Universidade Federal Fluminense

Raphael Pereira de Oliveira Guerra

Improving Response Time and Energy Efficiency in Server Clusters

NITERÓI

2006

Universidade Federal Fluminense

Raphael Pereira de Oliveira Guerra

Improving Response Time and Energy Efficiency in Server Clusters

A dissertation submitted to the Post-Graduate Program in Computing of Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master in Computing. Field: Parallel and Distributed Computing.

Supervisor:
Prof. Julius C. B. Leite, Ph.D.

NITERÓI

2006

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

G934 Guerra, Raphael Pereira de Oliveira.
Improving response time and energy efficiency in server clusters / Raphael Pereira de Oliveira Guerra. - Niterói, RJ : [s.n.], 2006.
90 f.

Orientador: Julius C. B. Leite.
Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2006.

1. Cluster de servidores. 2. Energia - Consumo. 3. Sistemas em tempo real. 4. Dinâmica de voltagem - Regulagem. 5. Qualidade de Serviço. I. Título.

CDD 004.35

Improving Response Time and Energy Efficiency in Server Clusters

Raphael Pereira de Oliveira Guerra

A dissertation submitted to the Post-Graduate Program in Computing of Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master in Computing. Field: Parallel and Distributed Computing.

Approved by Dissertation Examining Committee:

Prof. Julius C. B. Leite, Ph.D. / IC-UFF (Supervisor)

Prof. Orlando Loques, Ph.D. / IC-UFF

Prof. Gerhard Fohler, Dipl.-Ing.Dr. / Technische Universität Kaiserslautern, Germany

Niterói, September 2006.

To my grandparents and friends.

Acknowledgements

This is a very important moment in my life. The period I was writing this dissertation was a very tough time, but I knew it would repay my effort, and that is the thought that kept me on my way all this time. In these moments when we reach our goals it is like time freezes and we just stare high above all battles we have been through, the victories, the defeats and the lessons we take from everything.

For this victory, I would like to special thank my grandparents, who raised me up and made it possible for me to reach this point in my life. However, through a lifetime people come and go and make the difference. So I would like to specially thank Professor Julius Leite and Professor Gerhard Fohler, who trusted in me and my potential and gave me the opportunity to succeed in my career. Without their influence I would have quit university long ago.

For his prime importance to the work conducted here, I can not forget to thank Luciano Bertini. He is a PhD student at Universidade Federal Fluminense, whose contribution propelled this work.

Friends are also very important in our lives, because they are always there with us sharing the important moments. My roommates, my classmates, people I hang out with, I thank all of them. However, I would like to specially thank two: Felipe Carone and Cristiane Ferreira. I thank Cristiane because she was a very important person during my writing period, coming to my laboratory every day to support me and give me strength to hold on. She was there when I needed someone the most. And I thank Felipe for the wise advices, for sharing good and bad moments and specially for turning bad moments into good moments.

Thank you all.

Resumo

O desenvolvimento de servidores *web* energeticamente eficientes requer não só o estudo de políticas de despacho a serem aplicadas pelo nó central, como também o uso de técnicas de *hardware* que permitam um melhor uso dos recursos. Contudo, essa eficiência não pode ser obtida em detrimento do atendimento aos prazos de execução das requisições. O trabalho apresentado aqui descreve uma técnica que tenta balancear consumo de energia e tempos de resposta adequados para aplicações de tempo real não críticas em *clusters* de servidores. A validação deste trabalho foi feita através de sua implementado em um simulador.

Abstract

The development of energy-efficient server clusters requires the study of different request dispatch policies applied by the central access point of the cluster, and the application of hardware techniques that provide for the best usage of server resources. However, energy efficiency should not be attained at the expense of a poor quality of service. The work proposed here describes a technique that tries to balance energy consumption and adequate application response times in heterogeneous server clusters. This proposal was evaluated through simulation and showed results that outperform state-of-the-art techniques.

Keywords

1. Energy-Efficiency;
2. Server Clusters;
3. Quality of Service;
4. Soft Real-Time Systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Power management (PM) has been extensively studied for applications running on battery-powered devices. These devices have gone through an ever growing processing power increase in order to deal with the demands of the even more sophisticated applications designed to run on them. As long as these devices are not connected to the electrical power grid, energy becomes a critical resource and the trade-off between processing power and energy saving comes to be a main concern. This trend represents the original concept that PM had to address, that is, the device availability, by maximizing the battery lifetime.

On the other hand, performance and reliability have traditionally been the most important design and evaluation factors on desktop machines. As they have continuous energy supply, higher performance has been achieved at the expense of great energy consumption. The problem is that along with the energy consumption increase comes a higher energy dissipation in the form of heat. A way to achieve performance improvement on a single processor is by making its components smaller, so that more of them can be inserted onto the die. Because transistors using today's technology are so small, even when they are supposed to be in the off state, infinitesimal currents still leak through them. This leakage warms them constantly, and with the extra heat generated when transistors switch on or off, it produces a microfurnace on a chip [26].

In the quest for performance improvement, there is a growing industry trend to "outsource" computing services to large server clusters. Typical server clusters deploy hundreds or thousands of computers (like Google servers [6]), usually located in the same room. These densely packed systems demand a great amount of energy per unit of volume, thus overheating it. As a result, the need for a complex and expensive cooling infrastructure to ensure the proper operation of the servers becomes even more challenging: fans driving the cooling system may consume up to 50 % of the total system power in some commercial servers, and manufacturers are facing the problem of building powerful

systems without introducing additional techniques such as liquid cooling [22]. The power issue came to the extent of worrying the american government to the point of having a energy-efficient server incentive law approved by the Congress [37].

With the growing of the Internet, cluster systems have been widely designed for file distribution, web servers and database transactions aiming to handle peak loads conditions, thus providing an acceptable quality of service (QoS) to the final users even in the worst case. This practice, however, leads to poor system's infrastructure resource utilization since such systems are typically underutilized. In fact, their loads often vary significantly depending on the time of the day or other external factors. Therefore the average processor use of such systems may be even less than 50% with respect to their peak capacity [31].

The report [1] identified that if it was possible to have the clusters "rightsized" to incur only the infrastructure cost that was actually required at a given time, a potential reduction of 60% on the total cost of the system could be attained. Nonetheless, not all resources that compose the cluster infrastructure can be employed in a modular and scalable way (e.g., physical room, machines, network and cooling equipment, ...). Even so, it is shown that when modular and scalable technology is used to the extent which is currently practical, approximately 50% of the theoretical savings from rightsizing can be obtained. Electricity, cooling equipment and power equipment altogether comprise 44% of a data center expense. These expenses can be reduced by PM. Recent improvements on hardware to support energy control make this feature scalable, and taking advantage of the long inactive periods due to high utilization fluctuation on Web servers provides a potential economy on server overall expenses. Besides, excess power consumption not only hurts the operator economically, but it also limits the number of servers per unit of volume (in the machine room) due to heat dissipation considerations, thus limiting the cluster operation.

The cooling system is of major importance particularly because the microprocessor's overheat leads to intermittent system failures and crashes. It strongly impacts on system reliability and performance [12], which have traditionally been considered the most important design and evaluation factors on desktop machines. As a result, researchers have moved their focus from cooling infrastructures to PM schemes these days. This approach exploits the underutilization condition experienced by the server clusters most of the time and it is capable of addressing the following critical issues:

- **Energy consumption** – It reduces the energy consumption of the server cluster, thus

reducing the energy leakage as well. Furthermore, less leakage implies less heat dissipation, and hence, less demanding from the cooling system. As a result, PM is capable of reducing energy consumption in both cooling system and server cluster operations.

- **Reliability** – As it reduces the heat dissipation, there will be less failures and crashes.
- **Costs** – Money is saved both in energy consumption (energy is not free), cooling systems and hardware replacement due to crashes.

There is a wide variety of PM techniques. These techniques, primarily developed for mobile and energy-restricted devices, basically comprise turning off idle components like the monitor, the network interface and so forth. A prospect alternative that has been exploited lately is to set the CPU to operate in lower frequencies. Although operating the CPU in a lower frequency implies longer processing time, the vast majority of microprocessors today use CMOS logic circuits, which has a voltage-dependent maximum operating frequency. So when used at a reduced frequency, the processor can operate at a lower supply voltage. Since the energy dissipated per cycle with CMOS circuitry scales quadratically to the supply voltage ($E \propto V^2$ – already taking into account the longer processing), a potential energy saving approach arises. Therefore, PM is equally important for server environments, where high performance and reliability have traditionally been the most important design and evaluation factors. This position breaks with tradition and argues for considering energy consumption on the same footing. The PM topic will be further discussed in Chapter 3.

However, the Web environment is QoS-aware and CPU operating frequency management cannot be done carelessly. Online banking, reservations, stock trading, product merchandising, to name a few, are services that are being offered via Web. As this trend continues into the 21^{st} century, QoS considerations are expected to gain tremendous importance. For instance, dropped connections and slow responses from a Web site frustrate users and result in lost revenues for the Web site operator. Performance is a fundamental issue in the Web environment because final users do not want to wait indefinitely for a response.

Many works in the energy saving research field do not take into account the QoS feature of Web servers. Their target is usually to save as much energy as possible by running the server in a totally saturated configuration (using only the necessary computational power required to keep up with the incoming workload without cumulating requests in

the queue). In fact, it leads to a constant worst case QoS condition. In this work, we specifically target the research issues in incorporating PM in QoS-aware Web servers and the systems issues that arise in designing and implementing such Web servers with energy saving schemes.

Real world Web server clusters are a really complex structure. They must handle a wide variety of requests, like *static* and *dynamic* content, located in a single server or spread out among any number of machines and so on. The server cluster is most likely to be composed of heterogeneous machines (both in hardware configuration and information). A new category of machines named blade servers [41] arise as a novel solution for server cluster construction, as long as they are optimized for consuming a lot less power and take up a lot less room. Another relevant issue is the way the pieces can be gathered together, namely the cluster architecture. These structural matters and their features will be explained in more detail in Chapter 2.

By now, it is only relevant to mention that in this work we assume a 2-layer content-blind architecture: the server layer, comprised of the machines that will handle the users requests, called servers; and the front-end layer, constituted of a single machine, called front-end, which is responsible for balancing the load among the machines comprising the server layer. This architecture is called blind because the front-end has no access to the request's content when performing the dispatching policy. Therefore, all servers must contain the same information (in other words, should be able to handle any kind of requests).

Despite the simplicity, our scheme represents real life. Most Web server clusters nowadays use their servers to replicate data for QoS purposes. Highly accessed Web sites with *static content*, such as pictures or multimedia files, always replicate their data among several servers to avoid congestion. Therefore, the assumption of a server cluster where all servers contain the same data covers real life situations. Our method aims to provide QoS-aware PM in the environment described before based on queuing theory. We point out that, in spite of adopting the Web server context as our model, our methodology and the algorithms proposed apply to any kind of cluster. The next section describes how this dissertation is structured.

1.1 Structure

This work is structured in seven chapters, including this introduction. The next two chapters present the background of this work, covering some important concepts about Web cluster architectures and PM policies. The following two chapters account for the system model description and for a discussion of the obtained results. Finally, the last two chapters outline the related works and their contributions, point out future works and draw the conclusions. A brief description of the content of each chapter is shown below:

- **Chapter 1** – Presents an overview of the work and its motivation. It corresponds to this current chapter.
- **Chapter 2** – Presents the state-of-art in server cluster architectures. It covers the system's infrastructure and operation.
- **Chapter 3** – Presents the importance of PM relatively to performance and reliability issues and gives insights into some usual PM techniques, ranging from device level to cluster-wide application level approaches.
- **Chapter 4** – Describes our model and the assumptions we made, defines the problem to be dealt with and describes the given solution. Some implementation details are discussed.
- **Chapter 5** – Details the experiments performed and the obtained results.
- **Chapters 6** – Discusses the related works and other major contributions in the energy saving research field.
- **Chapters 7** – Discusses the future works and draws the conclusions.

