided into four classes according to their sizes: 35%: (100 bytes to 1kB); 50%: (1kB to 10kB); 15%: (10kB to 100kB); and 1%: (100kB to 1MB).

The second is an ecommerce application for selling computer systems, allowing users to browse, search and purchase products. This application includes both SSL and non-SSL requests. It is formed of fixed small size files extending from 43 bytes to 22.08kB. The image files are randomly generated and they can vary from 3.5kB to 400kB.

The third is an IT support system which allows users to browse a listing of available products and filter available downloads for a given criteria. This application aims at testing the server performance for the download of large files ranging from 100kB to 36MB. Downloads are done based on Web requests to the server and the files are never directly referenced. For all these three applications, the benchmark alternates between periods of inactivity that simulates user thinking time and periods when the users are

making requests.

SPECweb2009's architecture contains four major components, as shown in Figure 4.1. The first component, represented by the box named *Client* in *(i)*, is responsible for simulating application clients. It sends HTTP requests to the server and receives server responses. The second component is the Web Server and its related software [box *(iii)* in the figure], which is described in Section 4.1.4. The third component is PTDaemon, a background application used for collecting server power statistics. Finally, the last component is the Prime Client which is responsible for controlling workload execution (initializing processes, managing clients, collecting results). In our experiments we hosted the Client and Prime Client components on the same computer.

The following strategy was adopted for the execution of SPECweb2009. Seven computers were used as clients simulating 500 and 2000 real clients so as to emulate different server loads. This will impose different activity levels on the server, directly impacting its CPU load and network traffic. To synchronize power and performance measurements, a middleware layer was implemented [*(ii)* in Figure 4.1]. It intercepts controlling messages sent/received to/from PTDaemon [64].

4.1.2 Middleware

In order to integrate the SPECweb2009 software with the experimental measuring device, we implemented a software layer that handles the communication between the Prime Client and PTDaemon. Our middleware monitor TCP/IP messages (especially `go` and `stop`) sent between these components so as to synchronize the power measurements, obtained with our device, with the performance statistics from the Web Server.

When the Prime Client enters the run stage, it sends a `go` message and the middleware intercepts it. The `go` message is then forwarded to PTDaemon which, in turn, sends a `start` message to the Web Server daemon responsible for collecting performance statistics. Another message is sent to the power measuring device to start measuring the power consumed by the server. All these three messages are sent in parallel.

During this phase, the middleware keeps the synchronization between the power measurements and the performance statistics by sending `sync` messages to the measurement device and to the performance daemon running on the Web Server on a second-by-second basis.

When the Prime Client finishes the run stage, it sends a `stop` message to PTDaemon. In a similar fashion, the middleware intercepts this message and forwards stopping messages to both the performance daemon and the power measurement device.

4.1.3 Server Performance Measurements

In this work, we developed models for three different commodity systems. The first is a server with an Intel Core 2 Quad Q6600 processor with 8 GB of memory and a Seagate serial ATA hard disk of 500 GB and 7200 rpm. The operating system is the CentOS Linux 4.5 running Linux kernel version 2.6.31. The second is a server with Intel i7 860 processor, 4 GB of memory and a Western Digital serial ATA hard disk of 500 GB and 7200 rpm. The operating system is Ubuntu 9.10 x86. The third computer is a server with an AMD Opteron 6168 processor containing 12 cores, 16 GB of memory and a Seagate serial ATA hard disk of 1 TB and 7200 rpm with operating system Ubuntu 10.04 x64. All servers run the Apache 2.2.16 Web Server configured to use threads and PHP 5.3.3.

We collected statistics for the usage of CPU, disk, and OS in our experiments. The process of gathering these values does not have a significant impact during power measurement [9, 31, 32].

Hardware events and operating system measurements representing higher level system activity are collected to be used as proxies for power. Operating system measurements are important especially for the Web Server application where many processes of the same type are running in parallel. `Perf` [55] utility is responsible for assessing these events. The models correlate the collected performance rates (events per second) to the power measurements. Section 5.5 explains how these metrics affect the power models. The metrics are as follows:

Instructions retired per second: The CPI (Cycles Per Instructions) value or the equivalent BIPS (Billions of Instructions per Second) are directly correlated to CPU activity level making them important parameters for power models.

Unhalted Cycles per second: This also represents CPU activity level. Higher values are expected when more functional units are working; hence, affecting CPU power consumption.

Last level (L3) cache references per second: Under different *P-states* (i.e. different frequencies) and different workloads, the last-level cache references per unit of time can change significantly, making it an important parameter in power models.

Last level (L3) cache misses per second: When there is a processor cache miss, its pipeline stalls affecting power consumption. This parameter also reflects the demand for memory resources since a miss in the last level cache requires an access to the main memory.

Page faults per unit of time: This metric captures the demand for memory resources with low level of temporal locality, reducing the activity level in the node.

Context switches per unit of time: This metric is used to capture the activity level of the operating system.

CPU migration per unit of time: This counter increases every time a process

changes CPUs. When a process switches CPU, the instruction code cache is flushed, invalidated, and reloaded on the new CPU decreasing the system activity level.

CPU Load: The time share that the CPU spends executing useful processes, that is, $(1.0 - t_{Idle}) \cdot 100\%$, where t_{Idle} is the share of time that the idle process is scheduled. The idle process is an infinite loop of halt instructions that changes the core to the HALT state when scheduled by the Operating System.

The disk parameters are collected by reading Linux device block statistics [43]. Table 4.1 presents the disk events available on the Linux system[1].

Table 4.1: Disk Activity Parameters

number of reads	writes completed	time reading	time writing
reads merged	writes merged	I/O in progress	time doing I/O
sectors read	sectors written	time in queue	

In order to synchronize the power numbers collected from our power measuring device with the performance measurements, we simultaneously sample power and performance on fixed rates (1 second in our experiments). When a given workload execution is completed, the monitoring software stores the resulting power values and performance measurements on disk for model generation. Collecting system-level statistics (such as number of page-faults and context switches) requires only small amount of processing because our sampling rate is one second. Moreover, measuring performance counters requires executing only one instruction to start sampling (`wrmsr`) and other to read the register values (`rdpmc`) [29]. We measured the overhead of collecting performance statistics while measuring power and observed that they are negligible corroborating to previous works [9, 31, 32].

4.1.4 Server Power Measurements

We used the custom-made power measuring infrastructure described in Chapter 3 to collect the power values of the servers. This infrastructure uses current transducers, a 16-bit data acquisition system that is able to read the sensors' output at 25 k samples per second. The board is installed in series with the server power supply and a monitoring software stores the power data.

4.2 Characterization Model

Our models are derived using regression techniques on experimental data using system-level measurements and performance counters as proxies for estimating the power con-

[1] reads/writes merged count the frequency that two 4 kB operations become one 8 kB operation

sumption of the system for each CPU core frequency and voltage state. The following components contribute to the system power: CPU power; chipset, video board, network device, memories, and fans (miscellaneous components); and hard disk power. The total power of the system is computed by simply adding up the estimated power obtained for each component as shown by Equation 4.1. Further discussions about the model accuracies and precisions are done in Section 5.5.

$$P_{Total} = P_{CPU} + P_{disk} + P_{miscellaneous} \tag{4.1}$$

Modern processors support *Dynamic Voltage and Frequency Scaling* (DVFS), which can be exploited to optimize power and performance. Each voltage and frequency operating point represents a so-called power-saving state of the processor. The ACPI [2] is the operating system interface to these power-saving states, usually called *C-states* and *P-states*. C_0 is the active state and $C_1 \ldots C_{n-1}$ are the idle states. The deeper the state the higher the savings, at the cost of increasing time penalty for returning to the active state. When in idle states, the processor normally turns off some of its internal components. In C_0, it is possible to trade-off power consumption for performance by setting the processor to performance states (*P-states*) in accordance with the workload being executed.

On Linux, the *P-states* and their changing policy can be controlled by using the `sysfs` utility, which provides information about devices and drivers from the kernel space to the user space and interfaces to toggle them. There are operating system's utilities such as "On-demand Linux Governor" which can control the *P-state* transitions automatically based on the workload. There is also the "Userspace Linux Governor" utility which allows users to control the *P-states* of the CPU directly. Our models are developed in the context of the latter and they will be used as part of a global power and performance optimization policy [6]. The policy will choose the *P-States* for all CPU nodes and set them for each time-window.

P-states represent different operating points of frequency and voltage with different power and performance characteristics. The Intel i7 860 processor has 14 power states operating from 1.2 GHz to 2.8 GHz. The AMD Opteron processor has 5 power states operating from 800 MHz to 1.9 GHz. We derived models for CPU (i.e. CPU cores, L1, L2 cache, and L3 cache) by measuring power and performance on each CPU *P-state*. Thus, each benchmark run is repeated by setting all cores of the CPU to the same *P-state*. This is not a limitation of the work, as the same methodology described in this paper can be used on a core by core basis.

To develop the models, we run SPECweb2009 and SPECint2006 according to the following strategy. For SPECweb2009, seven computers run as clients simulating 500 and 2000 real clients so as to emulate different server loads. This imposes different activity levels on the server directly impacting its CPU load and network traffic. Since

SPECweb2009 benchmark is not able to exercise the full CPU capability due to its I/O boundedness characteristic, SPECint2006 is also executed to stimulate higher CPU activity levels. An instance of each SPECint2006 program runs in parallel on each core to stress all CPU cores. Therefore, by combining CPU-bound (SPECint2006) and I/O bound (SPECweb2009) data-sets, the complete spectrum of CPU utilization is covered.

To illustrate the differences on the CPU Load on each benchmark we use box plots [47]. A box plot is a convenient way of displaying data without assuming any distribution. Figure 4.2 shows an example to illustrate a box plot. The bottom and top of the box are the 25^{th} and 75^{th} percentiles values ranging from 4.9 to 6.5, the bar inside the box is the median (i.e. 50^{th} percentile), 5.7 in the Figure. The distance from 25^{th} percentile and 75^{th} percentile is know as interquartile range (IQR). The straight line below the box represents the lowest measurement still within 1.5 IQR of the 25^{th} percentile (i.e. lower quartile), which is 2.5 in the graph, and the line above the box represents the highest measurement still within 1.5 IQR of the 75^{th} percentile (i.e. upper quartile), which is 9.9. Finally, the points represent outliers.

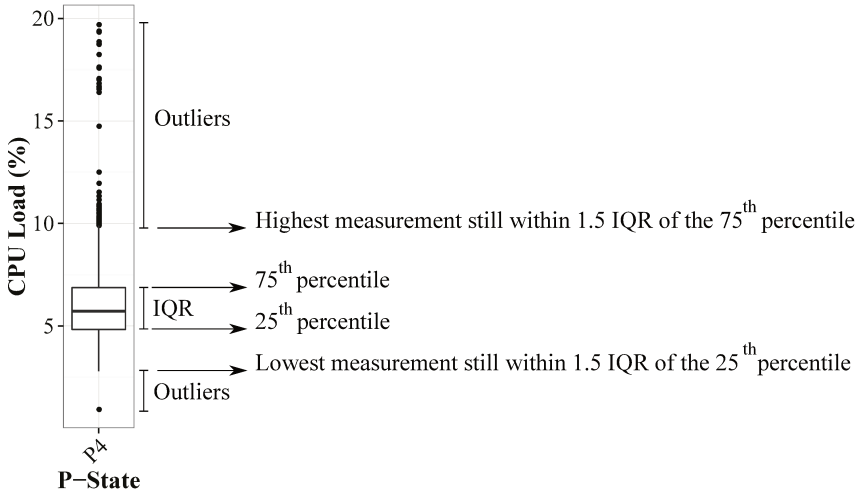


Figure 4.2: Example of box plot for CPU load for SPECweb2009 with 2000 users on the Intel i7 at P4 State.

Figure 4.3 presents box plots for SPECint2006 and for SPECweb2009 (running with 500 and 2000 users) for all *P-states* on the Intel i7 server. The plot shows that SPECweb2009 is not able to achieve CPU Load values higher than 80%. On the other hand, CPU Load for SPECint2006 is concentrated from 60% to 100% with median around 98%. Therefore, by combining both benchmarks we can cover all CPU Load spectrum.

Considering the SPECweb2009 benchmark in Figure 4.3, the outliers for the CPU-utilization are related to the nature of the Web Server application. For example, a Deposit

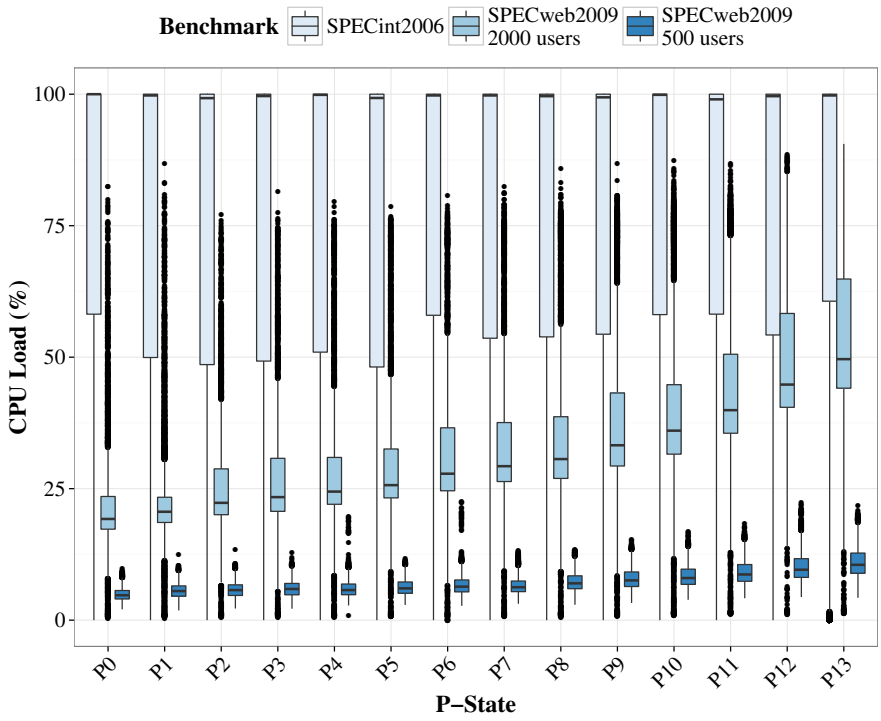


Figure 4.3: Box plot for CPU load for SPECint2006, SPECweb2009 with 500 users, and SPECweb2009 with 2000 on each i7 *P-State*. Each benchmark stresses different ranges of the CPU Load

operation is different from a Login operation, resulting in different CPU usages. Moreover, certain operations are more frequent depending on the operation mix, resulting on the outliers.

Next, we analyze the power variation on CPU, disk, and miscellaneous components. Figure 4.4 shows the power variation in the box plots. The CPU is the critical component of total power since it has the most contribution to total power and is responsible for the most power variation; consequently, it requires more detailed power models.

The work described by Zedlewski et al. [67] presents models using disk time reading and disk time writing to estimate disk power. Following this idea, we developed disk power models based on the disk usage parameters that estimate the serial ATA disk power consumption. However, these models displayed low coefficient of determination (R^2), low precision and accuracy; hence, this approach was discarded. We attribute this lack of correlation to the operating system that delays the update of disk statistic counters impacting on synchronization between disk statistics events and its power measuring.

However, it is possible to see that disk power varies very little. In addition, the standard deviations are less than 5% of the mean values for this power component. Figure 4.4 shows that the disk power is concentrated between 8.0 W and 9.0 W for the Opteron-based

system and 5.0 W to 6.0 W for the Intel i7-based system. The disk power consumption is responsible for only 10% to 12% of the total power; thus, we model disk power as a constant value. We grouped all other computer components such as chipset, video board, network device, memories, and fans in a power component called miscellaneous components. Following the same argument used for disk, miscellaneous components power has small contribution to total power and small variance; hence, we also model this component as a constant value equal to the average power of all points.

On the AMD Opteron Server, the CPU contribution to total power is even higher compared to disk and miscellaneous component power. Therefore, considering constant power models for these components has even smaller impact on total model accuracy when compared to the Intel i7 Server.

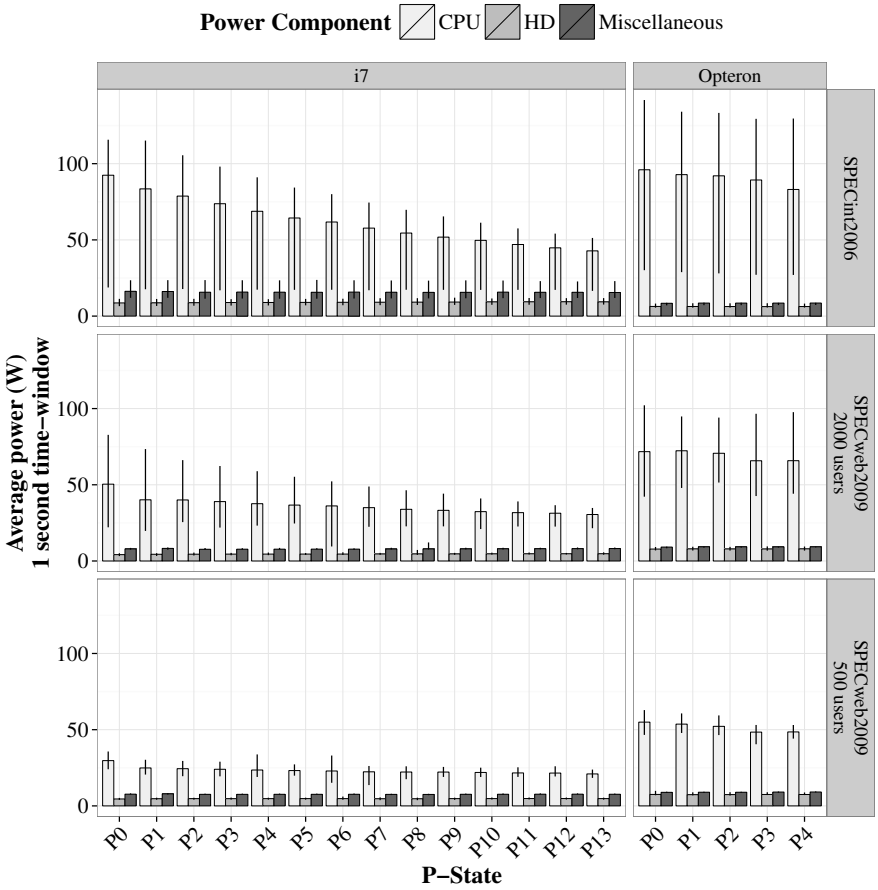


Figure 4.4: Average device power measurements for SPECint2006, SPECweb2009 with 500 users, and SPECweb2009 with 2000 on each *P-State* for AMD Opteron and Intel i7. The lines represents the minimum and the maximum device power measurement. CPU presents the most variation and has the most contribution for total power.

4.3 Experimental Results

This section presents the results for our experiments and the evaluation of our power models. We split up the data-set into training set (50 percent of the points) for building the models and testing set (remaining 50 percent of the points) for model validation. We build the power models incrementally starting from a *Global Power Model* (GPM), which does not distinguish *P-states* nor application running on the server and then we apply enhancements to improve accuracy and precision of the models.

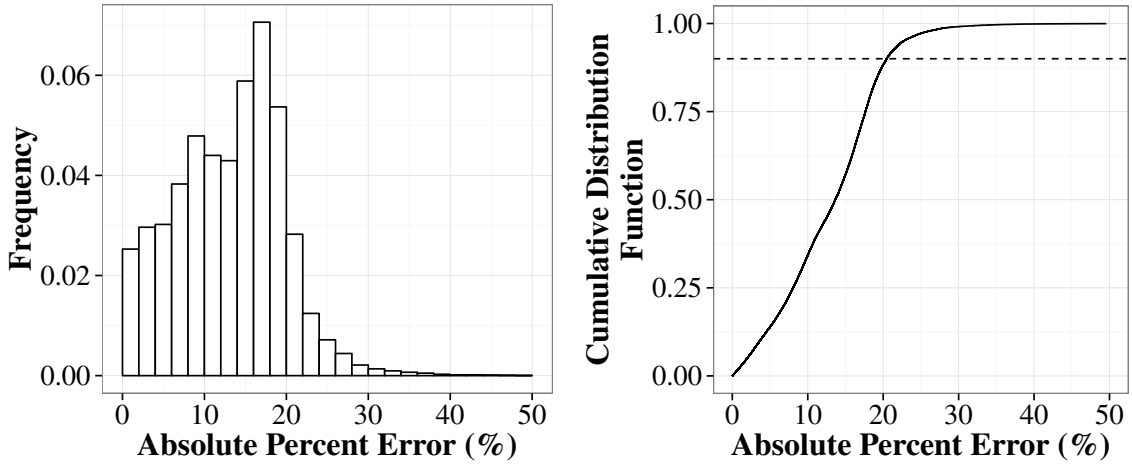
We observe that model generality, excess of parameters, and non-linear relation among power and performance measurements on I/O intensive applications impose limitations on the usage of linear regression techniques. We address these issues by developing computer system power models considering the following: (1) models that consider *P-states* as nominal parameters and the application that the computer is running, (2) models that make use of a machine-learning algorithm (CFS) for selecting the parameters most correlated to power, and (3) models that soften non-linear effects among computer system measurements and power by using K-means clustering.

4.3.1 Global Power Model

The first power model, called *Global Power Model* (GPM), is built using linear regression having the miscellaneous component and the hard disk power modeled as a constant value equal to the average power of these power components. We use points from all *P-states* and from both benchmarks (i.e. SPECint2006 and SPECweb2009) to develop this model; hence, we are not distinguishing *P-states* nor applications. For the CPU power component, the performance measurements described in Section 4.1.3 are the dependent variables and the CPU power the independent variable.

Figure 4.5 shows the histogram and the cumulative distribution function (CDF) for this model on the Intel i7 machine. There are outliers which can reach up to 120% of the absolute percent error (omitted in Figure 4.5(a)). Even though, most of the points are concentrated within 50% absolute error. The CDF (Figure 4.5(b)) clarifies this observation. The median is about 15% and the 90^{th} percentile is about 20% (dashed line) (i.e. 90% percent of the points display an absolute percent error lesser than 20%).

Rivoire et al. [57,58] claims that a power model must have the average for the absolute percent error lesser than 10% to be consider accurate. Therefore, this model does not fit this requirement. The remaining of this Section discusses enhancements done on the power models to improve their accuracy and precision.



(a) Histogram (b) Cumulative Distribution Function

Figure 4.5: Full-system power model for Intel i7 using the parameters described in Section 4.1.3. (a) presents the distribution for the absolute percent error, (b) presents the cumulative distribution function and the 90^{th} percentile of the absolute percent error in dashed line.

4.3.2 Nominal Parameters and Model Specificity

In Figure 4.4 we have shown that CPU power characteristics change when *P-states* change. Thus considering this parameter as nominal improves precision and accuracy. Hence, the first improvement that we do in GPM is to consider *P-state* as nominal parameter. In this approach, the CPU power is modeled by doing linear regression on the points of each P-state. This model is called *P-State-based Power Model* (pSPM). Differently from GPM, in pSPM, hard disk power and miscellaneous component power are modeled to a constant value equal to the average power on *each P-state*.

Figure 4.6 shows the cumulative distribution function for the GPM (solid line) and the pSPM (dashed line). The steeper the line, the more accurate the model. Thus, we observe that by doing a linear regression for each *P-state*, we improve the precision of the power models. Figure 4.6 also shows that for the AMD Opteron server both GPM and pSPM meet the accuracy requirements (i.e. average for the absolute percent error below 10%). However, they still do not meet the accuracy requirements for the Intel i7 server.

We have observed in our experiments that if a model is general (using points from either SPECint2006 or SPECweb2009), it needs more parameters to have similar accuracy and precision to a model developed for a specific application. To improve the pSPM, we consider the application that the machine is running. On Web Server environment the workload is known upfront, the power variation is mostly due to variation on the number of concurrent requests. Thus, we relinquish model generality and focus on an specialized

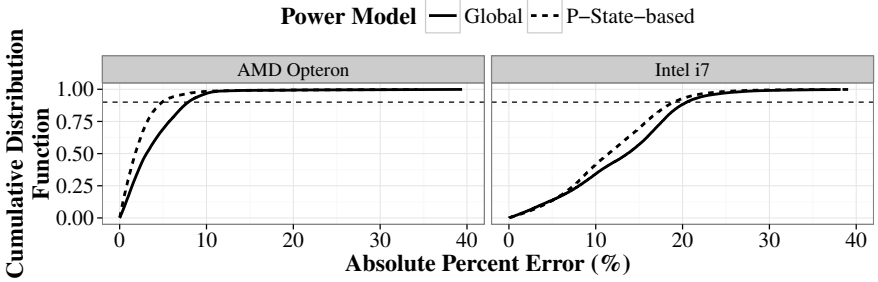


Figure 4.6: Comparison between Global Power Model (GPM) and P-State-based Power Model (pSPM). The latter is more precise, since it has narrower steeper CDF curve.

power model for Web Server application by considering just measurements done when running the SPECweb2009. This model is called *Web Server Power Model* (WSPM). In fact, we are targeting Web Server power models on our Data Center simulator; in this way, we take advantage of a more specific power model to also improve the simulator performance.

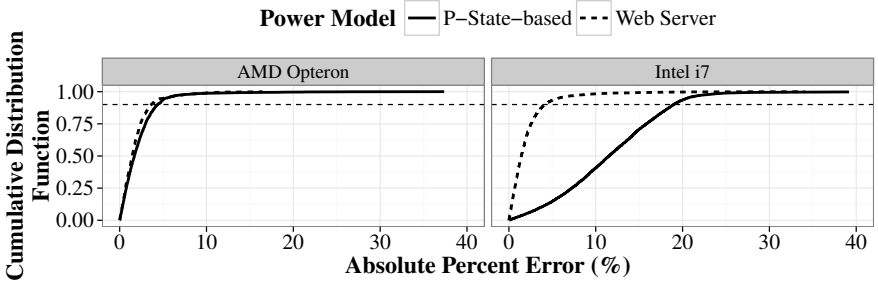


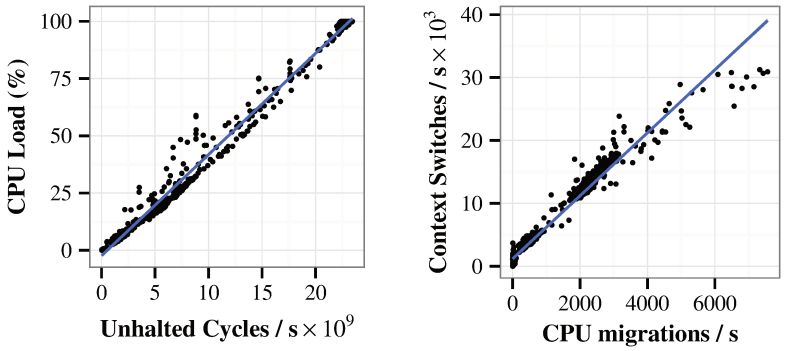
Figure 4.7: Cumulative distribution function (CDF) for the absolute percent error for P-State-based Power Model (pSPM) and Web Server Power Model (WSPM). The latter is more accurate having a steeper line.

Figure 4.7 presents the CDF for the pSPM and the WSPM We observe that the general power model is inaccurate specially on the Intel i7 server. On the other hand, an specialized power model has improved accuracy reducing the 90^{th} percentile of the absolute percent error on the AMD Opteron server from 4.4% to 3.5% and on the Intel i7 server from 18.9% to 4.3%. The WSPM meets the accuracy requirements, since it displays the average for the absolute percent error below 10%. However, this model requires all the parameters that were presented in Section 4.1.3. The next section shows how we reduce the number of parameters.

4.3.3 Pruning Model Parameters

The WSPM meets the accuracy requirements. However, it requires eight parameters plus the processor *P-state*. During our experiments, we observe that many of the performance measurements are highly correlated with each other. Since we are using linear regression models, we can eliminate highly correlated parameters preserving the model precision and accuracy.