As a final remark, it should be pointed out that as a result of this work two scientific papers were produced. The first one [17] was presented at the Workshop de Tempo Real (WTR, Curitiba, Brazil, 2006) and the other one [18] at the Conferencia Latinoamericana de Informática (CLEI, Santiago, Chile, 2006).

Chapter 2

Web Server Architectures

The World Wide Web became widespread in the mid 1990's, but its beginning can actually be traced back to 1980 when Tim Berners-Lee, a software consultant at CERN (Organisation Européenne pour la Recherche Nucléaire, in english European Organization for Nuclear Research) wrote a program for his personal use, called Enquire, which he called a "memory substitute" to help to keep connections between various people and projects at the laboratories. However, researchers outside CERN contributing to the CERN researches also wanted to share documents, but using different kinds of computers and software because it would be unfeasible to make their work conform to the CERN system. So Berners-Lee thought, "it would be so much easier if everybody asking me questions all the time could just read my database, and it would be so much nicer if I could find out what these guys are doing by jumping into a similar database of information for them" [42]. In 1990, he wrote the Hypertext Transfer Protocol (HTTP), the language computers would use to communicate hypertext documents over the Internet and designed a scheme to give documents addresses on the Internet. Berners-Lee called this address a Universal Resource Identifier (URI), which is now usually known as Uniform Resource Locator (URL). By the end of the year he had also written a client program (browser) to retrieve and view hypertext documents. He called this client "World Wide Web". Hypertext pages were formatted using the Hypertext Markup Language (HTML) that Berners-Lee had written.

In this environment, a user issues one request at a time for a *Web page*. There are many types of Web pages, but in the old days of the Web, a *Web page* was a multipart document consisting of a collection of *static objects* (a *static object* is simply a file in a specific format, e.g. JPEG). An object could be provided by the first contacted *Web site* or even by another site. Back then, a *Web page* consisted of a base HTML file describing the page layout and a number of *static objects* referenced by the base HTML file using URLs that indicate where

they were hosted. A URL has two main components: the symbolic name of the server that houses the object (e.g., *www.uff.br*) and the object's path name (e.g., */pub/index.html*). To retrieve all the *Web objects* composing a *Web page*, a *browser* (client program that interacts with the *Web server* and displays the *Web page* to the end user) issues multiple requests to the *Web server*. We refer to a *Web transaction* as the complete interaction that starts when the user sends a request to a Web site and ends when his client receives the last object related to the requested URL. A *session* is a sequence of Web transactions issued by the same user during an entire visit to a *Web site*. The HTTP is the protocol that rules this client-server interaction.

As we can see, the Web has been primarily designed for information sharing only. However, nowadays the Web is changing from a simple communication and browsing infrastructure for getting static information to a complex medium for conducting personal and commercial transactions that require dynamic computation and secure communications with multiple servers through middleware and application software. Furthermore, the proliferation of heterogeneous client devices, the need of data personalization, client authentication, and system security of corporate data centers and e-commerce sites place additional computational load on Web servers, incurring a significant performance penalty.

Because of the complexity of the Web infrastructure, performance problems may arise in many points during an interaction with Web-based systems. For instance, they may occur in the network due to congested Internet routers or poor quality network links, as well as at the Web and database server, either because of underprovisioned capacity or unexpected surge of requests. Although, in recent years, both network and server capacity have improved and new architectural solutions have been deployed, network bandwidth is increasing about twice as faster than the server capacity and the increased percentage of dynamic content of Web-based systems makes the server side likely to be the main future bottleneck.

In order to handle this ever increasing server capacity demanding of Web applications, Web-site administrators constantly face the need to increase server capacity. The server ability to support large numbers of accesses and resources while still providing adequate performance is called scalability. Scalable Web systems can be divided into the following categories:

- **Hardware scale-up** – It consists of upgrading the system by adding more powerful resources, like CPU, disk, additional whole machines and so on.

- **Software scale-up** – It consists of upgrading the system at software or infrastructural level. This category involves improving the operating system, building more efficient Web servers, implementing different scheduling policies and cluster design.

The number of on-line users is growing at about 90% per annum [9], thus making the single server approach unfeasible. The solution to keep up with ever increasing request load and provide scalable Web-based services is to deploy a distributed Web system composed by multiple server nodes that can also exploit scale-up advancements. The server cluster, like any other distributed system, must be designed to be scalable (additional nodes cannot decrease the system performance), reliable (node failures cannot compromise the whole system) and to provide performance enhancement. For these reasons, a lot of effort has been spent in new distributed server architectures. The paper [9] is a survey on distributed Web servers that extensively discusses these system's infrastructure and implementation aspects. As our goal is to attain energy saving in such servers, we considered it important to present to the reader some relevant features about their design. So, the rest of this chapter will be dedicated to point out the aspects exploited on this survey that allow for a better understanding of our work and to discuss them in further detail. Any additional information that the reader might miss here can be found in the original document, which presents a broader view on it.

2.1 Locally Distributed Web System Architectures

Servers based on clusters of workstations are the most popular configuration used to meet the growing traffic demands imposed by the World Wide Web. It has been observed that Web servers contribute to about 40% of the overall delay of Web transactions and this delay is likely to grow with the increasing use of dynamic Web contents. In view of this, distributed Web servers are envisioned to answer most of the requirements in a cost-effective manner. A cluster of servers, arranged to act as a single unit, provides incremental scalability, as it has the ability to grow gradually with demand, reliability, as it is not vulnerable to single node failures, and performance improvement, as all nodes join their computational power to work as a single one.

Depending on the way the servers are organized, they can be classified in: cluster-based Web systems, virtual Web clusters and distributed Web clusters. These architectures are mainly composed by the *server nodes*, which are responsible for serving the

requests, and the back-end nodes, which are responsible for providing services to the server nodes, like databases. These architectures differentiate from one another in the way that the server nodes that compose the cluster are accessed by the outside world. The first one, the cluster-based approach, is the only one among them that uses a central entity that coordinates any action in the cluster. Both centralization and decentralization have their advantages and drawbacks. In short, we can say that the former is able to perform more efficiently while the latter removes the single point of failure. In other words, the former is better from a performance standpoint and the latter is better from a reliability standpoint. As long as PM actions must be carried out by some entity, the cluster-based architecture seems to be most suitable to implement PM and we chose it, since it already has a central entity to be exploited, thus adding another advantage to it. These architectures will be discussed in more detail in the following sections, but thereafter only aspects pertaining to the cluster-based approach will be covered, as this is the relevant design to our work.

2.1.1 Cluster-Based Web Systems

A cluster-based Web server is a Web system architecture that consists of multiple server nodes distributed on a local area and interconnected through a high-speed network. The Web server presents a single system view to the clients through a *front-end*, which distributes the requests among the nodes. Each node in the cluster has its own disk and a complete operating system. Cluster nodes work collectively as a single computing resource.

Although a Web cluster may consist of tens of nodes, it is publicized with one site name (e.g., *www.uff.br*) and one virtual IP (VIP) address (e.g., 200.20.15.238). Thus, the DNS server performs a one-to-one mapping by translating the site name into the VIP address, which corresponds to the IP address of a dedicated front-end node(s). It interfaces the rest of the Web-cluster nodes with the Internet, thus making the distributed nature of the site architecture completely transparent to both the user and the client application. The front-end node receives all inbound packets that clients send to the VIP address and routes them to some Web-server node. In such a way, it acts as the centralized dispatcher of a distributed system with fine-grained control on client requests assignments. Figure 2.1 depicts this architecture.

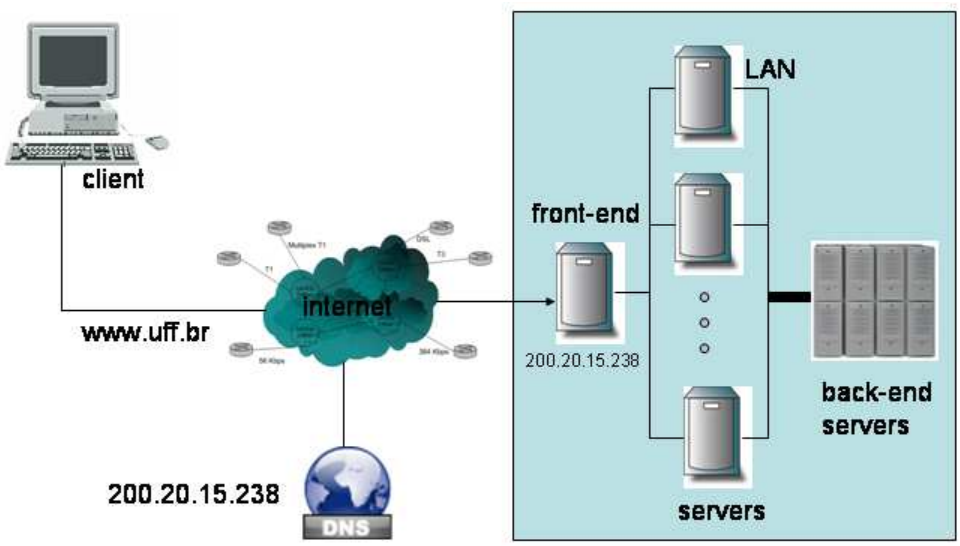


Figure 2.1: Cluster-based Web server architecture.

2.1.2 Virtual Web Clusters

The architecture of a virtual Web cluster is based on a fully distributed system design that does not use a front-end. Similarly to the previously described Web cluster architecture, a virtual Web cluster presents a single system image to the outside world through the use of a single VIP address. The main difference is that this address is not assigned to a centralized front-end node that receives all incoming client requests and distributes them to a target server. In a virtual Web cluster, the VIP address is shared by all the server nodes in the cluster so that each node receives all inbound packets and filters them to decide whether to accept or discard them. It is worth to note that this system removes the single point of failure and the potential bottleneck represented by the front-end. See Figure 2.2.

2.1.3 Distributed Web Clusters

A distributed Web system consists of locally distributed server nodes, whose multiple IP addresses may be visible to client applications (see Figure 2.3). An authoritative DNS is included into the system box because this component plays a key role in request routing for distributed Web systems: it maps the site name directly to the IP address of the server responsible for handling the request, thus making different mappings every time it is consulted in order to distribute the workload. As long as the client request assignment to a target Web server is carried out during the address resolution phase (look-up phase)

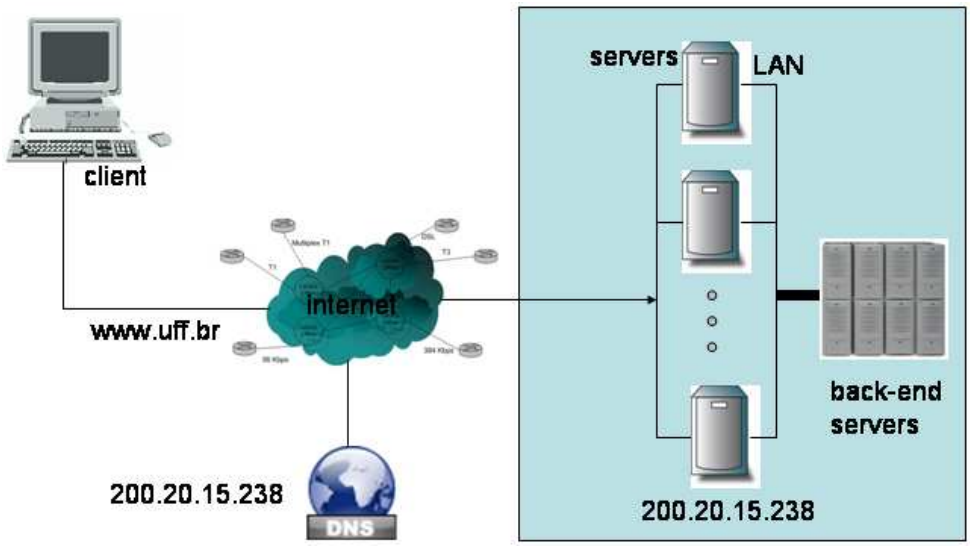


Figure 2.2: Virtual Web server architecture.

by the DNS mechanism, the Web system must have control over it. Unlike the cluster-based Web system, and similarly to the virtual Web cluster architecture, a distributed Web system does not rely on a front-end to perform the dispatching task. In some systems, there is also a second level routing that is typically carried out through some re-routing mechanism activated by a Web server that cannot fulfill a received request.

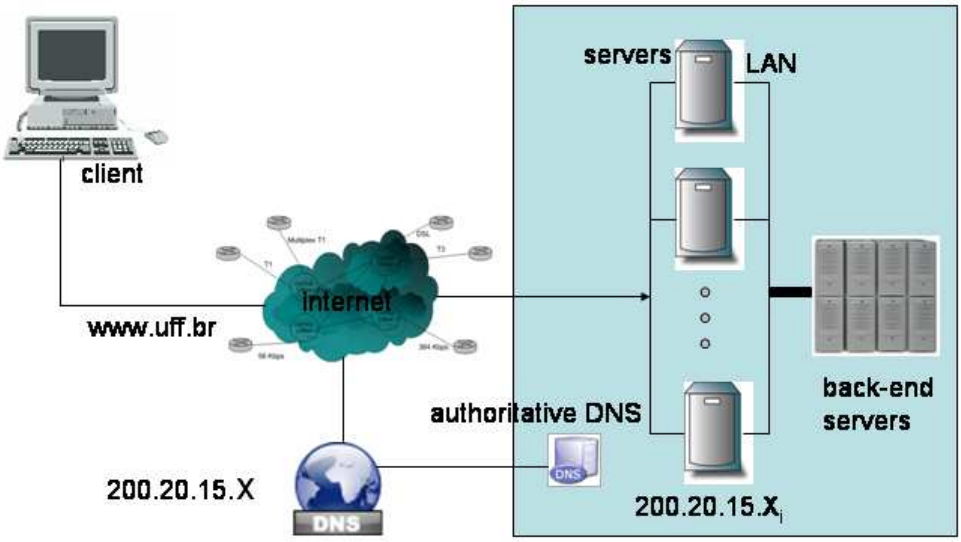


Figure 2.3: Distributed Web server architecture.

2.2 Request Routing Mechanisms

One of the main components of a typical multinode Web system is the routing mechanism, responsible for directing the request to the target Web server node. Although a server can be compound of several computers (sometimes in the magnitude of thousands), the client is not aware of it and this mechanism is responsible for presenting the cluster as a single entity to the outside world. Based on the way this mechanism works, it can be mainly classified into two categories: *content-blind* and *content-aware*. The following sections detail them.

2.2.1 Content-Blind

The routing mechanisms belonging to this category operate at network layer-4 (TCP/IP). They are called *content-blind* because at TPC/IP level the front-end can not know the request content when taking the routing decision. The association between client and server is made through TCP/IP mapping by the front-end (binding table). New TCP connections are directed to a node based on a given dispatching policy (see Section 2.3) while packets belonging to a previously established connection will be directed to the proper node by looking up the binding table.

Layer-4 Web clusters can be classified on the basis of the mechanism used to route inbound packets to the target server and outbound packets back to the client. The main difference is in the server-to-client way. In two-way architectures, both inbound and outbound packets pass through the front-end, whereas, in one-way architectures, only inbound packets flow through the front-end.

- **Two-way architecture** – The front-end works as a broker in this approach. The client send requests to the front-end, which routes the packets to the proper server (the front-end sees only TCP/IP packets as it works in network layer-4), and whenever the server needs to send a reply back to the client, the outbound packets must also pass back through the front-end. The front-end role in this approach is, thus, rewrite the TCP/IP header and keep the binding table to take the right routing decisions. The TCP/IP header rewriting is done on inbound packets (when the front-end sends a packet from a client to the selected server the source of the message is no more the client, but the front-end) and on outbound packets (when the front-end sends the packet from the server that handled the request to the client the source is no more the server, but

the front-end). For this reason, it is also called *double-rewriting*. It is worth to point out that the header rewriting incurs in checksum recalculation overhead (front-end processing).

- **One-way architecture** – In one-way architectures inbound packets pass through the front-end, while outbound packets flow directly from the servers. It reduces the processing burden of the front-end, relieving the potential bottleneck capability of the front-end. Routing to the target server can be done by means of several mechanisms, like:

  1. *Packet single-rewriting* — Inbound packets are handled as in the *two-way* method. The difference is in the modification of the source address of outbound packets. The Web server, before sending the response packets to the client, does the job of the front-end in the *two-way* method by replacing its IP address with the VIP address of the front-end and recalculating the IP and TCP header checksum.
  2. *Packet tunneling* — The effect of IP tunneling is simply to transform the old headers and data into the payload of a new packet. The front-end sends to the Web servers a message containing the incoming packet received from the client. The Web server is liable for extracting the content of this messages and building the original request of the client by itself. When the server replies directly to the client, it uses the VIP address of the front-end as the source address.
  3. *Packet forwarding* — In this mechanism, front-end and server nodes share the same VIP address and the front-end directs the inbound packets to the proper server nodes by changing the MAC address of the packets. The premise that the front-end and the server nodes share the same LAN segment must hold. As servers and front-end share the same VIP address, each server recognize itself as a destination and can reply directly to the client without any modification in the TCP/IP header. Although front-end and server nodes share the same VIP address, inbounding packets reach only the front-end because the Address Resolution Protocol (ARP) is disabled on the server nodes.

2.2.2 Content-Aware

The routing mechanisms belonging to this category operate at network layer-7 (application level, also named network layer-5 in some literatures). They are called *content-aware* because, as long as the front-end routing mechanism operates at layer-7, it is able to know the content of the incoming request and take fine-grained routing decision based on server nodes information availability. In other words, the premise that any server node should be able to handle any request does not need to hold anymore because, based on the request content, the front-end can direct it to the proper server. Nonetheless, selecting a server to handle the request is a dispatching algorithm task and the request mechanisms presented here only constitute the background requirements that allow these dispatching algorithms to work.

Here once again, as in layer-4 Web cluster approach, the request routing mechanisms can be classified on the basis of how they route inbound packets to the target server and outbound packets back to the client as two-way architectures and one-way architectures.

- **Two-way architectures** – A common way of implementing request routing in this architecture is by the technique called *TCP gateway*. In this mechanism the front-end keeps a TCP persistent connection with each server node and whenever a client request arrives it is sent to the selected server through its TCP persistent connection. When the response arrives from the server, the proxy forwards it to the client through the client-front-end connection. A drawback of this mechanism is the fact that whenever a packet arrives at the front-end it has to flow up to the application layer. An alternative would be to do it just at the connection establishment and the following packets belonging to the same connection would be redirected immediately from the network layer. It is done by a technique called *TCP splicing* and incurs less communication overhead but demands operating system modification.

- **One-way architectures** – In order to allow the server node handling the request to reply directly to the client, a technique called *TCP hand-off* moves the endpoint of the TCP connection from the front-end to the selected server node as soon as the connection is established. The TCP hand-off mechanism remains transparent to the client, as packets sent by the servers appear to be coming from the front-end. Incoming traffic on already established connections arrive at the front-end and are redirected to the proper server node.

2.3 Dispatching Algorithms

Besides the request routing mechanism, another crucial component in a typical Web system is the dispatching algorithm. Unlike the request routing mechanism, which is liable for directing the request to the node that will handle it, the dispatching algorithm is the component responsible for selecting the best suited server to respond. Of course, this task must be carried out by some entity, henceforth named *dispatcher*. The dispatching policy influences not only the performance experienced by the end user but also the system scalability.

Dispatching algorithms can be classified into several ways: *centralized* vs *decentralized*, *static* vs *dynamic* and *content-blind* vs *content-aware*.

- **Centralized vs decentralized** – It evaluates whether the decision is taken in common agree with all entities that comprise the dispatcher or not. In other words, in centralized techniques the decision process is done through a common agreement among the entities that comprise the dispatcher, whereas in decentralized techniques the decision is taken locally. For instance, in virtual Web servers, each machine receives all incoming requests but decide to accept it or not based on a hash function applied locally on the client IP and port information (all servers, besides handling requests, comprise the dispatcher). As long as the hash function is free of ambiguity, only one server will obtain a positive result and handle the request. On the other hand, in cluster-based Web servers, the decision is taken by the front-end (taking charge of the dispatcher task). In the former case we have a decentralized approach, whereas in the latter case we have a centralized approach.

- **Static vs dynamic** – The scheduling policies made without considering any dynamic state information, and hence fast, are named static. Most commercial products prefer these simple algorithms for doing server selection. Random and Round Robin (RR) are examples of static algorithms. In dynamic policies some kind of run-time system state information is used by the policy to allow a more accurate decision. Such information can be the server capacity, load, number of active connections, content and so on. Depending on the kind of information used, dynamic policies can be further classified into:

  1. *Client state aware* — The dispatcher dispatches requests based on client information. For example, a dispatcher working at application layer can

examine the entire HTTP request and take decisions on the basis of more detailed information about the client. On the other hand, layer-4 routing mechanisms can provide up to network layer client information.

2. *Server state aware* — The dispatcher assigns requests on the basis of some server state information, such as current and past load condition, latency time, and availability.
3. *Client and server state aware* — To use only client information can lead to server misutilization, while using only server information do not enable the request dispatching to take advantage of some legacy technologies used to improve server performance, like caching. Client information is often combined with some server state information to provide the so called *client affinity* taking into account server availability.

- **Content-blind vs content-aware** – Content-blind policies comprise the set of policies that do not take into account the client's request content, working at the TCP/IP layer. They are best suited for layer-4 request routing policies because the only reason why it makes sense to use layer-7 request routing mechanisms is to allow the implementation of content-aware dispatching algorithms, so that it can take more accurate decisions by exploiting possibilities like improved cache hit ratio and content partitioning.

Besides the wide variety of classifications, dispatching algorithms for Web servers are broadly classified into content-blind and content-aware because of the importance of this factor in the overall system infrastructure. Remember that request routing mechanisms are closely related to the dispatching policies (as long as dispatching policy implementation requires request routing mechanisms support) and that request routing are classified into content-blind and content-aware. The following sections will discuss some content-blind and content-aware policies.

2.3.1 Content-Blind

Both static and dynamic policies fit into content-blind dispatching policies. However, the range of information that can be provided to the dynamic policies are not so wide as in the content-aware approach. As a content-blind premise, any server must be able to handle any request.

Random and RR are two examples of static algorithms that use content-blind dispatching. In Random algorithm any server in the Web server system can be picked for request processing with the same probability. RR algorithms select servers in a cyclic manner, that is, the servers are organized in a specific order and the request is always handled by the next one. This approach requires only the knowledge of the last server that responded a request and the number of servers. As static algorithms do not consider the current state information, they have the disadvantage of possibly selecting faulty servers. An extra effort to improve this set of algorithms is the use of weights. A modified version of Random and RR algorithms are the Static Weighted Random (SWR), where servers can have different probabilities of receiving a incoming request, and Static Weighted Round Robin (SWRR) algorithms, where an weight is assigned to servers indicating their capacity. SWRR still works in a cyclic manner, but a integer greater than 0 is assigned to each server to indicate how many requests it has to receive in one round of the algorithm. This improvement is able to handle heterogeneity in the server but, as long as it is still a static algorithm, it can deal only with off-line given information, like server capacity. It can provide reasonable performance in case the request do not differ very much from one another in resource consumption.

In client-state-aware dynamic policies the dispatcher uses the client's source IP address and TCP port number for server selection. Servers are statically partitioned and the same clients are assigned to the same servers based on a hash function, which is applied to the client's address. Assigning clients to the same servers can better exploit some *client affinity* issues, like Secure Socket Layer (SSL). In the SSL case, server and client can use the same keys previously negotiated, avoiding the initial negotiation overhead. However, the lack of server state information and uncoordinated client usage of the server (at a given time only clients with a range IP addresses handled by a specific server might be accessing the Web server) can lead to load unbalancing.