Figure 4.8 shows examples of parameters that are correlated in the Intel i7 server at the highest frequency state using points from SPECweb2009 and SPECint2006. The line represents the linear regression between the two axes. The stronger the correlation between two variables the closer the points follow the line. In this way, on the left hand side, we notice that CPU load correlates well to unhalted cycles per second; and on the right hand side, we observe that CPU migrations per second correlates to context switches per second.



(a) (b)

Figure 4.8: Correlation between some of the model parameters. (a) CPU Load versus Unhalted Cycles per second and (b) Context Switches per Second versus CPU Migration per second

Consider the WSPM, which uses all proposed parameters in Section 4.1.3. This model can be stated as follows: $P_{CPU} = a_0 + \sum_{i=1}^{8} a_i \cdot p_i$ where p_i's are the performance measurements and a_i's their associated coefficient values. Table 4.2 shows the coefficients calculated using linear regression for all *P-states* on both architectures.

This model displays high coefficient of determination ($R^2 = 0.943$), which means that most of the cases can be explained by the model, but needs all parameters described in Section 4.1.3 and shown in Table 4.2.

In this paper, we use a correlation-based feature selection (CFS) [27] algorithm to facilitate choosing the parameters mostly correlated to power. The CFS algorithm focus

Table 4.2: Coefficients for the Web Server CPU Power Model at each *P-state* for both architectures

Server	Pstate	Const.	BIPS	Context Swts./s	CPU Load	Page Faults/s	CPU Migs./s	Unhalted Cycles/s	LL Cache Refs/s	LL Cache Misses/s
Intel i7	P0	20.77	18.83	6.69E-04	-1.2605	4.97E-04	-2.39E-03	-6.23E-09	2.00E-07	-1.89E-06
	P1	18.93	14.52	9.47E-04	-0.4822	2.58E-04	-1.09E-03	-7.28E-09	7.30E-09	1.46E-07
	P2	18.41	12.17	1.13E-03	-0.3459	2.64E-04	-1.74E-03	-6.03E-09	-1.26E-08	-8.27E-08
	P3	18.40	14.81	8.97E-04	-0.2476	2.42E-04	-2.30E-03	-1.03E-08	1.95E-07	7.75E-07
	P4	18.32	13.74	8.25E-04	-0.3675	1.98E-04	-2.54E-03	-8.46E-09	1.81E-07	5.73E-07
	P5	18.53	12.88	9.55E-04	-0.2574	2.14E-04	-1.15E-03	-7.81E-09	2.30E-08	5.35E-07
	P6	17.91	11.21	9.52E-04	-0.2387	1.80E-04	-2.40E-03	-6.49E-09	8.23E-08	3.58E-07
	P7	18.06	12.44	5.90E-04	-0.3628	1.59E-04	-1.32E-03	-7.24E-09	1.50E-07	6.23E-07
	P8	17.88	12.03	6.93E-04	-0.2775	1.60E-04	-1.24E-03	-6.56E-09	6.54E-08	3.74E-07
	P9	18.03	10.74	7.14E-04	-0.2064	1.41E-04	-2.06E-03	-5.92E-09	6.28E-08	5.92E-07
	P10	18.19	11.51	6.09E-04	-0.1187	1.25E-04	-1.70E-03	-7.50E-09	9.94E-08	8.15E-07
	P11	18.28	9.59	6.34E-04	-0.1077	8.85E-05	-1.40E-03	-5.82E-09	2.35E-08	1.07E-06
	P12	18.08	10.43	4.45E-04	-0.0269	1.08E-04	-8.90E-04	-7.37E-09	9.62E-08	7.91E-07
	P13	17.35	9.76	3.44E-04	-0.0715	1.19E-04	-9.05E-04	-6.28E-09	1.09E-07	6.70E-07
AMD Opteron	P0	46.95	-0.70	1.58E-03	0.4675	-1.03E-05	-3.94E-03	-4.68E-09	9.46E-09	2.32E-07
	P1	43.89	-12.46	2.27E-03	0.5750	1.95E-05	-2.34E-03	3.88E-09	2.73E-08	-7.30E-07
	P2	42.83	-10.84	2.03E-03	0.4801	7.61E-06	-2.15E-03	3.55E-09	2.45E-08	-6.23E-07
	P3	40.48	4.70	1.35E-03	0.3792	-1.02E-04	-7.51E-04	2.39E-09	-1.54E-08	-3.58E-07
	P4	42.28	3.14	1.10E-03	0.3851	4.64E-05	-2.51E-04	-8.52E-10	4.75E-09	-4.93E-07

on the job of feature selection for machine learning through a correlation based approach. The main assumption is that high quality feature sets (in our case all performance parameters) carry features that are highly correlated with the class (i.e. power). We believe that CFS is more appropriate for this task them other methods, such as Principal Component Analysis, because it returns a well determined subset of the input parameters (in contrast to other methods that rely the choice of them to experimenters). Therefore, it reduces the design space exploration and time analysing the models' accuracy. From our knowledge, this is the first work that uses this algorithm to support the choice of performance parameters to be used as proxies to power.

We develop the *Pruned Web Server Power Model* (PWSPM) by applying CFS on points of each *P-state*, which prune parameters. Then, we apply linear regression using CPU power as the independent variable and the CFS selected parameters as the dependent variables. Table 4.3 shows the coefficients calculated at each *P-state* for the Intel i7 and AMD Opteron servers. The algorithm is able to reduce from nine parameters to up to three parameters at each *P-state*.

Figure 4.9 shows the CDF for the WSPM and for the PWSPM (in dashed lines). The average of the absolute percent error for WSPM is 2.08% for the Intel i7 and 1.81% for the AMD Opteron. For the PWSPM, the average of the absolute percent error is 2.86% for the Intel i7 and 2.07% for the AMD Opteron. Therefore, PWSPM uses fewer parameters having similar accuracy and precision.

Table 4.3: Coefficients for the CPU Pruned Web Server Power Model at each *P-state* for both architectures

Server	Pstate	Cte	BIPS	Context Swts./s	CPU Load	Page Faults/s	CPU Migs./s	Unhalted Cycles/s	LL Cache Refs/s	LL Cache Misses/s
Intel i7	P0	24.69	1.98	1.17E-03						
	P1	18.92	8.93	1.05E-03	-1.24					
	P2	18.68	6.75	1.13E-03	-0.89					
	P3	18.47	6.85	1.05E-03	-0.86					
	P4	18.56	7.88	8.97E-04	-0.93					
	P5	18.13	7.17	1.00E-03	-0.84					
	P6	18.03	7.07	8.69E-04	-0.74					
	P7	17.99	7.33	7.45E-04	-0.69					
	P8	17.65	7.96	6.95E-04	-0.71					
	P9	19.49	2.63			9.15E-05				
	P10	18.81	1.88	4.62E-04						
	P11	18.38	1.91	4.56E-04						
	P12	18.27	1.90	4.46E-04						
AMD Opteron	P13	17.71	1.98	4.28E-04						
	P0	47.83	-1.52	1.03E-03	0.40					
	P1	47.88	-3.38	7.01E-04	0.67					
	P2	47.16			0.68					
	P3	42.62			0.59					
	P4	42.48			0.54					

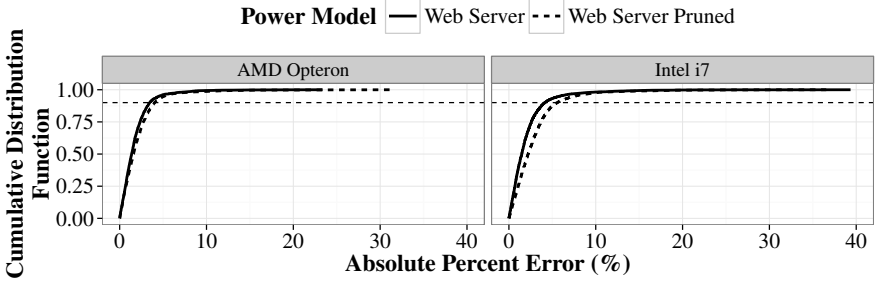


Figure 4.9: Cumulative distribution function (CDF) for the absolute percent error for Web Server Power Model (WSPM) and Pruned Web Server Power Model (PWSPM).

4.3.4 Softening non-Linear Effects

A Web Server needs to answer thousands of clients in a short period of time; hence, multiple processes are spawned creating an environment dominated by resource sharing. When using performance measurements as proxies for CPU power, non-linear relations are observed due to bottlenecks on shared resources, as noted by Rivoire [57].

Figures 4.10 and 4.11 show plots of CPU power versus BIPS and context switches, respectively, for Intel i7 sever at some of its *P-states* when running SPECweb2009 benchmark. The solid lines represent the linear model equation, while the dashed lines represent a log-log regression.

The non-linear effects are more remarkable at the higher performance states such as P0, P1, P2, and when running I/O bound workloads (i.e. SPECweb2009). At higher performance states, the frequency is higher and the CPU is relatively faster than the

I/O devices; hence, it becomes idle more often waiting for I/O. Moreover, the CPU can handle more requests, increasing the competition for the shared resources, such as memory controllers and hard disk. Therefore, the CPU operates in bursts of processing creating non-linear effects among power and performance measurements.

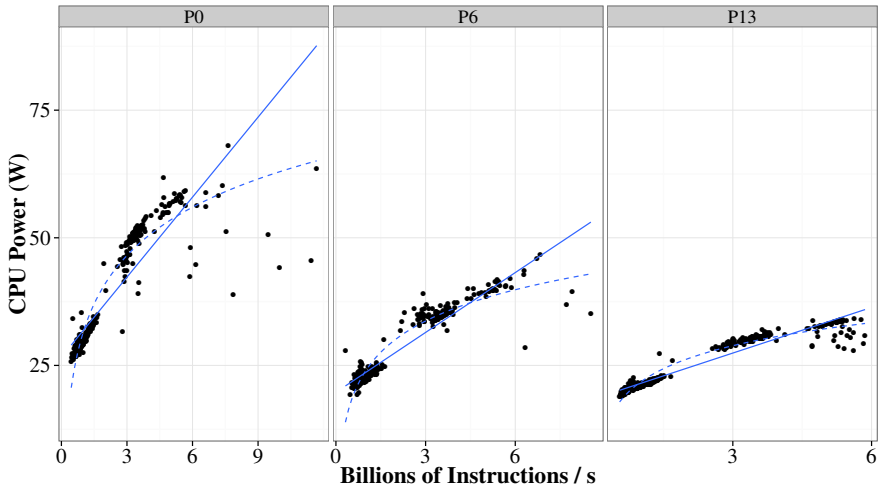


Figure 4.10: i7 power versus BIPS. The lines represent the linear regression while the dashed lines represent a log regression.

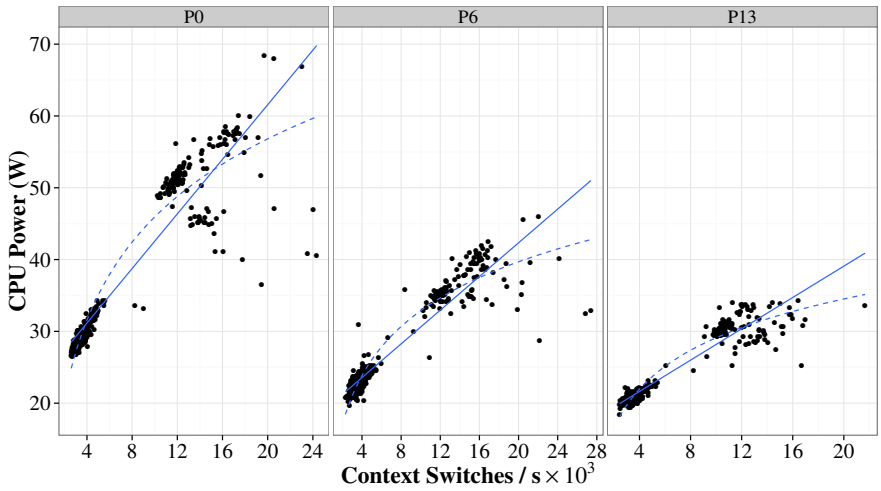


Figure 4.11: i7 power versus Context Switches per second. The lines represent the linear regression while the dashed lines represent a log regression.

Applying polynomial regression might be difficult when the polynomial order is unknown. Using logarithm regression results in models that do not take into account "idle power", because when no activity is observed in the computer the model yields a power

value that is equal to zero. We use K-means clustering technique [44] to soften the non-linear effects. This approach approximates a non-linear curve using multiple linear segments creating a different equation with a different slope for each cluster. We use average values of HD power and miscellaneous components power for *each cluster* at each *P-state* to derive these two components power model.

K-means clustering is a cluster analysis method which targets to divide n observations into k clusters where each point lies into the cluster with the nearest mean. In a formal presentation, let $\boldsymbol{X} = \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ be a set of observations, where each observation is a d-dimensional real vector, K-means clustering partitions the n observations into k sets $\boldsymbol{S} = S_1, S_2, ..., S_n$ $(k \leq n)$, minimizing the within-cluster sum of squares, where $\boldsymbol{\mu}_i$ is the mean of points in S_i:

$$\arg\min_{c} \sum_{i=1}^{k} \sum_{\boldsymbol{x}_j \in S_i} \parallel \boldsymbol{x}_j - \boldsymbol{\mu}_i \parallel^2 \tag{4.2}$$

To obtain models with higher accuracy, we need to select the most pertinent set of variables that is used to create the clusters (based on CFS algorithm), the most suitable distance function, and the most appropriate number of clusters (i.e. k). This model is called *Cluster Web Server Power Model* (CWSPM).

Selecting the most pertinent variables: When using cluster techniques, we observe that using a large number of attributes (i.e. all the gathered events) yields less precise models. We attribute this fact to the presence of high correlation among clustering variables which might overweight one or more parameters [34]. Hence, we apply the CFS algorithm to obtain a smaller set of variables mostly correlated to power.

Choice of the most suitable distance function between the points: We use the euclidean distance for the K-means clustering distance function in our models.

Selecting the most suitable number of clusters: To select an appropriate value of k, we use the sum of the squared errors of prediction (SSE) (also known as an F-test) as a function of the number of clusters and observe when adding another cluster does not yield better modeling of the data. This can be done visually by using "elbow criterion" where small values of k explain most of the variance. At some point the gain drops, resulting in an angle in the graph where the number of clusters is set.

Figure 4.12 shows the elbow plots for some *P-states* on the Intel i7 server where the circles highlight the number of clusters selected for each *P-state* (i.e. the location of the "elbow"). Note that the CFS algorithm may select different sets of variables for each *P-state* as shown in Table 4.3.

In order to explain the reasoning behind the usage of the K-means clustering, lets focus on the case of P0-state of the Intel i7 server. In this case, CFS returned BIPS and number of context switches per second as the performance statistics that should be used

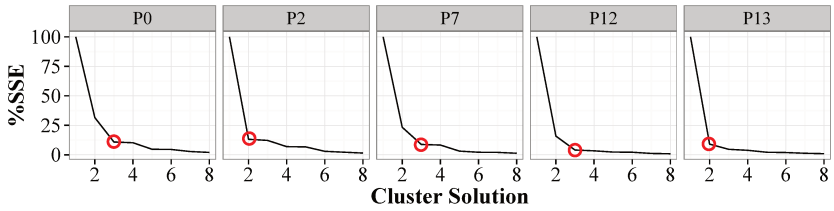


Figure 4.12: Elbow plot for determining the best number of clusters for the Intel i7 server power model. The circles highlights the number of clusters chosen to partition the data set.

to estimate the CPU Power. The "elbow criterion" selects $k = 3$. K-means clustering on the 2-dimensional observations (BIPS and number of context switches per second) is applied. Figure 4.13 shows the result of the clusterization for this case.

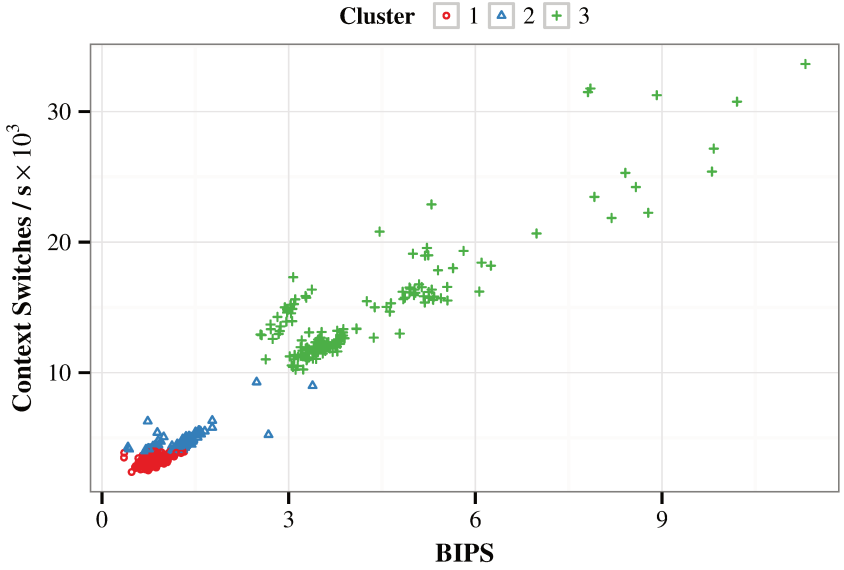


Figure 4.13: Result of K-means clustering for the Intel i7 server at P-0 state after CFS selected BIPS and number number of context switches per second as the performance parameters to be used to estimate CPU Power.

Figure 4.13 shows that the regions are well defined and are making a discretization of the points based on the system activity. The non-linear effects start to be noticed at higher activity levels (higher BIPS and higher number of context switches per second), as noticed in Figure 4.10 and Figure 4.11. The K-means clustering algorithm is making a discretization of the points based on the system activity level, as shown in Figure 4.13. Therefore, by applying a different linear-regression for each region, the non-linear effects

are softened.

Finally, Figure 4.14 shows a plot of CPU power versus BIPS for a subset of the data points for the Intel i7 server for all its *P-states* and we can see the effect of the discretization made by the K-means clustering.

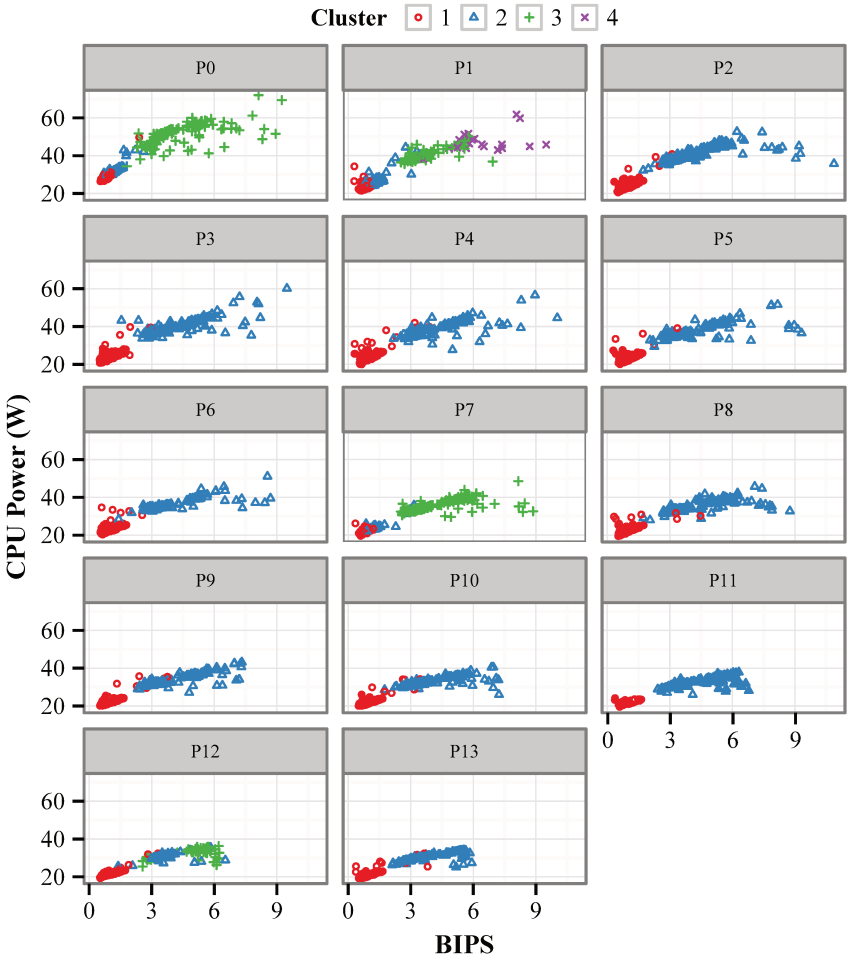


Figure 4.14: CPU Power versus BIPS for a subset of the testing set in their respective cluster for the Intel i7 server. K-means cluster groups up the points and linear regression is applied on each cluster to soften non-linear effects among power and the parameters.

Figure 4.15 shows the comparison among the three models. By using K-means clustering, we come up with a model that uses fewer parameters and is more accurate than a model that uses all of the others. Using fewer parameters is important for the simulator, since the input data could be reduced and is also important on real-time power estimation because fewer system measurements need to be sampled.

Finally, Figure 4.16 shows an analysis for each *P-state* on each architecture. We can see that all models meet the accuracy requirement (i.e. they display average for the

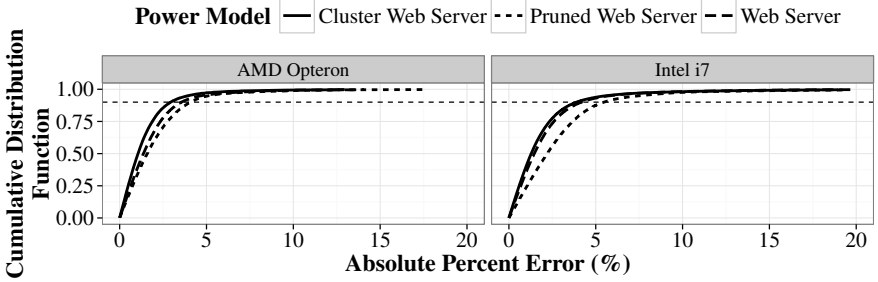


Figure 4.15: Cumulative distribution function (CDF) for the absolute percent error for WSPM, PWSPM, and CWSPM. The latter is the best model in terms of accuracy and also uses fewer parameters.

absolute percent below 10%) at all frequency states. Furthermore, the CWSPM model uses fewer parameters than the WSPM and is the most accurate at all *P-states* but P9 on the Intel i7. Therefore, we select the CWSPM as the best model.

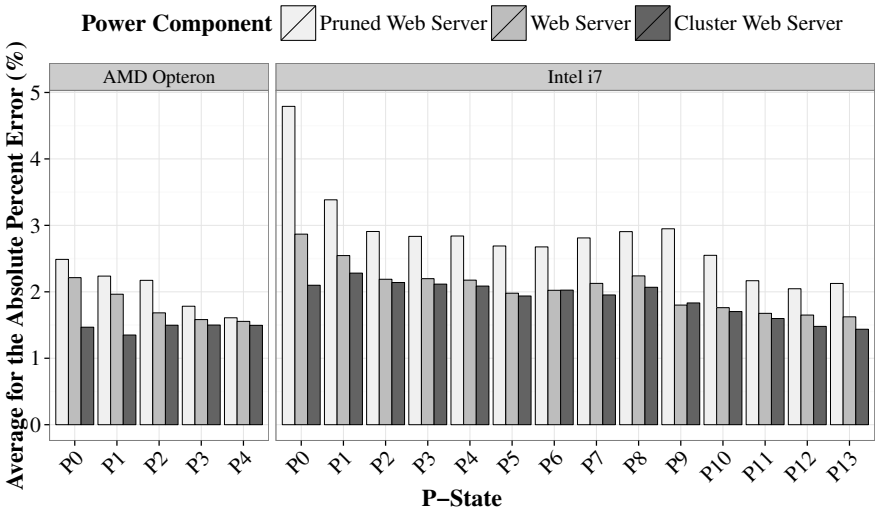


Figure 4.16: Average of the the absolute percent error for WSPM, PWSPM, and CWSPM.

Chapter 5

Global Power Optimization for Web Server

This Chapter presents the implementation of the *Slack Recovery* algorithm configured to minimize power under a performance threshold on a state-of-the-art, high-density, power-efficient SeaMicro SM15k cluster by AMD. It also presents a CPU utilization rate control mechanism, which is essential for the implementation of the Virtual Power States. Experimental results show that our *Slack Recovery*-based system can reduce up to 16% of the power consumption when compared to the Linux *performance* governor and 6.7% when compared to the Linux *ondemand* governor.

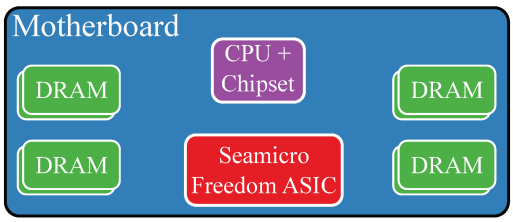
5.1 SeaMicro Cluster Overview

We deploy our Power and Performance Optimizations on an AMD's SeaMicro SM15k (Figure 5.1(a)) family of Fabric Compute Systems (SM15k cluster) [3]. The SM15k cluster is a high-density cluster composed of compute nodes, networking, and storage on a single 10 Rack Unit (RU). This system amortizes the power overhead of fixed system components such as power supplies and fans among the cluster by sharing them among the cluster nodes. Our SM15k cluster has 64 server cards and consumes between 3.0 kW and 3.5 kW.

A server card is logically described in Figure 5.1(b). Each server card is composed of DRAM, CPU+Chipset, and the Freedom ASIC. The latter includes network interfaces, which removes the need of network adapters, cables, and switches, resulting in a high-density and energy efficient cluster. It also implements an I/O virtualization technology, which virtualizes the disks to the server nodes. Each node accesses the disk as a virtual SATA disk. This feature reduces power and space without requiring any special software and driver. In our configuration, we used server cards composed of one Intel Xeon E3-1265Lv2 processor, 32 GB ECC DRAM, 8x1Gbit network interface card, connected with

one virtual SATA disk. Each virtual disk was assigned to one physical disk.

(a) SM15k front picture



(b) Freedom based motherboard

Figure 5.1: AMD's SeaMicro SM15k family of Fabric Compute Systems

The Freedom ASIC enables interconnection of servers in a 3D-torus topology with 1.28 Tbit bandwidth and less than 6μs communication delay between any two server nodes. Our SeaMicro SM15k cluster processors are from the Intel Ivy Bridge family, and feature special registers that provide estimates to the CPU power consumption. We took advantage of the fast interconnection and the embedded power estimators when implementing our power and performance optimization algorithms.

5.2 System Architecture Overview

In our previous work [6] we used Integer Linear programming (ILP) to find an optimal solution to the problem of minimizing power by selecting discrete frequency and voltage levels (P-states) while maintaining performance above a minimum threshold. This approach, however, did not scale for large number of nodes, because of the complexity of the ILP problem. To overcome this limitation we also developed a heuristic algorithm called *Slack Recovery* and implemented it in the simulation environment described in [6].

In this work, we took the *Slack Recovery* algorithm and adapted it to a real cluster environment. We used it to select the P-states in each CPU node in order to optimize the overall Web Server cluster power, while satisfying a given level of total performance (i.e., a performance threshold).

Current processor cores, such as the Intel Xeon E3-1265Lv2, have internal mechanisms in hardware and firmware to change its P/C-state according to its load and power. According to ACPI tables found in operating systems, Intel Xeon E3-1265Lv2 has 11 P-states, or 11 different levels of frequency and voltage.

In high-density servers, such as SM15k cluster, peripherals (e.g., power supplies, fans, and disks) are shared among the computing nodes; thus, their power component is lower than on regular servers. Therefore, in this work, we considered only the CPU power. Intel Xeon E3-1265Lv2 processors feature RAPL (Running Average Power Limit) interfaces [29, 59] which, among other capabilities, provide a power metering interface. We developed a Linux kernel module that reads these power registers and provides a power estimate for the CPU.

Our performance metric is billions-of-instructions-per-second ($BIPS$). We developed an additional piece of software that uses libpfm-4.3 [42], to monitor the CPU performance counters, allowing the measurement of number of instructions executed as well as user, system, I/O, and idle times.