Server-state-aware policies often use Least Loaded and Dynamic Weighted Round Robin (DWRR) algorithms as their routing approach. Algorithms like Least Connections (LC) and Fastest Response Time (FRT) fit into the least loaded approach. In LC new requests are assigned to the servers with the fewest active connections, while in FRT Web servers with smallest latency time in the last observation interval are assigned a new connection, as they respond faster. In the DWRR and DWR algorithms, each server is assigned a dynamic weight to, which is proportional to the server load state. They still logically work as their static versions. The dispatcher gathers load information periodically from the Web servers and the dynamic weights are computed. However, *client affinity*

is not tackled.

The client-server-state-aware policies set has been proposed to exploit *client affinity* yet taking into account server state information. As long as the content-blind approach only permits up to network layer information access, a very coarse *client affinity* can be obtained and server load information allows a more accurate dispatching decision. For instance, any of the server-state-aware policies previously discussed could be implemented with a buffer storing the last clients that accessed the Web server in a recent past, so that any new incoming request coming from one of these clients can be routed to the same server it interacted with.

2.3.2 Content-Aware

The complexity of layer-7 front-end that can examine the HTTP request motivates the use of more sophisticated content-aware distribution policies. Potential features to be tackled are:

1. improving reference locality in the server caches so to reduce disk accesses (*cache affinity*).
2. using specialized server nodes to provide different Web-based services (specialized servers), such as streaming content, dynamic content, and to partition the Web content among the servers, for increasing secondary storage scalability.
3. increasing load sharing among the server nodes (load sharing).
4. when HTTP/1.1 persistent connections are used, a layer-7 front-end can assign requests traveling on the same TCP connection to different servers, thus achieving a finer granularity control

.

Content-aware dispatching rely on content-aware request routing mechanisms and use of dynamic policies. It does not mean that static algorithms are not supported, but it just makes no sense to use a more complex routing mechanism if their advantages are not exploited. Cache Affinity, Specialized Servers, Load Sharing, and Client Affinity algorithms fall under *client-state-aware* policies. Cache Affinity algorithms partition the file space among servers using a hash function applied to each URL. Some Web sites provide heterogeneous services and their Web servers can be partitioned according to the

services they handle. These specialized servers are employed to manage certain types of requests like dynamic content, multimedia files, and streaming video. This content-aware dispatching algorithm is referred as Service Partitioning that comes under the specialized server category. Size Interval Task Assignment with Equal load (SITA-E) and Client-Aware-Policy (CAP) support load sharing. SITA-E algorithm deals with assigning tasks of certain size range to individual servers. The advantages of using the SITA-E is that it limits the range of task sizes assigned to each server and reduces the waiting time of the tasks. A popular content-aware algorithm is the Locality-Aware Request Distribution (LARD), which considers both client and server information for its server selection. In LARD the incoming requests are directed to the same servers from the same clients until a given load threshold is reached. When the threshold is exceeded requests are redirected to the least loaded server or the server with the fewest connections.

2.4 Placement of Web Content and Services

The scalability of a Web cluster depends also on the methods used to organize and access information within the site. Content-aware dispatching algorithms are tightly coupled with data location as the best suited server decision will surely rely on both data location and server's availability information. Data popularity (term used to designate how often the data is required), size and temporal locality (term used to express that, once the data has been accessed, the probability of being accessed again in a foreseeable future increases), among other factors, must be considered in order to allow improved load balancing and performance. For instance, storing all most popular data into just a group of servers and all unpopular data into another group of servers would incur more request routing to the first group, unevenly balancing the load and degrading responsiveness. Several strategies aiming to tackle this problem are under research, being basically classified accordingly to the data category: *static content* and *dynamic content*. The following sections present an insight into these strategies.

2.4.1 Static Content

When we consider locally distributed Web systems that do not use a content-aware dispatching mechanism, any server node should be able to respond to any request, unless they employ internal rerouting (see section 2.2). It means that either all servers have the same stored data (replication) or they have access to the same data storage.

The first technique, named replication, implies that each server maintains a local copy of the data on its disk. The main advantages of this approach is that access to data can be done with no extra communication overhead and it is fault tolerant. However, some drawbacks arise, like data consistency (it requires any content update to be propagated to all the nodes in short periods of time) and storage overhead (maintaining a copy of the data on each server wastes a lot of disk room). Taking into account that the majority of Web requests are for read-only access, where maintenance of consistency is not crucial, and that for very popular data replication becomes essential to prevent disk from being the system bottleneck, this technique is highly employed under these circumstances.

The other technique consists of having all servers sharing the same information through a distributed file system such as the Network File System (NFS) [36]. It avoids the consistency problem, as all servers access the same data, and the waste of disk room necessary to store the data copies. However, it incurs extra communication overhead to retrieve the data, thus increasing the response time experienced by the end user, and in case of highly accessed data it becomes the system bottleneck. Each technique has its benefits and drawbacks. The choice of the best solution is highly dependent on the system characteristics and a hybrid implementation of both techniques can also be exploited to minimize each other's drawback. For instance, using both replication and data sharing we can minimize the storage overhead and the potential system bottleneck, as data is replicated only at a level enough to keep an acceptable system performance. However the consistency and communication overhead still hold.

Web clusters that do use content-aware dispatching mechanisms can use the same two strategies presented previously and they can also use a third alternative by partitioning the content tree among the Web server nodes. This technique has two main advantages: it increases secondary storage scalability without the overhead due to a distributed file system; and it allows the use of specialized server nodes to improve responses for different file types, such as streaming content, CPU-intensive requests, and disk-intensive requests. However, content partitioning can lead to load imbalance produced by the uneven distribution of Web document popularity. Techniques to improve features like responsiveness, data consistency maintenance and content partitioning are still a growing research area.

2.4.2 Dynamic Content

The Web has changed from a place for static content storage and sharing to a place that provides personalized services for clients, bearing security, commercial transaction, client

usage control by server administrators (with this information it is possible to tailor the content served to a client according to his most recent needs and expectations) and so on. These activities usually demand a little server processing, besides the data retrieval, requiring orders of magnitude higher service time. New technologies supporting this innovation in Web-based services must take into account the inherited Web legacy and take place harmoniously with them. Dynamic Web-based services, databases and other (legacy) applications are typically hosted on a set of servers behind the Web server nodes, forming a multitiered system. Typically, requests for dynamic content are forwarded to servers of other layers that run the software that executes the business logic, and interacts with database servers or other legacy applications.

Furthermore, a multitier system require request routing and dispatching to be implemented at different levels, from the front-end to the Web server layer to the Web application server layer. Moreover, dynamic request serving must be aware of traditional problems like load balancing, bottleneck effect, responsiveness, reliability, and so on, in order to avoid hindering the overall system behavior. Similarly to previously analyzed techniques, here we can also make use of traditional techniques like partitioning, replication and so on, taking into account their advantages, drawbacks and analyzing the trade-off. As long as dynamic content providing is very important nowadays in the Web context, there are a lot of commercial solutions under utilization. However, for simplicity, all products prefer to use fast development solutions, implementing very simple algorithms, typically round-robin and least loaded [9]. The techniques and challenges in this field goes beyond the scope of this work and will not be further exploited.

Chapter 3

Power Management Techniques

Computing systems, ranging from small portable systems to more complex general purpose systems, are designed to satisfy various computational demands in some acceptable time. Concerns with portability, performance and economical issues are the reasons why the main goals of different architectural studies have long been speed, size and price, driving technological innovations to the development of faster, smaller and cheaper computer systems.

Computer systems are mainly composed of digital logical circuits. As long as digital logical circuits are composed of transistors, the search for smaller and faster computers has made it possible to incorporate millions of these devices on a very small die, and to clock these transistors at very high speeds. To pack more transistors together and enable them to operate at higher speeds, chipmakers have shrunk them to once unthinkable sizes. For example, using its newest 0.13 micron manufacturing process, Intel now makes transistors as small as 0.06 microns. By comparison, a human hair is about 50 microns wide [28].

Nonetheless, while these innovations and trends have helped to provide tremendous performance improvements over the years, they have at the same time created new problems that demand immediate consideration. One of the major obstacles in designing smaller transistors is how to prevent electricity from seeping out of components, which becomes a bigger problem as the parts are made ever smaller. Within a transistor, an element called *gate* controls the electrical flow through it, and as transistors continue to shrink, it becomes more difficult to effectively control *switching*. Historically, the primary source of energy dissipation in transistor devices has been the so called *dynamic energy*, which is consequence of charging/discharging load capacitances when a device switches. When the circuits were not active the current was extremely low relative to switching the gates, and thus, the energy consumed, called *static energy*, was negligible. In order to

improve performance, chip designers have relied on scaling down the transistor supply voltage in subsequent generations to reduce the *dynamic energy* dissipation, which become too high as a much larger number of transistors are set on a die. However, if chip makers had continued on their old path, by the year 2015, microprocessors would be dissipating more watts per square millimeter than the surface of the sun [26]. In order to maintain high transistor switching speeds, scaling down transistor threshold voltage along with the supply voltage was required, thus giving rise to a significant amount of *static energy* leakage due to an exponential increase in subthreshold leakage current even when the transistor is not switching [14].

These aspects presented so far about transistor design refer to the CMOS transistor family. The vast majority of nowadays microprocessors are based on this technology. CMOS (Complementary Metal-Oxide Semiconductor) refers to both a particular style of digital circuitry design and the family of processes used to implement that circuitry on integrated circuits (chips). The central characteristic of this technology is that it only uses significant power when its transistors are switching between on and off states, unlike other forms of logic implementation. The word *complementary* refers to the fact that the design uses pairs of transistors for logic functions, only one of which is switched on at any time. The phrase *metal-oxide-semiconductor* is a reference to the nature of the fabrication process originally used to build CMOS chips. To address the thermal issue, research has moved toward utilizing new transistor designs and materials that permit further shrinkage and other improvements, but CMOS technology still remains widely employed in most circuitry systems, like microprocessors, Static Random Access Memory (SRAM) and other digital logic circuits, despite some innovations.

The settling time for a gate in a CMOS circuitry is proportional to the signal and the lower the voltage drop across the gate, the longer the gate takes to stabilize. Therefore, to lower the voltage and still operate correctly, the cycle time must be lowered first. As performance remained the main efficiency factor in some computer systems, the transistor shrinkage was followed by an increasing cooling research effort to remove heat from these new generation processors highly packed with shrunk transistors, or else it would be unfeasible to build such processors. Since CMOS circuitries are used in several digital logic circuits that compound a computer system, they come to be of main concern in thermal issues.

In this chapter we will discuss the motivation for PM research (Section 3.1), a taxonomy of PM techniques (Section 3.2) and some recent investigations on PM that has been

conducted in clusters of servers (Section 3.3), which is the focus of this work.

3.1 Power Management Importance

From portable battery-operated computing systems to server farms in data centers, power consumption is rapidly becoming the key design issue. Portable devices usage has increased a lot these days, as users call upon them to manage their schedules, check their e-mail, maintain their address books, and carry out their everyday work tasks. Along with the increasing wireless network development, it moves toward a vision of completely cooperative intelligent mobile devices with instant access to spread information and cleverly embedded into everyday user life, the so called *pervasive computing*. However, resource limitations of these devices will have a great effect on available services, and power supply was identified as a critical issue ever since the beginning of mobile computing research. As long as these devices are not connected to the electrical power grid, energy becomes a critical resource and the trade off between processing power and energy saving comes to be a main concern. These devices must have computational power to support the complex applications envisioned to run on them in the *pervasive computing* world, yet taking into account the power issue to maximize the device lifetime.

The desktop computer systems, such as data centers, mainframes and so on, lived another reality. The growing necessity of more processing power to support complex applications propelled researches in this field toward performance and reliability directions. It urged the development of distributed systems and processor improvements. In the quest for higher performance, the cluster solution appeared, grouping a bunch of faster machines to work together to achieve a common goal. The appearance of blade servers in the cluster context totally reflected the seek for faster, smaller and cheaper computer systems goal. Blade servers are self-contained computer servers designed for high density, having many components removed for space, power and other considerations while still having all the functional components to be considered a computer (memory, CPU and I/O devices). However, this large amount of computers fitted into a single room turned power supply into a main concern. In spite of having constant power supply provided by the the electrical power grid, energy is not free and its cost represents a major contribution in the overall system expense.

Besides the financial cost issue, which represents a important motivation toward energy saving research in the real world, researchers have recognized that thermal issues

merit a thorough investigation, that is, more than just an extension of power-aware methods. Overheating deserves special attention because it negatively impacts on important factors, like:

- hardware failures, thus compromising reliability;
- considerably reduction of system components lifetime, incuring more expenses with their replacement;
- energy dissipation, that is, electrical power waste;
- and, very complex and efficient cooling system demanding, which in its turn also consumes energy to operate and result in extra expense.

Fundamentally, since temperature is a consequence of power dissipation, power-aware design and the emerging field of temperature-aware design are intrinsically related, but they also have significant differences. Although generally consistent with each other, the two goals are not identical. Some PM techniques address both goals, but most techniques focus on only one. Temperature is proportional to *power density*, not just power, so methods to reduce thermal effects can either reduce power, increase area, or both. The common use of heat spreaders in modern high-performance microprocessors provides one example of increasing area to reduce power density and to deal with thermal effects. Power density has come to be of main concern lately specially because of the development of even more compact computer systems, thus concentrating more power to operate these devices in ever less room, and hence increasing power density.

As we can see, so far performance and energy-aware computing design have traditionally led to opposite directions. The former targets miniaturization, packing and more power to reduce clock. The latter targets to increase the area, reduce voltage (hence power) to reduce energy dissipation and save electrical power for financial purposes. However, understanding and exploiting thermal effects is also critical because of their impact on packaging, reliability issues, performance, and leakage power. Electromigration, aging of the thin oxide, mechanical failure due to thermal gradients, and expansion coefficients are the main factors that result in increased failure rates under thermal stress [38]. Performance is lower and leakage power exponentially higher at high temperatures [35], which suggests that effectively dealing with thermal issues can increase performance and reduce power consumption simultaneously. For this reason, researchers are investigating active cooling and PM techniques to control temperature, which may become a requirement in a foreseeable future. Henceforth, we will only investigate the PM approach.

3.2 Power Management Taxonomy

Most energy management techniques consist of one or more mechanisms that determine the power consumption of system hardware components and a policy that determines the best use of these mechanisms. Although developers can implement some energy management techniques either in hardware or in software, combining software policy with hardware support is the most beneficial. Placing the policy in software allows for easier modification and adaptation, as well as more natural interfaces for users and administrators.

Despite being designed for high performance, the full potential of a machine is not fully utilized all the time. Most computing systems have at least two modes of operation:

- an *active mode*, in which the processor or device continues to operate, but possibly with reduced performance and power consumption. Processors might have a range of active states with different frequency and power characteristics;
- and an *idle mode*, in which the processor or device is not operating. Idle states vary in power consumption and in latency for returning the component to an active state.

It is acceptable to have higher power consumption in the active mode as a trade-off to increased performance, but any power consumed when the system is idle is a complete waste and ideally should be avoided by turning the system off. Most often, PM techniques aim to exploit this characteristic by transitioning hardware components back and forth between high and low-power states or modes in order to tune performance accordingly to the computational power demand of the system.

There is an open industry specification, called ACPI (Advanced Configuration and Power Interface) [3], co-developed by Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba to standardize interfaces for OS-directed configuration and PM. In ACPI, the BIOS provides the OS with methods for directly controlling the details of the hardware, so that it has nearly complete control over the power savings. It allows more accurate energy savings than could be obtained when the BIOS was in charge of this task, since the OS has control over all elements of the system and their on-line state.

Power reduction techniques can be classified as *static* and *dynamic*. *Static Power Management* (SPM) techniques, such as synthesis and compilation for low power, are applied at design time (off-line) and target both hardware and software. In contrast, dynamic

techniques use runtime (on-line) behavior to reduce power when systems are serving light workloads or are idle. These techniques are known as *Dynamic Power Management* (DPM). The latter is more flexible, but the results achieved are worse than in the former approach [12]. The following sections discuss them in more detail.

3.2.1 Static Power Management - SPM

Power dissipation limits have emerged as a major constraint in the design of microprocessors because of its imposed constraints on implementation feasibility, performance and reliability. For these reasons, power optimization requires careful design at several levels of the system architecture. These studies can be divided mainly into two areas. The first one is a low-level approach targeting the architectural level by investigating power consumption at both cycle and instruction levels. The second approach is a high-level approach targeting different or all system components. These approaches will be discussed in the following sections.

3.2.1.1 Low-level Approaches

Modern computer architecture designers face the new reality of trading off between performance and power dissipation. Historically, designers have concentrated their effort on high performance attainment by shrinking transistor's size, decreasing their gate stabilization time, executing more instruction per CPU cycle, optimizing memory data access and so on. Performance improvement generally comes along with computational resource waste. For instance, waste of computational resources occurs when redundancy is applied at instruction level to predict a execution flow or a memory data access, and hence, useless work leads to useless energy consumption.

Power consumed by the CPU is significant, followed by main memory and hard disk. For instance, a mid-range server with dual 1.4-GHz Pentium III processors draws about 90 W of DC power under load: roughly 55 W for the two CPUs, 10 W for a disk drive, and 25 W to power DRAM and other elements. With a typical efficiency of about 75 percent for an ATX power supply, this translates into 120 W of AC power per server [6]. That is why these are the main target components to investigate power aware strategies.

Some off-line strategies targeting these components are:

- ***Transistor design*** – Originally, transistor design headed to ever more shrunk and faster directions. However, overheating due to energy leaking made it un-

feasible, denying Moore's law that predicted a transistor density duplication every 18 months on semiconductor chips. Nowadays, researchers seek another means to improve performance rather than shrinkage. A promissory trend is to find new materials to compose the transistors, specially the gate, which due to shrinking is the main responsible for energy leakage. Instead of metal, as designed originally, recently gate electrodes have been almost always made of a different material, usually polysilicon. However, Intel investigated a new material called High-K/Metal and claims that found the perfect composition of this material that avoids the leakage drawback on very small gates, yet providing performance accordingly to Moore's law [4]. Moreover, *Silicon On Insulator* (SOI) technology is a layered structure that replaces the traditional silicon as a semiconducting material on chip design. This material yields reduction in the amount of electrical charge that the transistor has to move during a switching operation, thus increasing speed (up to 15%) and reducing switching energy (up to 30%) [34]. New transistor design features, like channel width, are also investigated.

- ***Multi-core processor*** – Processor architecture, traditionally designed to exploit performance improvement, face a new reality. Transistor shrinkage turned power dissipation into a main concern due to energy leakage increase. The new designing trend was shifted from maximizing performance to maximizing performance per watt, mainly by putting more than one microprocessor on a single chip and running them all well below their top speed. Because the transistors are switching less frequently, the processors generate less heat. And because there are at least two hot spots on each chip, the heat is spread more evenly over it, so it is less damaging to the circuitry and easier to get rid of fans and heat sinks. This approach targets power dissipation distribution over the chip in order to tackle the power density constraint and has been broadly investigated [26].

- ***Gating pipeline registers*** – The selective gated pipeline register optimization focuses on reducing the power consumption by using the control signals of the datapath for selectively gating subsets of the pipeline registers. It was observed that 23-36% energy reduction is possible in the datapath, 50% in the pipeline registers and 46% in the register file [12].

- ***Bus power optimization*** – It consists of reducing the switching activity on the *instruction caches* (i-caches) data bus by relabeling the register fields of the compiler-

generated instructions. Up to 12% reduction in the total energy reduction in the i-cache data bus using the register relabeling optimization can be obtained [12].

- ***Memory hierarchy optimization*** – Unlike dynamic energy which depends on the number of actively switching transistors, static energy is a function of the number of on-chip transistors, independently of their switching activity. As the gap between static and dynamic energy diminishes, caches account for a large (if not dominant) component of energy dissipation due to leakage in recent computer system designs, and will continue to do so in the future. Several strategies have been proposed to reduce energy leakage in all levels of memory hierarchy, like data duplication, standby leakage mode, cache decay mechanisms and so forth [24].

  Memory has a huge internal bandwidth compared to its external bus bandwidth. To exploit the wide internal bus, cached DRAM (CDRAM) adds an SRAM cache to the DRAM array on the memory chip that acts as an extra memory hierarchy level. It potentially improves system performance, but the work presented in [5] exploits the benefit of having on-memory cache with respect to its energy consumption by powering off DRAM after the data is transfered to CDRAM.

- ***Instruction level optimization*** – Instruction level issues impact on:
  - switching effects;
  - reducing pipeline stalls;
  - instruction packing;
  - and, instruction's energy consumption.

    These features are now being studied from a performance/power-aware trade-off standpoint rather than from a purely performance one. It is well known that a high level instruction can be mapped to several low level instructions that result in the same logical operation. OS and compiler optimization can attempt to minimize energy consumption by generating a optimum energy-aware set of instructions. This instruction set choice tries to reduce switchings, pipeline stalls and choose the less energy hungry ones. Meeting these constraints is not straightforward but undertakes significant energy saving.

3.2.1.2 High-level Approaches

There is little benefit in optimizing only at the device level if other elements participate or sometimes even dominate the power consumption and dissipation. To effectively optimize system energy, it is necessary to consider all the critical components and their relationship. Several strategies investigate the power consumption on different system levels, targeting both hardware and software on different levels of abstraction. However, high-level approaches are more common in dynamic PM because it can take advantage of application and system information to perform on-line adaptations.

A simple PM strategy used in server clusters is the newly blade server [41]. This server basically consists of a case that provides power, cooling and other general purpose components to a set of mainboards connected to it. Yet, the main components that compound a computer must still be present on each mainboard (CPU, memory and I/O devices). A blade server carries only the absolute essentials: a small motherboard with processor, memory, and networking circuitry, and maybe a hard drive. Because they share power supplies and have fewer functional chips and other components, blades consume less electricity and generate less heat, addressing both energy saving and dissipation.

3.2.2 Dynamic Power Management - DPM

Dynamic power reduction techniques use runtime behavior to reduce power when systems are serving light workloads or are idle. These techniques are often implemented in software with hardware support, where the software captures system dynamic behavior to perform more flexible PM adaptation on hardware level. DPM can be achieved in different ways. For example, *Dynamic Voltage Scaling* changes processor supply voltage at runtime as a method of PM. DPM can also be used for shutting down unused I/O devices or unused nodes of server clusters. Alike SPM, DPM can also be deployed at low and high levels. The following sections describe, respectively, the low-level and high-level approaches.

3.2.2.1 Low-level Approaches

CMOS circuits in modern microprocessors consume power in direct proportion to their frequency and to the square of their operating voltage. However, voltage and frequency are not independent of each other: A CPU can safely operate at a lower voltage only when it runs at a low enough frequency. Thus, reducing frequency and voltage together reduces energy per operation quadratically, but only decreases performance linearly.

A mechanism named *Dynamic Voltage Scaling* (DVS) aims to save power by dynamically changing the voltage supply to the processor. The intuition behind the power savings comes from the basic theoretical result that energy is proportional to the cubic of the voltage while frequency is directly proportional to voltage. Therefore, voltage scale-down provides quadratic energy savings ($\frac{E}{clock} \propto V^2$) [40]. Nonetheless, for a real chip it may not be possible to reduce the voltage linearly with the clock reduction due to external factors, like thermal effects. However, if it is possible to reduce the voltage at all by running slower, then there will be a net energy saving per cycle. DVS techniques attempt to minimize the time the processor spends running the operating system idle loop, but DVS is not a free operation by time and energy standpoints and accurate high level knowledge of system behavior collaborates to apply DVS efficiently. These aspects can be exploited to attain energy savings in many other digital logic circuits, like memories.