In our optimization problem, we observed that the CPU presents different power consumption and different performance levels (measured in $BIPS$), under different utilization rates. We define utilization rate as the ratio of the time that the CPU is doing useful work (i.e., the CPU is not in idle mode) over the total amount of time in the observation window. Figure 5.2 shows the relation of $BIPS$ and CPU power for different P-states. For example, a performance level of ten $BIPS$ can be achieved in five different configurations consuming from 16.7 W to 22 W as follows: P4-state under 100% utilization rate, P3-state under 95% utilization rate, P2-state under 91% utilization rate, P1-state under 86% utilization rate, and P0-state under 77% utilization rate. This observation leads us to add the utilization rate as an extra dimension to the optimization problem. Therefore, our problem is to find a P-state and a utilization rate for each CPU that minimizes power under a minimum performance requirement.

In Figure 5.2, we derived an envelope curve, which is the Pareto Frontier of states. A point in the Pareto Frontier will provide the power consumption and the performance of the CPU under a P-state and a utilization rate. Moreover, there is no other state and utilization rate that would result in higher performance for the same power, or lower power for the same performance. The union of the Pareto Frontier states and the idle C-states constitute the set of VPS for a core. These Virtual Power States are the states used by our optimization algorithm.

We showed how a *Slack Recovery*-based system can be deployed in practice using a production, state-of-the-art, SeaMicro SM15k cluster, instead of simulation, and to show how we can integrate a utilization rate control technique to such a system. The reader may find more details about the optimization problem in our previous work [6].

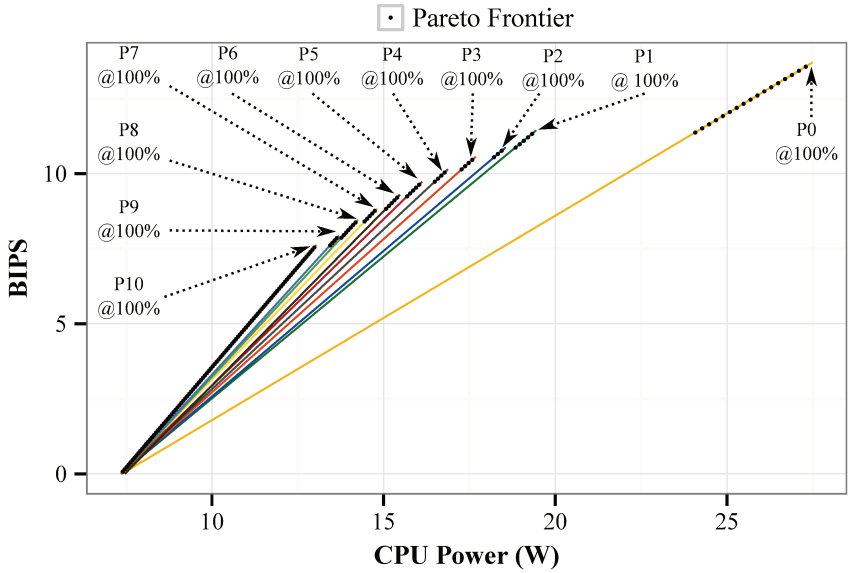


Figure 5.2: Pareto frontier of P-States on Intel Xeon E3-1265Lv2

We organized our system as follows: a Web Server Cluster, which handles HTTP requests, and a power manager and load-balancer node that configures the cluster to an optimal power state and distributes the load accordingly, as illustrated in Figure 5.3. Each Web Server is composed of a back-end that runs MySQL 5.5.20 and a front-end that executes the Nginx 1.0.10 Web Server with PHP 5.3.5. The back-end and the front-end run on the same node in order to fully utilize the CPU. The Web Server application is the CloudSuite Web Server benchmark [24] running Olio, a Web 2.0 web-based social calendar. The power manager is the implementation of the power and performance policy (e.g., *Slack Recovery*, Linux *ondemand* governor). The load-balancer is the HAProxy 1.5 [65], which is a fast, reliable, and open-source proxy solution.

The Web Server Cluster produces the sensor data, that is, readings of the power and performance values of each node in the cluster. The manager then takes into account the power optimization policy and the input workload to determine, on-the-fly, the best configuration of power states for all CPU nodes (i.e., which CPU node needs to have its power state or utilization rate changed). It then passes this new configuration information and the new workload distribution back to the cluster which is then reconfigured while the workload is running.

We characterized one server node of our SeaMicro SM15k cluster when running the Olio benchmark in order to derive the pareto frontier shown in Figure 5.2. We set all CPU cores to a given P-state and ran the benchmark for an increasing number of concurrent users until we found a number of users that makes the CPU operate at 100% of utilization.

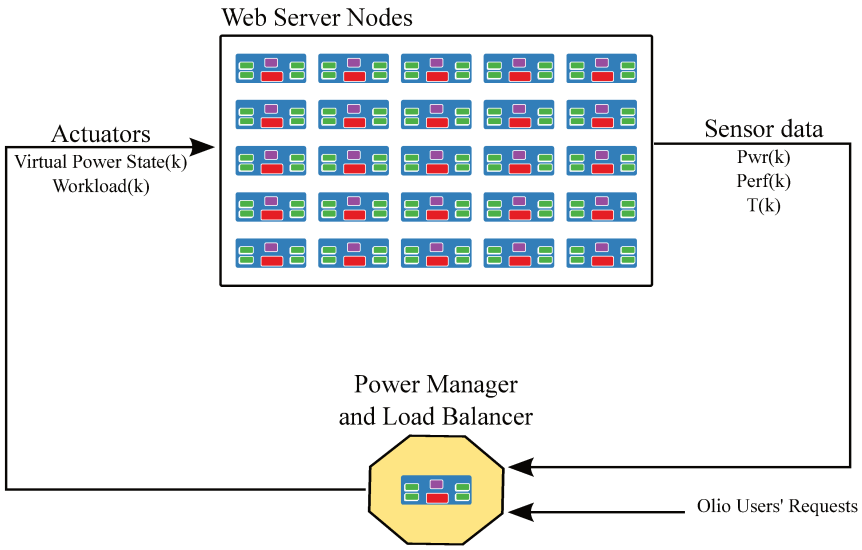


Figure 5.3: High-level description of system organization to deploy our global optimization algorithm

At this point, we ran the benchmark measuring power and instructions to determine the $BIPS$ at maximum utilization rate and power at maximum utilization rate for each P-State. We measured the power at idle for all P-States in order to obtain the idle power ratio. For this CPU, the idle power is around 7.3W for all P-States.

5.3 Experimental Methodology

In order to recreate a realistic workload stream, we obtained the load distribution over time for a specific server cluster hosting the chat room from a large Internet provider. This load is represented in Figure 5.4, which shows the intra-day variation (for 37 hours) of the load on the servers (where the load is represented as a percentage of the maximum load supported by the whole server cluster). We modified CloudSuite Web Server Benchmark [24] to follow this trend line creating a synthetic load representing the same variation with the same characteristics and same relative load percentage with respect to each experiment.

The remainder of this section describes the CPU frequency scaling algorithms, our test bed, the software tools used to evaluate the *Slack Recovery* implementation on a SM15k cluster, and the methodology for predicting performance.

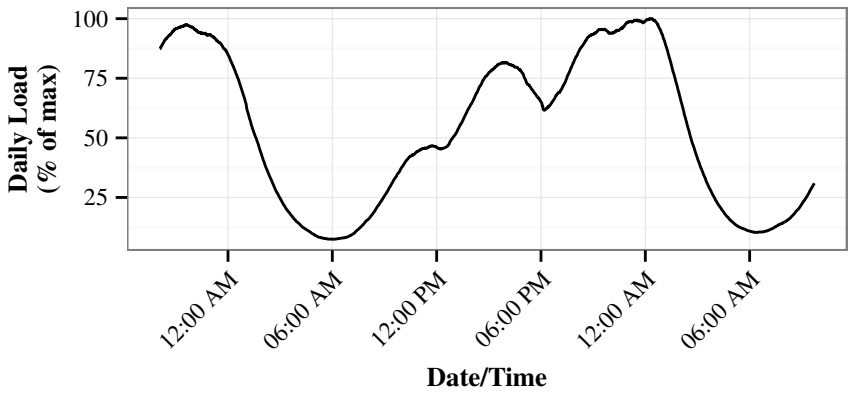


Figure 5.4: Intra-day variation of the load on a server

5.3.1 CPU Frequency Scaling Algorithms

This section describes the CPU frequency scaling algorithms used in this work. We evaluated the algorithms Linux *performance* governor, Linux *ondemand* governor, which performs local power optimization, and the *Slack Recovery* algorithm, which performs global power optimization.

The first algorithm is the Linux *performance* governor [14], where the CPU frequency is kept at the maximum when the CPU is executing any instruction. When in idle, it is automatically put in power-safe states (also known as C-States) by the CPU hardware [59].

The second algorithm is the Linux *ondemand* governor [14]. The governor samples the CPU usage at small intervals (typically 10ms) and decides which frequency to set the CPU based on its utilization rate. A transition is done when the CPU utilization rate is more than a threshold (typically 95%) in an interval.

The last algorithm was introduced in our previous work called *Slack Recovery* [6]. This heuristic algorithm is based on the idea of *Slack Recovery* (in power) to determine a near optimal solution. At first, it assigns power states to all CPU nodes to a state of the largest slack possible, that is, they are set to the highest performance virtual state, so there will be performance slack to be exchanged for power. In this case, the algorithm will switch the states of certain CPU nodes to decrease power and consequently lowering performance up to a threshold.

The algorithm runs over a cluster model to find the optimal configuration and at the end it assigns the VPS to the physical cluster. The cluster model is a set of node models, which are implementations of the Pareto Frontier described in Section 5.2. The implementation needs a model because it requires the estimation of power consumption and performance for different configurations. The model also reduces significantly the number of virtual state transitions.

For the sake of completeness in this text, we present the basic steps of the *Slack*

```
1 bool SlackRecoveryPerNode::minimizePower(
2                 float minBIPS, float powerCap,
3                 float actualPower, float actualBIPS) {
4
5   if (!findInitialStateOnMinimizePower(minBIPS,currPower,currBIPS))
6     return false; // No feasible solution
7
8   // Now we are ready to execute the slack recovery algorithm
9   do {
10    nodeSel = NULL;
11    for all nodes n do
12      // There is slack: attempt to decrease power slack by moving
13      // to a lower power/performance state
14      if (n.getCurrentVirtualState() < NumVirtualStates -1) {
15        // Check if we meet the constraint for the next state
16        bipsChg = currBIPS  -
17            provisionBIPS(currBIPS,n.getCurrentVirtualState(),
18                n.getCurrentVirtualState()+1);
19        if (currBIPS - bipsChg >= minBIPS) {
20          // Provision power for the next state
21          nextSttPower =
22              provisionPower(currPower,
23                             n.getCurrentVirtualState(),
24                             n.getCurrentVirtualState()+1);
25          powerChg = currPower - nextSttPower;
26          if (powerChg > bestPowerChg) {
27            nodeSel = n; // Remember this node
28            bestPowerChg = powerChg;
29            bestBipsChg = bipsChg;
30          }
31        }
32      }
33    }
34    if (nodeSel != NULL) {
35      currBIPS = currBIPS - bestBipsChg;
36      nodeSel->oneStateMoveUp();
37    }
38  } while (nodeSel != NULL);
39  // Do the assignment in the actual nodes
40  assignVirtualStates();
```

Figure 5.5: Slack-Recovery Algorithm Pseudo-code

Recovery algorithm. For further details, the reader should refer to Bergamaschi et al [6]. Figure 5.5 lists the *Slack Recovery* pseudo-code configured to minimize power given a minimum performance. The algorithm starts by finding an initial configuration state (line 5) that is able to sustain the minimum performance requirement. Next, it executes the slack routine, where it tries to decrease power by moving to a lower performance state

(lines 9 to 38). The routine iterates over all model nodes (lines 11 to 33), checks if the node could be moved to a higher VPS, which means a state with lower performance and power consumption (line 14). The algorithm provisions the performance (lines 16 and 17) in the cluster model and checks if it is greater than the performance threshold (line 19). If it satisfies the performance constraint, it provisions power and checks if it is lower than the minimum configuration so far (lines 22 to 24), storing this node. After iterating over all nodes, the algorithm moves the selected node (if any) to a higher VPS (lines 34 to 37). Finally, it assigns the VPS to the actual cluster nodes (line 40).

5.3.2 Scaling-out Web Server Benchmark

We used 63 server nodes configured as follows: 25 nodes running the Web Servers; 37 nodes running as client machines; and one node executing the power manager and the load balancer.

Although HAProxy is one of the fastest known proxy implementations, its TCP stack saturated and the Operating System quickly ran out of TCP ports when we made all the client-server traffic pass through it. Therefore, we had to adapt the benchmark in order to alleviate the proxy server traffic. This was accomplished by developing a mechanism to emulate a reconfigurable network switching fabric. In this mechanism, the client requests a connection to the proxy (I in Figure 5.6). The proxy forwards the connection to a server from the pool, selected based on a weighted round-robin policy (II in Figure 5.6). During the next five seconds, the follow-up requests coming from the same client are directly addressed by the server selected in the first connection call (III in Figure 5.6). Then, this connecting process is repeated.

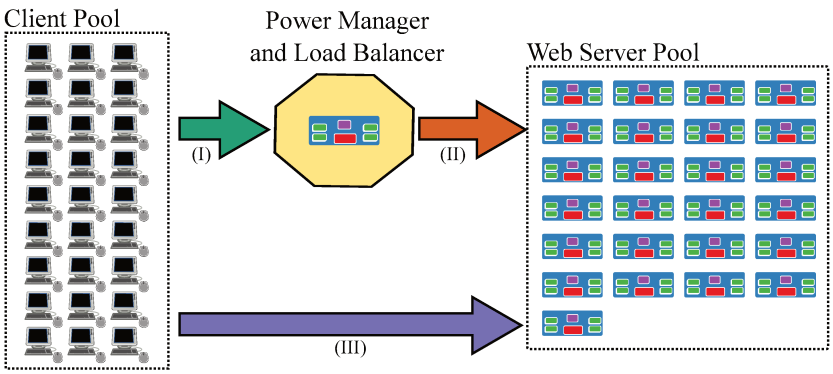


Figure 5.6: System configuration to avoid proxy saturation

On a large data center, it is possible to probe reconfigurable network switching fabrics that perform load balancing in order to implement our power management technique.

Therefore, our requirements are the provision of a mechanism to change the weights of a round-robin load balancer and a metric that reports the amount of traffic redirected for each server.

5.3.3 Predicting the Demanded Performance

The predictor is developed by using simple linear extrapolation as follows: let Sr_{t-1} be the session rate in time window Π_{t-1} and Sr_t be the session rate in time window Π_t. To estimate the session rate Sr_{t+1} in time window Π_{t+1}, we find a tangent line as by using Equation 5.1. Figure 5.7 illustrates the extrapolation method used by our predictor.

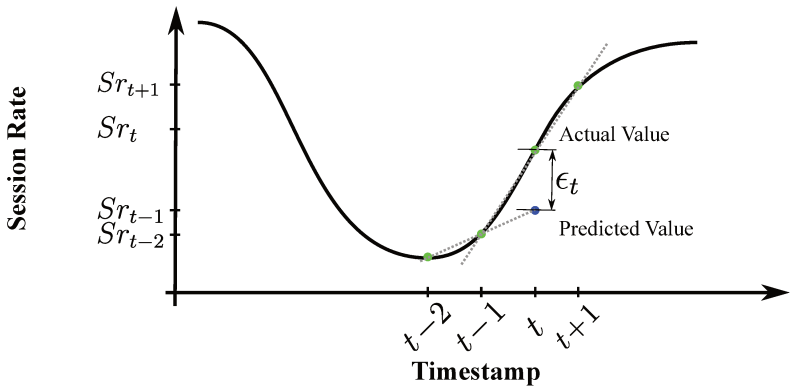


Figure 5.7: Variables used to predict the value of the next session rate, Sr_{t+1}

$$\begin{aligned} Sr_{t+1} &= \alpha \cdot (t+1-t) + Sr_t \\ &= \alpha + Sr_t \end{aligned} \tag{5.1}$$

where α is the slope of the line given by Equation 5.2.

$$\begin{aligned} \alpha &= \frac{Sr_t - Sr_{t-1}}{t-(t-1)} = \\ &= Sr_t - Sr_{t-1} \end{aligned} \tag{5.2}$$

Now replacing Equation 5.2 in Equation 5.1, Sr_{t+1} can be calculated as follows:

$$\begin{aligned} Sr_{t+1} &= alpha + Sr_t \\ &= Sr_t - Sr_{t-1} + Sr_t \\ &= 2 \cdot Sr_t - Sr_{t-1} \end{aligned} \tag{5.3}$$

We also add an estimator error, ϵ_t, in order to amortize the prediction errors, calculated as follows:

$$\epsilon_t = \begin{cases} 0 & \text{if } t = 0 \\ Sr_{t_{Actual}} - Sr_{t_{Predicted}} & \text{if } t > 0 \end{cases} \tag{5.4}$$

Finally, the predicted session rate for the next time window is given by adding Equation 5.3 to Equation 5.4 as follows:

$$Sr_{t+1} = 2 \cdot Sr_t - Sr_{t-1} + \epsilon_t \tag{5.5}$$

The performance prediction is done based on the variation of the system-wide workload. We empirically observed that the total workload, when measured at discrete time intervals (i.e., 10 seconds in this work) tends to change smoothly with very few inversions. In addition, even when the load variation changes direction, the prediction may get it wrong for one or two intervals at most, before correcting itself.

After having predicted the performance for the next time window Π_{t+1}, we used this information as an input parameter for *Slack Recovery*. The algorithm returns a set of VPS and we set each node to the corresponding virtual power state. The next Section, discusses the implementation of VPSs in our environment.

5.4 Virtual Power State Implementation

A Virtual Power State (VPS) is a tuple containing a utilization rate and a P-state. While one can easily set a node's P-state; controlling each individual CPU utilization rate is more difficult. This section discusses our approach to controlling the CPU utilization rate on a Web Server cluster.

In order to follow the remainder of this section, let's first define a number of variables used in the ensuing formulation. Table 5.1 presents the variables and their corresponding definitions.

Each server will handle a certain number of users until a point at which the CPU utilization rate reaches 100%. Thus, one way to control the CPU utilization is to change the number of users connected to a server. Figure 5.8 shows the relation of the users connected to a server and its corresponding utilization rate for the Olio benchmark when the server is operating at highest frequency. when the server is operating at highest frequency (P0-state). When the server is operating at other P-states, we normalize the utilization to P0-state using Equation 5.7 and Equation 5.8.

The CPU utilization rate (Ψ) could be used to estimate the number of connected users (μ) to a Web Server using a second order degree polynomial fit from the curve in

Table 5.1: Variables for the Formulation of CPU Utilization Rate Controller

Symbol	Definition
k	P-state index
s	Server index
μ	Number of users
χ_s	Number of connected users to the server s
ν_s	Number of new users to the server s
Ψ	Utilization rate
$T_{s,k}$	Target utilization rate for server s running at P_k-state
$K_{s,k}$	Current utilization rate for server s running at P_k-state
ϕT_s	Normalized target utilization rate for server s
ϕK_s	Normalized current utilization rate for server s
Δ_s	$\phi T_s - \phi K_s$
ϵ_s	Error in number of users
η_s	Number of expected users to the next iteration

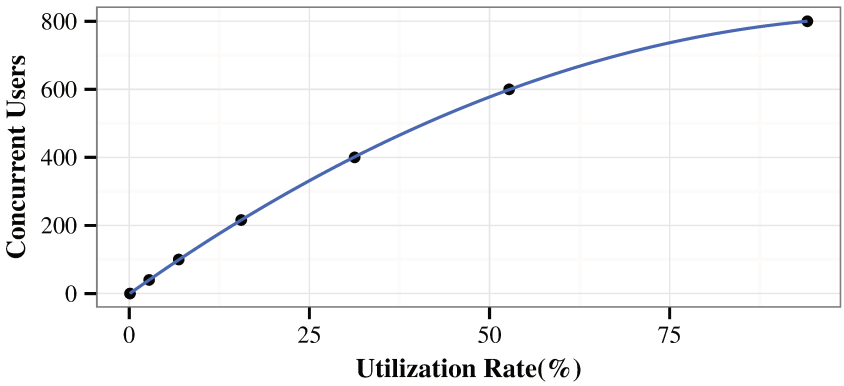


Figure 5.8: Number of benchmark users (μ) versus utilization rate (Ψ). We use this relation to estimate the number of users that should be connected to a given server node

Figure 5.8, as given in Equation 5.6.

$$\mu = f(\Psi) = -0.98 + 15.03 \cdot \Psi - 0.07 \cdot \Psi^2 \tag{5.6}$$

As we have shown in Section 5.3.2, the HAProxy will redirect a benchmark user to a Web Server following a weighted round-robin policy, and this user-server assignment lasts 5 seconds. So, we can control the number of users that will be redirected to a given machine by changing the server weight.

Therefore, if we want to reduce the utilization rate of a server (s), we need to reduce the number of new users (ν_s) arriving at the server and wait until the number of connected users (χ_s) drops, which means that we need to wait until some connected users finish their requests, entering into the connection phase again (where they request another server to the proxy). Figure 5.9 shows the variation of the CPU utilization rate over time when

a server has 800 connected users and we suddenly stop binding new users to it. We can see that the CPU utilization rate over time can be modeled as a step function, which facilitates the system modeling using modern control theory as we show later in this Section.

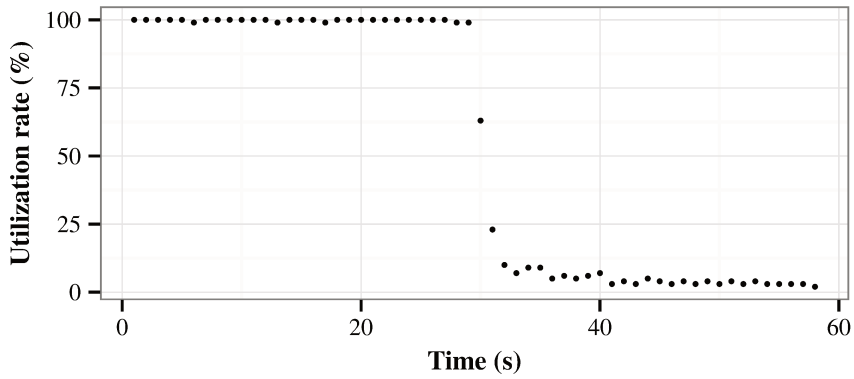


Figure 5.9: Transient Response for Utilization Rate: Behavior of the utilization rate on time when server stops receiving new users

By using this methodology, we developed a Utilization Control Agent (UCA) to control the CPU utilization rate of each server on the Web Server Cluster. Figure 5.10 illustrates the elements involved in this implementation.

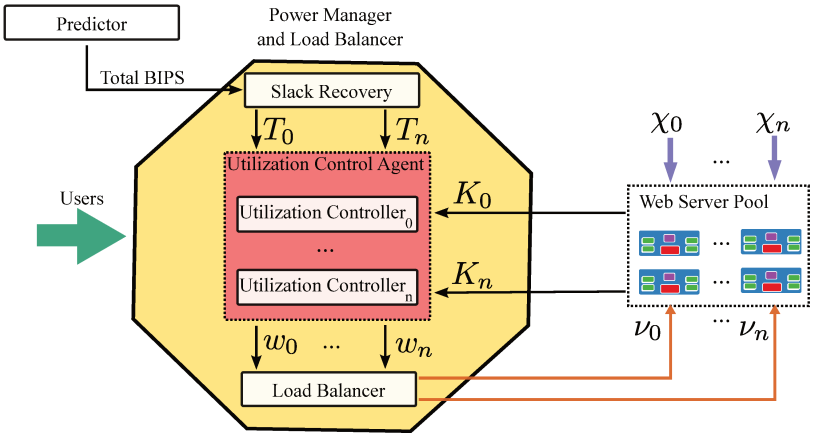


Figure 5.10: An overview of the components involved on the implementation of the utilization controller mechanism

Each server (s) running at P_k-state is assigned to a target utilization rate ($T_{s,k}$) by the *Slack Recovery* algorithm. The UCA receives from each server (s) its corresponding current P-state (k) and its current utilization rate ($K_{s,k}$). $T_{s,k}$ and $K_{s,k}$ are normalized to

P_0-state, since the performance at 100% utilization rate is different among P-states. This is accomplished by using the information in Figure 5.2, which displays the maximum performance ($BIPS_{100\%}$) for each P-State. Thus, the UCA calculates the normalized target utilization rate, (ϕT_s) and the normalized current utilization rate (ϕK_s) as stated by Equations 5.7 and 5.8.

$$\phi T_s = T_{s,k} \cdot \frac{BIPS_{100\%@P_k}}{BIPS_{100\%@P0}} \tag{5.7}$$

$$\phi K_s = K_{s,k} \cdot \frac{BIPS_{100\%@P_k}}{BIPS_{100\%@P0}} \tag{5.8}$$

We apply modern control theory to model an utilization controller as illustrated in Figure 5.11. The controller box in this figure is the HAProxy, which will have changed its weights associated to each server. This will be translated to an increment or decrement to the number of new users. The number of new users plus the number of connected users will impact the utilization rate. The measured utilization rate of the server is compared to the target utilization rate and will be translated to a new weight closing the loop.

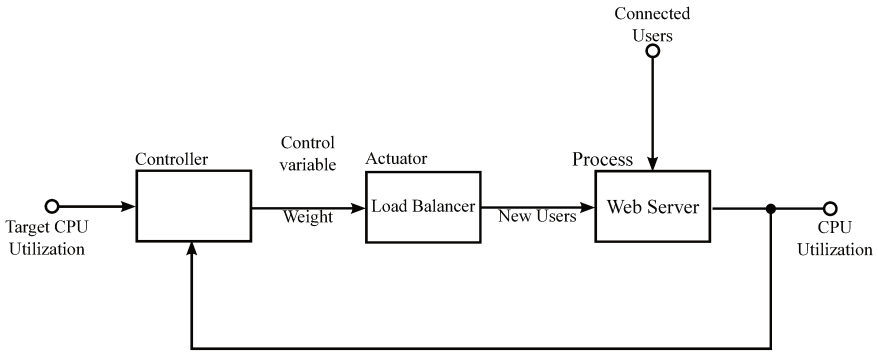


Figure 5.11: Control loop approach for enforcing a CPU utilization rate

An utilization controller converts ϕK_s to number of connected users (χ_s) as follows:

$$\chi_s = f(\phi K_s) \text{ ,where } f \text{ is given by Equation 5.6}$$

Let Δ_s be the difference between the normalized target utilization rate and the normalized current utilization rate as follows:

$$\Delta_s = \phi T_s - \phi K_s$$

The error in number of users (ϵ_s) is calculated using the absolute value of Δ_s as follows:

$$\epsilon_s = f(|\Delta_s|) \text{ ,where } f \text{ is given by Equation 5.6}$$

A Δ_s greater than zero means ϵ_s users must be added to the server s, in order to make it reach the target utilization rate. Otherwise, ϵ_s users must be removed from the server. Therefore, the number of expected users to the next iteration (η_s) on the s server is given by Equation 5.9.

$$\eta_s = \begin{cases} \chi_s - \epsilon_s & \text{if } \Delta_s < 0 \\ \chi_s + \epsilon_s & \text{if } \Delta_s \geq 0 \end{cases} \tag{5.9}$$

The HAProxy allows weights from 0 to 254; thus, the UCA converts each η_s to a HAProxy weight w_s as follows:

$$w_s = round\left(\frac{\eta_s}{max(\eta)} \cdot 254\right)$$

In order to evaluate the utilization control mechanism, we set up a small cluster composed of 5 servers. We configured the benchmark to generate a fixed input load that would be sufficient to keep the servers' utilization at their respective utilization rate targets. Our problem is to distribute the load across the servers by adjusting the load balancer weights in order to keep the servers' utilization rate at their respective targets. The controller will adjust the HAProxy weights in order to keep the utilization rates at their targets. Figure 5.12 shows the behavior of the utilization along 100 seconds for a fixed load.