Memories are also based on CMOS circuitry to store data. Some new memory types have different activity states to exploit energy saving opportunities. For instance, the *Double-Data Rate Synchronous Dynamic Random Access Memory* DDR-SDRAM, has two low power modes: *power-down* and *self-refresh*. In the deep *power-down* mode, the memory array's power is eliminated to further reduce power, thus not retaining data. In *self-refresh* mode energy is saved by allowing only data retainment and disabling read/write operations. Switching to power-down mode or back to active mode takes only one memory cycle and can reduce idle DRAM power consumption by more than 80% [22]. Self-refresh mode might achieve even greater power savings, but it currently requires several hundred cycles to return to active mode. Thus, realizing these mode's potential benefits requires more sophisticated techniques.

Reducing overhead due to cache coherence traffic can also improve energy efficiency. Shared-memory servers often use invalidation protocols to maintain cache coherency. The cache controllers of all processors in the server share a bus to memory and the other caches. Each cache controller arrives at the appropriate coherence action to maintain consistency between the caches based on its ability to observe memory traffic to and from all caches, and to snoop remote caches. Snoop accesses from other processors greatly increase L2 cache power consumption [22].

Hard Disks (HDs) are a non-volatile data storage device that, unlike DRAM memory family that are based on logic circuitry, store data on a magnetic surface layered onto hard disk platters. A HD uses parallel rotating platters (disks) to store data through a mechanical arm equipped with a read/write head that applies a magnetic field onto

the platters. This data storage technology provides two opportunities to save energy when active: reducing the platters rotation by applying a lower voltage and save energy based on the quadratic energy saving due to the voltage scaling; and minimizing the need of the mechanical arms movement [23]. Minimizing the need of mechanical arm movement requires higher level knowledge of data storage, but rotating platters slowlier is a hardware support that enables higher level strategies to work. Obviously, reducing the platters rotation speed decreases data throughput, thus cannot be done carelessly. These and others high-level approaches will be discussed next.

3.2.2.2 High-level Approaches

As we could see in the previous section, CPU, memory and HD have different active states. Several components in most computer systems have this capability and in all of them state transitions are not cost free. General purpose algorithms like *time-out policies* and *predictive policies* perform poorly and consider only active-inactive-active transitions, thus state transition must be strongly dependent on system behavior and target component in order to be more flexible and offer better saving.

Much of the recent work in DVS has sought to develop prediction heuristics, but application specific approaches can better exploit this feature. The bottom line is to use workload information to apply DVS and avoid idle CPU cycles. Possible approaches include slowing down frequency and taking longer to serve the workload, working in batch and so on. Of course, the kind of application must be taken into account. For instance, real-time applications cannot benefit from batching approaches and frequency scaling must be done carefully, since this applications must respond within a given time.

Multiprocessor systems can use processor packing to save energy in high performance systems, since in these systems power dissipation in active and idle states are similar because as transistors shrink ever more static and dynamic energy dissipation get closer to each other. Processor packing consists of concentrating the load onto a smallest number of processors turning the remaining processors off, because since in these high performance processors idle cycles consume almost as much energy as busy cycles it is better to keep as many of them as possible inactive and the others running at full power. CPU can also benefit from parallelism to improve utilization and energy consumption by replacing high-performance processors with a larger number of slower but more energy-efficient processors providing the same throughput. Of course not all kind of applications can easily benefit from parallelism, since dependency chains can stall other parts of the system [22].

Similarly, HDs can save energy by scaling down platters rotation speed but it results in less throughput. Systems with intensive I/O operations can benefit from it in moments of low workload, while general systems can exploit caching on HDs to minimize the necessity of accessing it. Efficient data placement strategies can aim to pack data together using some knowledge over them in order to minimize HDs's arm movement [23]. In server clusters, where data replication is crucial for performance purposes, data placement can be exploited to turn off some parts of the storage facility during periods of low workload [21].

The Operating System (OS) can change components' state based on its activity. Any component that is not performing any task is consuming energy uselessly, and hence should be turned off. However, just because a component is idle now does not mean that it will not be required in the near future. Moreover, state transition is not a free operation, resulting in time and energy overhead. As a matter of fact, the more saving a state is producing from a power standpoint, more expensive is in time and energy overhead to bring it to the active state. The number of activity states is a component dependent information. For instance, some mobile devices' displays have a backlight to light the screen. Whenever the OS detects a large period without user interaction with the system, first it turns off the backlight and then, if the inactivity holds, the whole display is turned off. This period is usually a user configuration.

OS can also exploit PM at process level. *Multithreading* is a common feature in most computer systems to improve system's responsiveness. However, process scheduling is a complex activity because state saving can involve complex data management, which is power expensive. For instance, caching takes advantage of code locality, like loop codes in the program that causes a small set of instructions to be executed repeatedly, justifying its translation to a small and faster memory (cache). During a process preemption, its information stored in cache might be deleted due to lack of space and it will have to be reloaded later on. It is a expensive task from a performance and power-aware standpoint that must be addressed at OS level. Another research direction involves reordering the program's instruction execution to optimize it from a energy efficient point of view, aiming to minimize data transfer necessity [12].

3.3 The Cluster Case

Power consumption is rapidly becoming a key design issue for servers deployed in large data centers and Web hosting facilities. These centers typically host clusters of hun-

dreds and sometimes thousands of servers, such as Web and application servers running on off-the-shelf hardware. These installations are designed to handle peak load, hence leading to a seek for performance. However, as transistor density and demand for computing resources have rapidly increased, even high-end systems face energy-consumption constraints.

All the power a system consumes eventually dissipates and transforms into heat. The power dissipation and related thermal issues affect performance, packaging (remember that temperature is proportional to power density), reliability, has environmental impact, and heat removal costs; the power and energy consumption affect power delivery costs because high power consumption demands proper power delivery infrastructure, which, in its turn, limits the number of nodes that can be fitted into a room, thus compromising performance.

Systems with high peak power demands require complex and expensive cooling configurations to efficiently move heat away. Providing proper cooling becomes ever more challenging as increasing performance demands have been leading to higher power densities. High peak power requirements also translate into large and expensive constant power supplies and backup power generators, which are necessary in case of a power outage [41].

Reduced-power servers are attractive for large-scale clusters, but some aspects have to be taken into account. First, reduced power is desirable, but, it must come without a corresponding performance penalty: what counts is watts per unit of performance, not watts alone. Second, the lower-power server must not be considerably more expensive, because the cost of depreciation typically outweighs the cost of power. Realistically, a server will not last beyond two or three years, because of its disparity in performance when compared to newer machines [6].

Server clusters are particularly attractive for applying PM techniques because, although they are designed to handle high peak workload, the system is not under this condition most of the time. It means that often computational resources are being wasted in a typical data center, and so is energy. Moreover, as explained earlier, often energy efficient servers are very expensive and the energy saving might not keep up for the initial outlay due to their fast depreciation. Therefore, data centers are often composed of energy inefficient machines that are underutilized most of the time, a perfect background for PM application.

Several data placement techniques have been proposed in the literature for both memory and disk [22]. Data partitioning was first mentioned in Section 2.4 with performance

purposes, but energy-aware approaches are also being investigated. Since memory and disk greatly contribute for energy consumption, new hardwares are equipped with power control mechanisms that allow them to operated at different power consumption states. Data partitioning techniques attempt to save energy by transitioning memory and disk states, yet keeping up with the incoming workload. It is obvious that the less power-hungry a state is, the worst is its performance, if it is active at all.

In these systems, memory, disk and processor are the major contributors in energy consumption. Energy consumed by the cooling system is bothering as well, but as long as reduction in energy consumption comes along with less energy dissipation, solving the former we indirectly tackle the thermal issue.

Processors are one of the most power-hungry component in a server cluster. They are the target of this work and the PM techniques dealing with them will be discussed in the sequel. Paper [16] states that processor PM for server clusters can be mainly classified into five categories:

- **Independent Voltage Scaling (IVS)** – In this policy each node independently manages its own power consumption through DVS. This policy performs the intra-node PM, so all the nodes in the cluster stay in the active state even during periods of low workload. Each node may operate at different frequencies and voltage due to the workload variation and the difference in the computational demands of individual requests. However, since the request distribution mechanism balances the workload across all nodes, on average, each node will operate under the same load but possibly in different frequencies due to different processor capacities (heterogeneity). IVS requires that the cluster nodes have processors and infrastructure that support DVS and nothing else.

- **Coordinated Voltage Scaling (CVS)** – This policy uses DVS in a coordinated manner to reduce cluster power consumption. In contrast to IVS, the cluster coordinate their voltage scaling actions through a centralized monitor that periodically computes system load and energy consumption, decides the cluster configuration and reports to each node what frequency they have to operate at. As in IVS policy, there is no inter-node PM. Hence, all the nodes in the cluster are active even during periods of very low workload. The CVS is expected to save more energy than IVS because it can better handle frequency and energy consumption heterogeneity among the cluster nodes. Since the monitor has full knowledge of cluster nodes features, it can properly distribute the load and

set each node's frequency operation so that they can handle its part of the load. To implement CVS it is necessary the same support for IVS, that is, processors and infrastructure that support DVS, plus a central facility liable for performing the monitoring task and software infrastructure to allow this central facility to remotely coordinate the DVS adaptation.

- **Vary-On Vary-Off (VOVO)** – This policy turns off server nodes so that only the minimum number of servers required to support the currently incoming workload are kept active. Nodes are brought on-line as and when required. VOVO does not use any intra-node PM and hence can be implemented in a cluster that uses standard high-performance processor without DVS. Besides, as already explained in the beginning of this chapter, high performance processors tend to dissipate as much static energy as dynamic energy, which makes idle moments extremely inefficient and it is totally worth to keep only the minimum required number of nodes active at a time. To implement VOVO some hardware support, such as a Wake-On-LAN network interface, is needed to signal a server to transit from inactive to active state. Wake-On-LAN network interfaces allow a "sleeping" computer to be brought up to full power remotely over a network and the computer, while in this low-power state, spends little energy only supplying the network interface to keep it active. VOVO can be implemented as a software service running on one of the cluster nodes (or a separate support server) to determine whether nodes must be taken off-line or brought on-line and requires software probes running on each of the cluster nodes to provide information on the utilization of that node. The load distribution mechanism must be aware of the state of the servers in the cluster so that it does not direct requests to inactive nodes, or to nodes that have been selected for vary-off and have not yet completed all received requests.

- **Combined Policy (VOVO-IVS)** – This policy is a combination of the VOVO policy to reduce the number of active servers and the IVS policy to reduce power consumption on individual nodes. VOVO-IVS complexity is similar to that of VOVO.

- **Combined Policy (VOVO-CVS)** – Conceptually, this policy is a combination of the VOVO policy to use the fewest number of active servers and the CVS policy to reduce the power consumption of individual active nodes. However, in this policy the PM actions of the two policies are integrated to use the most effective

mechanism whenever the maximum cluster capacity is not needed. In particular, VOVO-CVS places a larger emphasis on voltage scaling than VOVO-IVS, since DVS provides a quadratic benefit whereas node vary-off provides only a linear benefit. Besides, the overhead to bring a server on is much higher than the one due to voltage scaling. Approaches for this policy may consider applying CVS as much as possible and only then apply VOVO; apply VOVO and use CVS to individually adjust each processor to tightly keep up with the incoming workload by avoiding idle cycles; or, monitor the workload and energy consumption to decide the most suitable cluster configuration at a time based on some algorithm. The last approach is more flexible, since it can balance the number of VOVO and DVS adaptation to provide more accurate energy savings. Operationally, this policy achieves this integration by constraining the range of frequency and voltage scaling based on the number of active nodes. It is more complicated to implement than VOVO-IVS and all the other policies previously described. The voltage scaling part requires processor with DVS support, the VOVO part needs hardware support to remotely awake computers in sleeping mode and both DVS and VOVO parts need a central monitor and software infrastructure to coordinate the cluster configuration. However, in case an algorithm is used to set up the cluster configuration based on the incoming workload, this technique might become too complex to make it up for little energy saving benefit.

Chapter 4

Related Works

Designing mechanisms for energy-efficient systems requires a good knowledge of the several power reduction techniques in several levels of the system, as presented in Chapter 3. In the context of Web server systems, the work presented in [7] identifies memory, storage device and processor as the most power hungry elements and describes some techniques to save energy in these components in a Web environment. This work also identifies not only energy consumption but also energy dissipation as a critical element when designing such systems. Moreover, it discusses the heterogeneous nature of Web server systems, feature often disregarded by early researches on this field. Heterogeneity is a key issue, since crashes are very frequent in these systems and replacement is rarely done by the same element due to the fast depreciation of them in the market (computer systems rapidly become obsolete). Several techniques have been proposed in the last decade to tackle these elements, like DVS for processors, self-refresh mode for memories and platters spin-down for HDs. Two recent works [39, 12], survey the "state of the art" in these techniques for reducing the total power consumed by a microprocessor system over time, giving insights into all levels and elements of the system.

The work in [21] aims to save energy in storage level through a request distribution mechanism named PLARD (Power and Locality Aware Request Distribution). Basically, it consists of partitioning the data and applying local techniques (multi-speed disks and pin-down memory) and dispatching request accordingly to server and data availability (cluster-wide decision). However, the majority of the researches on energy saving approaches for Web servers aims the processor. The work presented in [40] is one of the first attempts to exploit idle CPU cycles to use DVS, but did not take into account application level information. In paper [32] more accurate prediction methods are explored to detect idle moments (like making use of IA and stochastic algorithms), including the idea of using application level information.

In [8] the authors applied DVS to a single server based on processor utilization to change frequency. Also for single servers, the technique of DVS with delaying requests are presented in [15]. The work in [16] presents a brief classification of energy saving approaches for clustered Web servers at processor level. Energy saving approaches aiming to the processor can utilize VOVO and/or DVS. VOVO must be performed by a coordinator, but DVS can be done independently (locally) or coordinately (cluster-wide). It gives rise to 5 categories: IVS (uses DVS locally), CVS (uses DVS coordinately), VOVO, VOVO-IVS and VOVO-CVS. These categories are explained more extensively in Section 3.3. This work also presents a VOVO-CVS approach that can better exploit energy saving opportunities, but it considers that frequency can be continuously tunned in a processor, which is not true in real life, thus impracticable. Moreover, it assumes all machines have the same configuration, and even the given solution set all the active machines to the same configuration. A good point is that it uses different threshold values to perform upgrading and downgrading reconfigurations in order to avoid constant cluster reconfiguration due to load variability. Its goal is to define different threshold values for upgrading and downgrade reconfigurations. For instance, suppose that the incoming workload increases over a threshold x that demands an upgrade in the cluster configuration. The cluster configuration will not change to the previous one as soon as the incoming workload decreases below x. Another value y ($y < x$) is defined for the downgrading reconfiguration. Important works for clustered servers [11, 10, 27, 29] presented similar ways of applying DVS and cluster reconfiguration, using threshold values, based on the utilization or the system load, to define the transition points and keep the processor frequencies as low as possible, with the fewer possible number of active nodes. All these works are summarized in the survey presented in [7].

The work in [32] evaluates DVS policies for power management in systems with unpredictable workloads. One simple technique, used in [43], is the application-oblivious prediction, based on periodical utilization monitoring. However, a good result is obtained because of the Web traffic behavior nature, since although the Web traffic presents high instantaneous fluctuation, it becomes constant as we increase the observation window. Therefore, by choosing the appropriate observation window the monitoring tends to present low fluctuation. The work in [43] gives a step further because they consider machines with discrete voltage scaling and the given solution allows the servers to operate at different frequencies. However, the homogeneous cluster assumption still holds. They propose the use of tables built off-line so that the application can consult it on-line to chose a cluster configuration based on the incoming workload monitored. It consumes

more memory but minimizes the processing overhead needed to make the maths on-line. They also show more complex techniques which attempt to predict performance needs by monitoring the arrival rate and CPU requirements of each request. Despite supporting more than one kind of request, all servers must be able to handle any of them, thus not supporting data partitioning.

The work in [27], besides combining DVS techniques with VOVO to save energy at processor level, analyzes other system's resources utilization and demand to perform VOVO reconfiguration. For instance, HD and memory demand are pondered when deciding whether to turn machines on or off. This work is particularly important because for the first time in the literature energy saving strategies for Web servers takes into account QoS constraints, which is fundamental in this environment as long as it characterizes a soft real time system. It also proposes the usage of Proportional-Integral-Differential (PID) feedback controller, a well-known tool of control theory that takes into account past history and speed of changes in demands. In this work, only continuous frequencies are considered.

The work in [33] considers DVS in QoS enabled Web server clusters, assuming load balancing in the nodes, which makes the power management problem symmetric across the cluster. Besides, the problem is modeled as a task scheduling, with a computation time and a deadline assigned to each request type. As long as the computation time can not be known in advance, the mean computational time is utilized, but the Web traffic's high variation is not taken into account. In [25] is presented a simple reconfiguration technique for a server cluster. Their model assumes a M/M/m queue and the energy consumption is calculated using the system expected waiting time. However, they do not consider heterogeneity, QoS constraints, nor the DVS capability.

In [19], a technique for the dynamic reconfiguration of the cluster is presented. It is based on a predefined metric, especially for energy/throughput. They use off-line techniques (simulated annealing) to obtain tables for each load level, to help in the dynamic reconfiguration of the cluster. Their model consider different types of requests and their approximate resource usage, including processor, network, disk, and software resources. In their model both requests and machines can differentiate from one another in all kind of resources. Therefore, a machine can have a faster processor and slower HD access and vice-versa. Similarly, a request type can be processor intensive and another request type I/O intensive. They model these characteristics about request and machines in matrices and solve the maths using optimization techniques due to their complexity. They use a

content-blind distribution with intra-cluster cooperation, so that different requests types can be directed to the most appropriate server to handle them.

Finally, the papers [43] and [31] are the most relevant to our work. The former propose the technique LAOVS (Load-Aware On-off with independent Voltage Scale), where the determination of the numbe of active nodes is made using a table calculated off-line, with a load discretization. For each load value, the best number of active nodes is obtained. The local power management is based on DVS using the same techniques presented in [32]. Since they assume that the servers are homogeneous, the local power management tends to stabilize having all servers operating at approximately the same frequency through simply distributing the incoming workload evenly among the active nodes. They do not consider heterogeneity.

In [31] they include heterogeneity and QoS restrictions. In this work the machines are sorted off-line accordingly to a policy and they are always brought on and off as needed in the same order. It avoids constant switching of the nodes, which is a expensive operation from both time and energy standpoints, and minimizes the complexity of the method. In order to determine when a node must be brought on or off they analyze off-line the cutoff point where it is more efficient to have more machines running slower than a few machines running faster. They show through experiments that once this point is crossed, having more machines will most often be the best option from this point on. The quadratic energy consumption increase with the frequency increase accounts for this behavior, thus reaching a point where it is more energy efficient to have more machines working slower.

Paper [17] was our first work in this field. In this work we also addressed both heterogeneity and QoS constraints, as the work presented in [31]. We modeled our system as N M/M/1 servers, where N is the number of server nodes. However, in our previous work we assumed only one kind of request and the Web traffic generation and service time followed a Exponential distribution. Now we consider a case where the Web traffic generation follows a Pareto distribution and more than one kind of request arrive at the cluster, but all of them having the same QoS constraint yet. Nonetheless, the PM technique still assumes the Exponential distribution of the Web traffic and we analyze the impact of it.

Chapter 5

System Model

In this chapter we will describe our work. The previous sections presented the background necessary to conduct it. Chapter 2 presented the state-of-art for Web servers architectures and Chapter 3 a PM overview. Basically, our work consists of saving energy in Web servers while meeting the QoS constraints of the application. Section 5.1 describes the Web server model that we adopted in our work; Section 5.2 formally presents the problem to be tackled; and finally, Section 5.3 describes the proposed solution.

5.1 Model

As explained in Chapter 2, there are several ways to build a server cluster. It is a complex infrastructure comprised of many machines, yet looking to the outside world as a single entity. The main functional concerns in such infrastructure were identified as being the request routing (see Section 2.2), request dispatching (see Section 2.3) and data placement (see Section 2.4). Furthermore, the service that the server cluster provides rules the architectural design deployed. For instance, a server that utilizes data partitioning to improve performance (service issue) must use content-aware request dispatching or allow request rerouting at server node level (architectural issue). The PM mechanism developed in this work targets a specific server cluster design that will be detailed in Section 5.1.1.

Finally, in order to validate the proposed mechanism we built a server cluster simulator and implemented the mechanism on it. For the sake of simplicity, the simulator does not reproduce a real life server with fidelity and some simplifying assumptions done are detailed in Section 5.1.2.

5.1.1 Cluster Features

In our work we adopt the cluster-based approach rather than the other architectures due to the presence of a central element which has total knowledge of the whole system, the *front-end*. In this architecture, the front-end is responsible for redirecting the incoming requests to the server nodes and we believe that this architecture is best suited to incorporate PM techniques because we can exploit the front-end knowledge of the whole system to coordinate the PM actions, thus providing better opportunities for overall system energy saving. Furthermore, as discussed in Chapter 2, the other architectures lack of efficient request routing mechanisms, and hence dispatching mechanisms, which are very important when performing PM control. Besides, local adaptations cannot fully exploit cluster heterogeneity when taking cluster-wide decisions, like bringing servers on and off, hence making the cluster-based approach the most reasonable option to take.

The cluster-based server architecture described in Section 2.1.1 (see Figure 2.1) is comprised of a front-end, a server layer and a back-end server layer, like database or application specific servers. However, here we consider only the front-end and server layers, disregarding the other layers. We do it because our goal is to save energy at the server layer and the front-end is present in the model only because of its crucial importance to the server operation.

We have also made an assumption about the service that the server will provide. We assume that all servers can handle any request, hence relieving the dispatching policy of content awareness. Requests are characterized by a probability of requesting I/O operation, a mean inter-arrival time, a mean service time for the CPU execution, a mean service time for the I/O execution and a deadline (QoS constraint). Although deadline specification is not a default requirement for request in the Web environment, a popular transactional Web e-Commerce benchmark called TPC-W [2] defines it as a server parameter to ensure QoS to the final user. As long as user's comfort comes to be of main concern nowadays, we adopt it in our model. However, we consider the response time as the interval between the arrival of a request at the cluster and its departure from the cluster. Nonetheless, the most adequate would be to consider the response time from the client standpoint. In this work we disregard any client-server communication overhead, focusing only on the server behavior.

We demand that at least a fraction β of the incoming requests must meet their deadlines, since a small delay in the response time will not compromise the system. We call β the *reliability factor*. And finally, there can be any number of requests that the server must

deal with, each of them having their own particular features, but all of them subject to the same QoS constraint and probability of requesting I/O nevertheless.

The front-end has total knowledge of the servers that comprise the cluster and their features. The front-end tasks are:

- to monitor the system workload;
- to compute the system utilization and energy consumption;
- to set up the cluster configuration, that is, to decide which servers must be active at a time and their operating frequencies;
- and, to act as a dispatcher for the dispatching policy.

First of all, let's differentiate load from utilization. Although both are related to the workload the system is handling, load is related to the amount of work it can handle at full power, whereas utilization is related to the amount of work it can handle in the actual configuration. For instance, suppose a server S that can handle 4 requests per second (rps) at full power but it is currently running at half-power (hence it can handle 2 rps). If the actual workload is 1 rps, the server load is said to be 25% (since it can handle 4 rps at full power) but the server utilization is 50% (since in the actual configuration it can only handle 2 rps).

There are several ways of calculating the utilization or load of a system. Since they cannot be calculated instantaneously, the solution is to calculate them in a recent past and consider that it will barely change for a while or to use some kind of prediction strategy. Usually prediction strategies are based on stochastic processes, exponential averaging or specific application level information, and their goal is to predict the future based on past information and some knowledge over the domain.