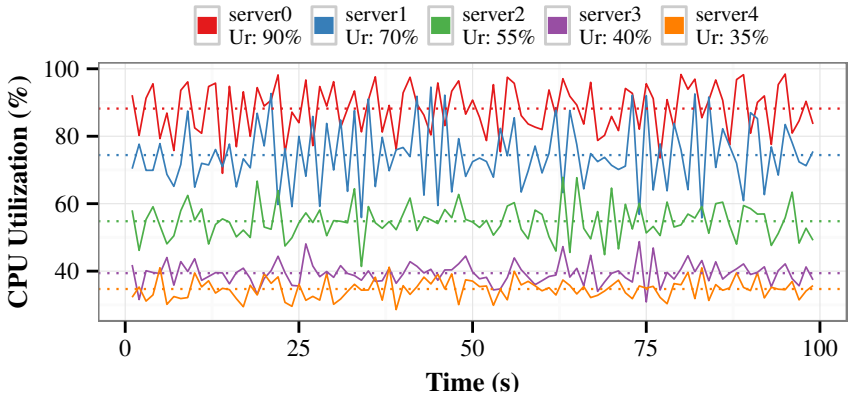


Figure 5.12: Evaluation of the utilization controller mechanism for a fixed load. We see that the controller is able to place the servers at the target utilization

The controller exhibits some variation around the target utilization rate because UCA is based on the number of users, but an user can do different types of operations. For example, an operation of adding a person uses more CPU than a logout operation. However, on the average the utilization rates converge to the targets as shown in the dotted lines.

5.5 Experimental Results

This section shows the evaluation of our power management mechanism in two different scenarios: Constant number of users and Variable Number of Users, where we configured the benchmark to follow the trend line described in Figure 5.4.

We compared our mechanism to the Linux *performance* governor, where the P-state is kept at maximum frequency when the CPU is executing, and to the Linux *ondemand* governor, where the operating system changes the frequency automatically.

We also evaluated a lower-bound for our *Slack Recovery* heuristic, by considering that at idle the CPU would not consume any energy. Therefore, the lower bound is the minimum power consumption on a fictitious environment where the power at 0% of utilization rate is zero. This would be equivalent to turning on/off the idle CPUs in zero time. In practice this might have detrimental effects on the QoS due to the time necessary to switch the CPU back on.

5.5.1 Constant Number of Users

The first set of experiments was to evaluate the *Slack Recovery* implementation under constant number of users. Our 25 Web Server node cluster supports up to 20,000 Olio users when the front-end (PHP + nginx) and the back-end (MySQL) are running on the same machine. We ran the benchmark 9 times changing the number of users on each execution. The different number of users impacts the CPU cluster utilization rates and the power and performance optimization space. Figure 5.13 shows the average power per server for different number of users.

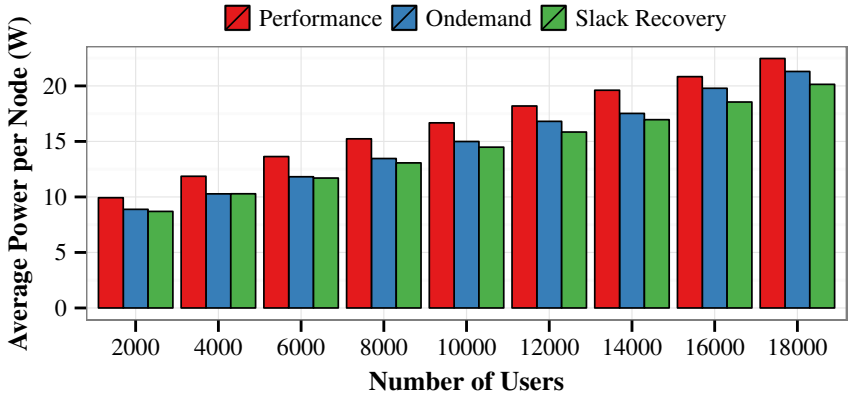


Figure 5.13: Average power per node for different number of Olio benchmark users. The *Slack Recovery* exhibits higher power savings when the number of users is higher.

Our results show that *Slack Recovery* can reduce the power consumption by up to 16% when compared with the *performance* Linux governor, and 6.67% when compared

with the Linux *ondemand* governor. The *Slack Recovery* increased the response times. However, it was still able to meet the benchmark SLA constraints as shown in Figure 5.14.

We observed that the higher power savings are concentrated when the load is higher. *Slack Recovery* trades power for performance keeping a minimum performance threshold. Therefore, it increases the response time for the benchmark (although still meeting the SLA requirement) and reduces the power consumption. When the load is low, there is much idleness on the system, and the idle power dominates the total power consumption.

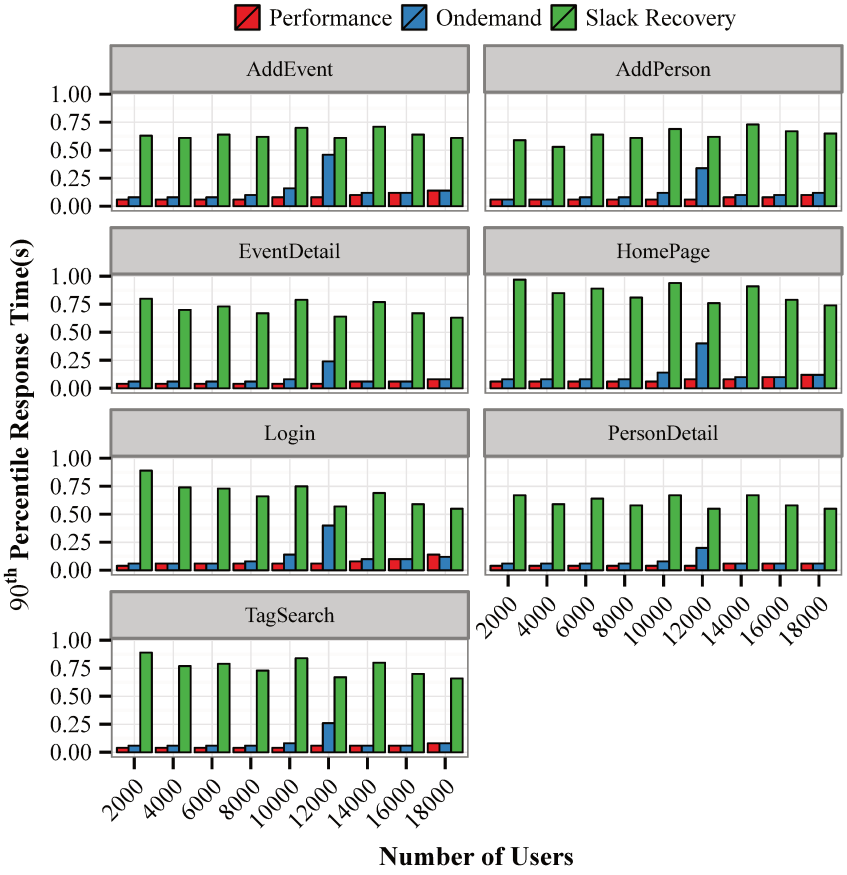


Figure 5.14: SLAs for the Olio benchmark operations. The SLA requirements in 90^{th} percentile response time must be 1 second for HomePage and Login; 2 seconds for EventDetail, PersonDetail, and TagSearch; 3 seconds for AddPerson; and 4 seconds AddEvent. Note that, for all cases, *Slack Recovery* could meet these requirements

5.5.2 Variable Number of Users

The next experiment evaluated the behavior of the algorithm under a variable number of users. The 37-hour load curve, taken from a real Internet cluster hosting the chat room,

shown in Figure 5.4 was shrunk to five hours by calculating the average number of users over a 7-hour time window in order to accelerate the experiments. The maximum number of users was set to 18,000 (about 90% of the maximum capacity). The maximum follows the provision standards that reserve some processing capacity to handle any utilization spikes.

Figure 5.15 illustrates the power consumption of the cluster along the benchmark execution in this scenario. The results agree with those from the constant number of users, where the higher power savings are in the region of higher loads. *Slack Recovery* was able to save 13.1% of the power on average when compared with the Linux *performance* governor, and 5.6% when compared with the Linux *ondemand* governor.

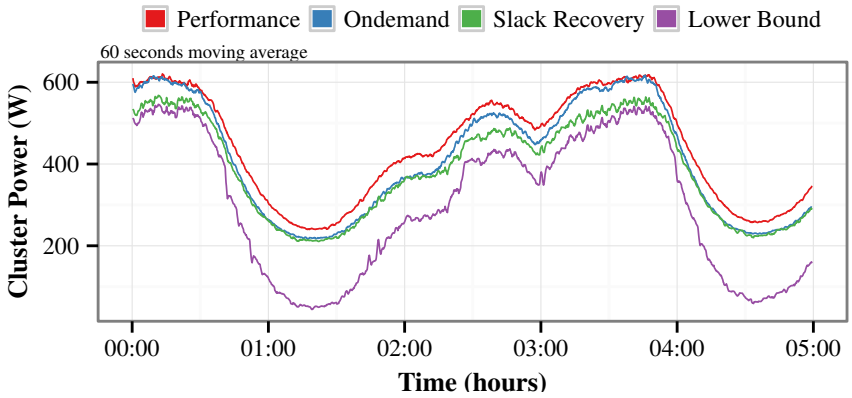


Figure 5.15: Cluster power variation along five hours of benchmark execution. The Lower Bound is the minimum power consumption that would result by *Slack Recovery* if the CPU power when utilization rate is 0% were zero watts

We also want to investigate the power behavior for each node. Figure 5.16 shows the power consumption for the individual nodes along the benchmark execution for the variable number of users.

The first observation is that three nodes (servers 22, 23, and 24) out of 25 are always idle, as we can see in Figure 5.16 . This is related to the assumption that the maximum load for the trend line curve corresponds to about 90% of the maximum processing power capacity of the cluster. The algorithm concentrates the processing power to some nodes while others are placed in idle mode. This fact raises a question about the potential of adding to the algorithm the capacity of powering on/off nodes. We extrapolate our data to determine this value.

The extrapolation is done by setting power to zero instead of 7.3W when the utilization rate of a node is at 0% in a given time window. This is the Lower Bound showed in Figure 5.16, because all idle nodes are considered off.

The Lower Bound corresponds to a reduction in power consumption of 39% when com-

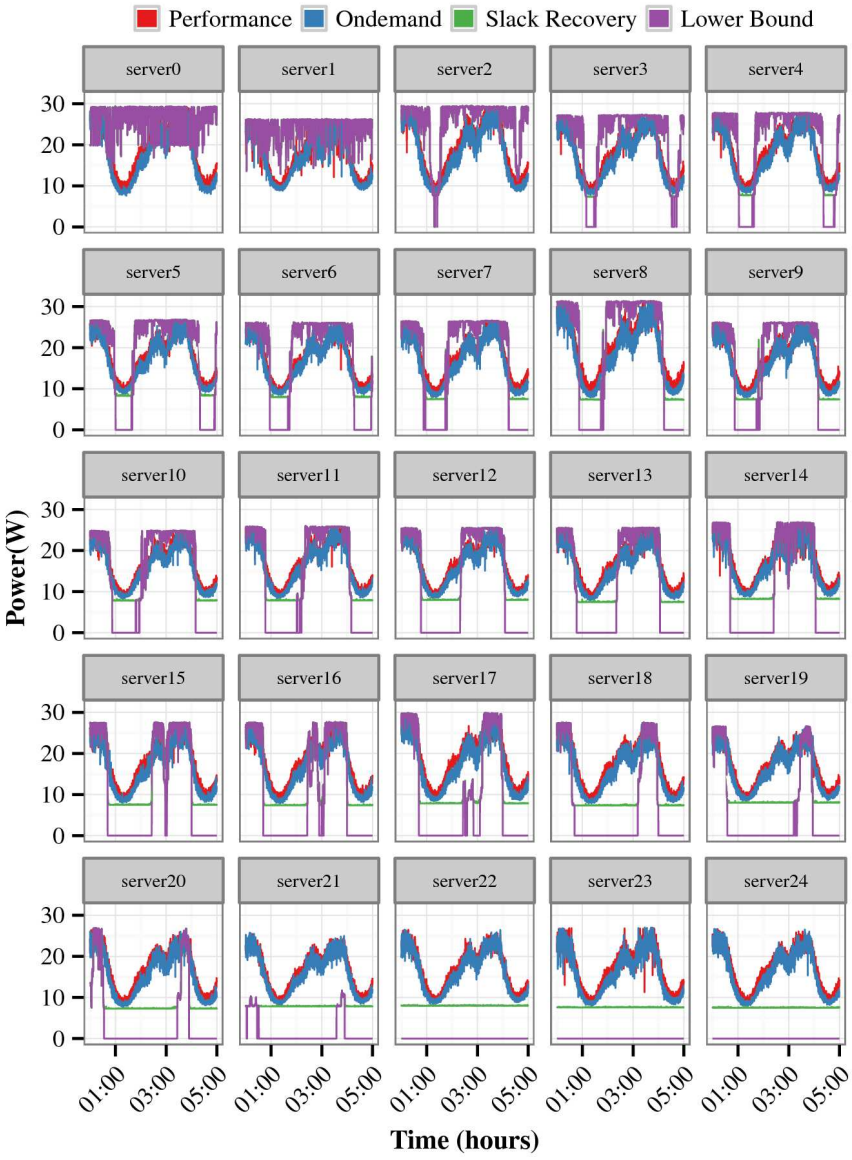


Figure 5.16: Cluster power variation along five hours of benchmark execution for each server. Observe that the last three servers are always idle.

pared to the Linux *performance* governor, to 30% when compared to the Linux *ondemand* governor, and to 23% to *Slack Recovery*.

On our SM15k cluster, a node takes about five minutes to be powered on. For this reason, we did not consider the possibility to power on/off nodes at first sight. However, this possibility is promising and could be feasible if we elaborate our demand predictor by taking into account the time overhead for power on/off the nodes. We leave the evaluation of this approach as a future work.

Chapter 6

Conclusions

The stunning increase in the demand for Internet services in the last few decades is the primary reason for the need of larger data centers, which currently can host several thousands of computers interconnected within a single facility. The corresponding growth in the power consumption of such facilities has become a significant economical and environmental issue. Thus, power consumption has become the main design constraint of modern data centers.

This thesis presented empirical models for estimating the power consumed by Web Servers. The models were validated with SPECweb2009, a state-of-the-art Web benchmark which characterizes different Web applications and contains both static and dynamic content. We modeled two Web Servers having different processors and configurations.

The Web Server power measurements were done by a custom-made infrastructure that enables power breakdown for the individual system components. We found that the processor is the dominant component in the server's power consumption, corroborating the results presented by others [4]. This result reinforces the fact that the processor should be the main target when devising power-aware optimization algorithms for Web Servers.

We also presented a novel approach for modeling full system power based on CFS algorithm and k-means clustering. This new approach softened non-linear effects among system measurements and system power improving model accuracy. Our models considered the different frequency and voltage operating points (*P-states*) of the processor and took into account all major components of the server, such as processor, disk, network, memory, and other motherboard components. Our best full-system power models displayed an average absolute error of 1.92% for the Intel i7 server and 1.46% for the AMD Opteron as compared to actual measurements, and 90^{th} percentile for the absolute percent error equal to 2.66% for the Intel i7 and 2.08% for the AMD Opteron.

We also implemented an adaptive power management system for a Web Server cluster running on a production environment. The cluster is composed by state-of-the art high

density and power efficient architecture nodes, the AMD SeaMicro SM15k cluster.

Our system was based on the *Slack Recovery* [6] heuristic, which relies on the concept of Virtual Power States (VPS), and was previously evaluated on a simulation environment. In order to bring it to a production cluster, we needed to show how VPSs could be implemented on practice. We do so by presenting new techniques to predict future demanded performance and a *Utilization Control Agent* (UCA). Our experimental evaluation shows that the UCA is capable of maintaining the cluster at the desired utilization rates.

Our power management system was compared to two Linux power consumption governors: *performance* and *ondemand*. The experiments were conducted using Olio, a Web 2.0 web-based social calendar extracted from the CloudSuite Web Server benchmark [24].

We showed that our *Slack Recovery*-based system can save up to 16% of the power consumed in the cluster when compared with the Linux *performance* governor, and up to 6.67% when compared with Linux *ondemand* governor. Finally, we evaluated the potential for power savings that could be brought by powering on/off cluster nodes, an alternative that may be promising but we decided not to include in this version of our system due to the penalty of turning on/off SM15k cluster nodes. We plan to include this feature when evaluating our power management system in future works.

6.1 Future Work

In Chapter 3, we presented Web Server power modeling techniques. We collected system level performance statistics and used them as proxies to system power. In the experiments, we correlated the power and performance measured on a time-window of one second. We used this approach to reduce possible shifts among the measurements. This approach was sufficient for Web Server power models. However, the approach might not work for modeling applications that need finer granularity level. Therefore, a possible future work is to study techniques to synchronize power and performance measurements. On way to do this is to put the machine on idle for a period of time, which will reduce the power consumption and the performance statistic rates. When the application starts, spikes will be observed on the timeline graphs. Therefore, by using statistical methods an experimenter could try to identify this marks and synchronize the measurements.

In Chapter 5, we presented a method to optimize power consumption globally on a Web Server cluster. We createad the infrastructure and showed that the method was able to save some energy. We also presented a lower-bound for the *Slack Recovery* heuristic if we consider the idle power as idle, which is the same to power-on/off machines in zero time. Therefore, the most straightforward future work would be to study techiniques to power-on/off machines.

Powering-on/off machines will need some changes on the current infrastructure implementation. The implementation assumes that the nodes were already powered up. Moreover, it synchronizes the nodes by using broadcast messages only once. Periodically synchronization messages should be used to implement power-on/off capability.

The predictor also needs modifications to allow the capability of powering-on/off nodes. Performance prediction is done for the next time window. In our experiments, we used a time window of 20 seconds. A SM15k cluster node takes about 5 minutes to be powered up. Therefore, adding a power on/off capability to the algorithm could be done by spliting the prediction in two phases, a long term and a short term, so that, we can account the time to power on/off the machine.

Chapter 5 also showed that by using our heuristic algorithm, the SLA increased by 8 times when compared to other algorithms. Although the bechmark SLA was met, this penalty could be unacceptable for some clients. Therefore, it is necessary to investigate methods to improve the SLA. One possible aproach would be to add a more intelligent predictor that takes into account possible load variation. So as future work, it is possible to investigate different predictions methods and their impacts on SLA and power consumption.

Finally, we can use the RAPL interfaces to implement the performance optimization under a power cap. The *Slack Recovery* is able to optimize power under a performance threshold and also optimize performance under a power cap. Power capping is for free when using RAPL interfaces, since it can be readily done by writing the target power on RAPL registers [29,59]. Therefore, the by combining RAPL interfaces, the *Slack Recovery*, and the monitoring cluster infrastructure, the performance optimization problem under a power cap could be easily implemented.

Appendix A

Acronyms

This is a list of the acronyms used in this thesis.

- CFS: Correlation-based Feature Selection
- CDF: Cumulative Distribution Function
- EAC: Energy-Adaptive Computing
- GPM: Global Power Model
- ILP: Integer Linear Programming
- pSPM: P-State-based Power Model
- PMC: Performance Monitoring Counter
- PUE: Power Usage Effectiveness
- PWSPM: Pruned Web Server Power Model
- RU: Rack Unit
- RAPL: Running Average Power Limit
- SLA: Service Level Agreement
- SIMO: Single-Input Multiple-Output
- SISO: Single-Input Single-Output
- VPS: Virtual Power States
- WSPM: Web Server Power Model

Bibliography

[1] Zahra Abbasi, Georgios Varsamopoulos, and Sandeep K. S. Gupta. Tacoma: Server and workload management in internet data centers considering cooling-computing power trade-off and energy proportionality. *ACM Transactions on Architecture Code Optimization*, 9(2):11:1–11:37, June 2012.

[2] Advanced Configuration and Power Interface Specification. online, 2011. `http://www.acpi.info/spec.htm`, Accessed on 29th November 2011.

[3] AMD, Sunnyvale, CA, USA. *SeaMicro SM15000 Fabric Compute Systems*, 2012.

[4] L. A. Barroso and U. Holzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12):33–37, 2007.

[5] Frank Bellosa. The benefits of event–driven energy accounting in power-sensitive Systems. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, 2000.

[6] Reinaldo A. Bergamaschi, Leonardo Piga, Sandro Rigo, Rodolfo Azevedo, and Guido Araujo. Data center power and performance optimization through global selection of p-states and utilization rates. *Sustainable Computing: Informatics and Systems*, 2(4):198–208, 2012.

[7] Luciano Bertini, Julius C. B. Leite, and Daniel Mossé. Power and performance control of soft real-time web server clusters. *Inf. Process. Lett.*, 110(17):767–773, August 2010.

[8] Luciano Bertini, Julius C. B. Leite, and Daniel Mossé. Power optimization for dynamic configuration in heterogeneous web server clusters. *J. Syst. Softw.*, 83(4):585–598, Apr 2010.

[9] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. Decomposable and responsive power models for multicore processors us-

ing performance counters. In *ICS '10: Proceedings of the 24th ACM International Conference on Supercomputing*, 2010.

[10] R. Bianchini and R. Rajamony. Power and energy management for server systems. *Computer*, 37(11):68–76, 2004.

[11] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. Power aware computing. chapter The case for power management in web servers, pages 261–289. Kluwer Academic Publishers, 2002.

[12] M. Breternitz, K. Lowery, A. Charnoff, P. Kaminski, and L. Piga. Cloud workload analysis with swat. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, pages 92 –99, oct. 2012.

[13] Intel Technology Brief. Balancing performance and power consumption. `http://download.intel.com/embedded/322286.pdf`.

[14] Dominik Brodowski. CPU frequency and voltage scaling code in the Linux(TM) kernel. Technical report, kernel.org, May 2013.

[15] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *ICS '03: Proceedings of the 17th annual international conference on Supercomputing*, 2003.

[16] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 103–116, 2001.

[17] Xi Chen, Chi Xu, Robert P. Dick, and Zhuoqing Morley Mao. Performance and power modeling in a multi-programmed multi-core environment. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 813–818, 2010.

[18] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing server energy and operational costs in hosting centers. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '05, pages 303–314, 2005.

[19] R. Cochran, C. Hankendi, A. Coskun, and S. Reda. Pack & cap: adaptive dvfs and thread packing under power caps. In *44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.

[20] Gilberto Contreras and Margaret Martonosi. Power prediction for Intel XScale®processors using performance monitoring unit events. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, 2005.

[21] E. N. Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Proceedings of the 2nd international conference on Power-aware computer systems*, PACS'02, pages 179–197, 2003.

[22] Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 8–8, 2003.

[23] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, 2007.

[24] Michael Ferdman, Almutaz Adileh, Yusuf Onur Koçberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Ilknur Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*, pages 37–48, 2012.

[25] D. Filani, J. He, S. Gao, M. Rajappa, A. Kumar, P. Shah, and R. Nagappan. Dynamic data center power management trends, issues, and solutions. *Intel Technology Journal*, 12:59–67, 2008.

[26] GTKWave. online. `http://gtkwave.sourceforge.net/` [Accessed on 16 February 2012].

[27] Mark A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato, 1999.

[28] National Instruments. Bus-Powered M Series Multifunction DAQ for USB - 16-Bit, up to 400 kS/s, up to 32 Analog Inputs, Isolation Data Sheet, 2009.

[29] Intel, Santa Clara, CA, USA. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2*, June 2013.

[30] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39 2006)*, pages 347–358, 2006.

[31] Canturk Isci and Margaret Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[32] Russ Joseph and Margaret Martonosi. Run-time power estimation in high performance microprocessors. In *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, 2001.

[33] Krishna Kant, Muthukumar Murugan, and David H. C. Du. Enhancing data center sustainability through energy-adaptive computing. *J. Emerg. Technol. Comput. Syst.*, 8(4):33:1–33:20, November 2012.

[34] DAVID J. KETCHEN and CHRISTOPHER L. SHOOK. The application of cluster analysis in strategic management research: An analysis and critique. *Strategic Management Journal*, 17(6):441–458, 1996.

[35] Ricardo Koller, Akshat Verma, and Anindya Neogi. Wattapp: an application aware power meter for shared data centers. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 31–40, 2010.

[36] Jonathan G. Koomey. Estimating total power consumption by servers in the U.S. and the world. Technical report, Stanford University, 2007.

[37] Jonathan G. Koomey. Growth in data center electricity use 2005 to 2010. Technical report, Stanford University, 2011.

[38] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. In *Proceedings of the 2008 International Conference on Autonomic Computing*, ICAC '08, pages 3–12, 2008.

[39] J.H. Laros, K.T. Pedretti, S.M. Kelly, J.P. Vandyke, K.B. Ferreira, C.T. Vaughan, and M. Swan. Topics on measuring real power usage on high performance computing platforms. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009.

[40] LEM Components. Current transducer lts 25-NP data sheet, 2008.

[41] Adam Wade Lewis, Nian-Feng Tzeng, and Soumik Ghosh. Runtime energy consumption estimation for server workloads based on chaotic time-series approximation. *ACM Trans. Archit. Code Optim.*, 9(3):15:1–15:26, October 2012.

[42] libpfm4 documentation. online, 2013. `http://perfmon2.sourceforge.net/docs_v4.html`, Accessed on 04th July 2013.

[43] Linux Kernel Organization. Block layer statistics – Linux Documentation Project, 2010.

[44] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 2006.

[45] Chris D. Lucer and Chakravarthy Akella. Power Profiling for Embedded Applications. White paper, 2009.

[46] Christopher Malone and Christian Belady. EAC & PUE: metrics to characterize IT equipment & data center energy use. In *Digital Power Forum*, 2006.

[47] Robert McGill, John W. Tukey, and Wayne A. Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.

[48] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 319–330, 2011.

[49] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and Thermal Management in the Intel Core Duo Processor. *Intel Technology Journal*, 10(2), may 2006.

[50] L. Piga, R. Bergamaschi, F. Klein, R. Azevedo, and S. Rigo. Empirical web server power modeling and characterization. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, page 75, nov. 2011.

[51] Leonardo Piga, Reinaldo Bergamaschi, Rodolfo Azevedo, and Sandro Rigo. Power Measuring Infrastructure for Computing Systems. Technical report, Institute of Computing, University of Campinas, 2011.

[52] Leonardo Piga, Gabriel F.T. Gomes, Rafael Auler, Bruno Rosa, Sandro Rigo, and Edson Borin. Assessing computer performance with SToCS. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, 2013.

[53] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '03, pages 111–122, 2003.

[54] Karthick Rajamani, Freeman Rawson, Malcolm Ware, Heather Hanson, John Carter, Todd Rosedahl, Andrew Geissler, Guillermo Silva, and Hong Hua. Power-performance management on an IBM POWER7 server. In *ISLPED '10: Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, 2010.

[55] Red Hat Inc. Performance counters for linux, 2010.

[56] S. Rivoire, M.A. Shah, P. Ranganatban, C. Kozyrakis, and J. Meza. Models and metrics to enable energy-efficiency optimizations. *Computer*, 40(12):39 –48, dec. 2007.

[57] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *HotPower'08*, 2008.

[58] Suzanne Marion Rivoire. *Models and Metrics for Energy-Efficient Computer Systems*. PhD thesis, Department of Electrical Engineering of Stanford University, 2008.

[59] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *Micro, IEEE*, 32(2):20–27, 2012.

[60] Efraim Rotem, Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eli Weissmann. Power management architecture of the 2nd generation intel® core™ microarchitecture, formerly codenamed sandy bridge. In *Hot Chips 23*, 2011.

[61] David Schneider. Under the hood at google and facebook. online, 2011. `http://spectrum.ieee.org/telecom/internet/under-the-hood-at-google-and-facebook`, Accessed on 20th August 2013.

[62] Greg Schulz. *The Green and Virtual Data Center*. Auerbach Publications, Boston, MA, USA, 1st edition, 2009.

[63] Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, and Xiao Zhang. Power and energy containers for multicore servers. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 403–404, 2012.

[64] Standard performance evaluation corporation (SPEC). online, 2009. `http://www.spec.org/web2009`, Accessed on 17 March 2009.

[65] Willy Tarreau. HAProxy Configuration Manual version 1.5. Technical report, HAProxy, June 2013.

[66] Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 29–40, 2010.

[67] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, 2003.