In our cluster we calculate them using the recent past result because the Web traffic presents low long-term fluctuation and the past behavior might be a good guessing for the future if the range of time we look into is big enough. Every server node computes its own utilization and reports it to the front-end, and since the front-end knows each server's actual configuration, it can locally compute the load for each server and for the overall system as well.

Let ρ_i be the utilization of server i, fop_i the current operating frequency of this server and max_freq_i its maximum operating frequency. Then, the normalized load for this server is computed as follows:

$$load_i = \frac{\rho_i \times fop_i}{max\_freq_i} \tag{5.1}$$

And the normalized system load is:

$$load = \frac{\sum_{i=1}^{N} \rho_i \times fop_i}{\sum_{i=1}^{N} max\_freq_i} \tag{5.2}$$

Furthermore, we calculate the system load as a moving average. Moving averages are used to smooth out short-term fluctuations, thus highlighting longer-term trends. In other words, the load calculation does not rely only on the last obtained result, but also take into account its behavior over a longer past. Since the Web traffic presents high instantaneous fluctuations but tends to stabilize over the time, the use of moving averages seems to be very suitable.

The determination of the best cluster configuration at a given time is the problem to be solved and will be discussed in Section 5.3. However, as soon as this decision has been made, it is the front-end role to send it to every server node, so that they can perform the necessary reconfigurations, if needed. The front-end informs to each server whether they must be on or off, and, in case they must be on, the operating frequency. Of course, servers chosen to be turned off must deal with its pending requests before and no further requests can be dispatched to them.

The dispatching algorithm adopted is the DWR policy (Dynamic Weighted Random). This policy randomly chooses a server to handle an incoming request upon its arrival, but the probability of any of the servers being selected to handle the request are not the same. Each server is assigned a weight that is linearly proportional to its operating frequency. Therefore, the faster a machine is, the more requests it can serve, and hence, the higher its weight will be. The weight of a server i (q_i) is computed as follows:

$$q_i = \frac{fop_i}{\sum_{j=1}^{N} fop_j} \tag{5.3}$$

However, the time a machine spends to serve a request is not only a function of its processor speed. Processor's architectural issues (like *Cycles Per Instruction* (CPI)) and memory (cache size, data access policy and so on) are some examples of factors that affects request serving. For this reason, the most appropriate approach would be to run benchmarks for each type of request on each server for each frequency that the server supports and extract a efficiency factor for the servers based on how fast they can execute

it. Furthermore, in the presence of more than one kind of request a machine can have different efficiency factor for each request type. It results from two factors:

- different requests have a different set of instructions;
- and, different architectures are likely to have different CPI for each type of instruction.

So, a reasonable efficiency factor should take into account this heterogeneity by considering the benchmark result and the percentage that a particular request contributes for the overall incoming workload. Let R be the number of request types in the system, p_r the percentage that request r contributes in the overall incoming workload and ef_{rif} the efficiency factor of server i operating at frequency f. Then, a server weight could be defined as:

$$q_i = \sum_{r=1}^{R} p_r * ef_{rif} \tag{5.4}$$

This expression is simply a weighted average that assigns higher weights to servers that are able to, on average, execute requests faster. However, as long as we only performed simulations, we were unable to obtain the benchmark results. That is why we use the processor speed as the only parameter to determine the machine efficiency. Nonetheless, it is the way we use this machine efficiency factor and not the way we obtain it that is the main concern in this work. So, henceforth we will consider the server's operating frequency as its efficiency factor, hence using Equation 5.3 to determine q_i.

5.1.2 Simulation Environment

In the previous section we described our target environment. The implementation of the work proposed here was done in a simulator that models this environment, rather than implemented in a real environment. The simulated server cluster is composed of only two layers: the front-end layer and the server layer. As mentioned earlier, our focus is on saving energy at server layer and the front-end layer is present in the model only because its crucial. However, as long as we are only interested in its functionality and not in its behavior, the front-end in our simulation is comprised of a single node that can perform any operation without any overhead. It is responsible for taking the traditional dispatching and routing decisions tasks, besides the newly PM tasks. In other words, for

the sake of simplicity, we assume that the front-end is composed of a single node that consumes no power and executes any task instantaneously. We assume this because our focus is on the server layer and taking into account front-end layer details would only complicate the model.

Real life machines take up some time to turn on and to shut down. There is also an inherent energy overhead, which makes this event even more important. In the simulator a node is assumed to shut down instantaneously and without any overhead, but to be brought on its time and energy overhead are computed. Neither the energy overhead nor the time overhead of this activity are negligible as they impact on energy saving and QoS provisioning, respectively. That is why VOVO must be applied wisely.

We characterize a server by two features: its processor speed and its energy consumption. Each node has a different energy consumption assigned to every operating frequency it supports and a energy consumption when idle. We assume that the energy consumption of a server during idle cycles is the same energy consumption of this server when operating at its lowest frequency. This is the same assumption done by [30], the first paper we based our work on, and we did it for comparison purpose.

Besides the cluster environment, there is another vital component to finalize the system: the Web traffic. The utilization of a real life trace would be more realistic, but in the current phase we simulated it as well. A request type r is characterized by a mean execution time $\frac{1}{\mu_r}$, a mean inter-arrival time $\frac{1}{\lambda_r}$ and a probability α_r of a incoming request being of this type. Nonetheless, we only have as system input parameter a inter-arrival rate λ and R probabilities α_r, where $\sum_{r=1}^{R} \alpha_r = 1$. Upon arrival of a request, it will be of type r with α_r probability, and it is a well-known result that in this case we obtain R random variables with $\lambda_r = \lambda * \alpha_r$ mean rate. The mean execution time parameter assumes that the request will be executed on the fastest machine at full power (max_freq).

The inter-arrival and execution times are random variables that could follow any distribution. At first, we adopted the Exponential distribution due to its well-known results and simple analytic treatment. However, it was identified that the Web traffic follows a Pareto distribution rather than Exponential one [13]. As detailed in this work, factors like client thinking time when accessing a Web page, data popularity, bursty request arrival (even when a client requests a single page, it may be composed of a lot of objects and any of them are retrieved in the same request), network delay between the time a client sends a request and the time it arrives at the cluster impact on the distribution characteristic of the inter-arrival time. A Pareto random variable, unlike

a Exponential one, has the property of representing the burstiness effect. Moreover, it presents a property called self-similarity, which means that a distribution shows the same statistical properties at many scales, and the same property is observed by empirical analysis of Web traffic traces. Therefore, for the inter-arrival time we adopted the Pareto distribution.

The requests are assumed to be CPU-bounded, which accounts for the PM targeting the processor. However, in order to investigate the I/O impact on QoS provisioning, requests can demand I/O operations with a given probability. In case a request needs to be I/O served, it releases the CPU and goes to another queue to wait for I/O service. Every request type has a mean I/O execution time $\frac{1}{\mu_{IO}}$ and the time it will spend on I/O follows a Exponential distribution, like that for the CPU execution time. After being served it goes to the CPU again to finish any further processing. However, in the simulator, a request, after being I/O served, takes up the processor even though it spends no time there. We do it so because drawing another CPU execution time would imply more processing from requests that demand I/O operation than from the ones that do not. It would jeopardize our goal of analyzing the impact of the I/O and queueing time on requests that require I/O. Moreover, since we are interested only in the time overhead when considering I/O operation, the energy consumed by the I/O device is disregarded.

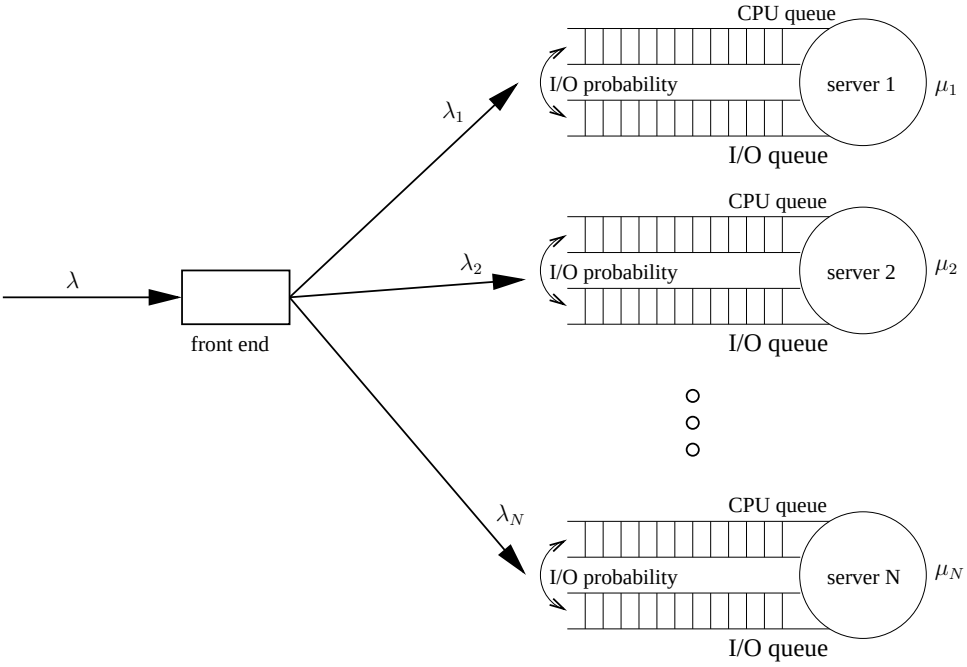


Figure 5.1: Cluster model.

5.2 Problem Definition

The problem to be solved is to establish, for each processor, whether it will be on or off and, in the former case, its operating frequency, subject to energy and timing restrictions. The solution to the problem is a vector $\{fop_1, fop_2, \ldots, fop_N\}$, where fop_i is the operating frequency of processor i. fop_i is set to zero if the processor i must be inactive.

As system input parameters, we inform the desired response time for the requests and the fraction β of requests that must be served meeting this deadline. The power consumption of a server i running at frequency f_j is represented as $p_i(f_j)$ (f_0 represent that processor i is turned off, and consumes no energy). With these assumptions, considering only the active servers, the aggregated power of the cluster is $P = \sum_{i=1}^{N} [\rho_i p_i(fop_i) + (1 - \rho_i)p_i(idle)]$, where $p_i(idle)$ is the power consumption of processor i when idle, and ρ_i is the utilization of this processor.

Our goal is to find a solution vector that minimizes the system's overall energy consumption while avoiding overflow in any of the system queues (in other words, a cluster configuration that is able to deal with the incoming workload) and that meets all QoS constraints. However, we must also be aware of the number of VOVO adaptations, because bringing a server on and off is a high cost operation and should not be done carelessly. Therefore, it is also part of the problem to minimize the server switching over time.

5.3 Solution

Basically, our goal is to save energy as much as possible. However, if we only desired to save energy we could simply shut down all servers. The point is that we desire to save energy, yet under two constraints:

- deal with the incoming workload;
- and, meet the QoS constraint.

Furthermore, the steps the cluster needs to take to satisfy these constraints demand processing, which is a critical resource under heavy load conditions. For this reason, we tackle this problem in the same way as proposed in the work [30]. That is, we build tables off-line whose entries are the workload that the system must handle and the content of each entry is the proper cluster configuration.

As we will see in the sequel, only with servers' processor specification it is possible to know how much workload the cluster can handle. Since the servers' processor specification is a information we have off-line, we can have it computed and stored into tables before the system runs. However, in order to meet the deadlines of the requests we might need to process them faster, but how fast we have to do it is the key point. Here is the point where we come up with two solutions: an on-line and an off-line approach. The off-line approach assumes that the Web traffic follows a Poisson process and makes the maths off-line based on the analytic treatment for M/M/1 servers. And the on-line approach monitors how fast requests are being served during execution time and make the decisions of cluster reconfiguration based on this monitoring.

In order to come up with a final solution, we attack each of the constraints individually. The next three sections describes how the items listed above are dealt with and jointly considered into a final solution.

5.3.1 Dealing with the Incoming Workload

First of all, the most important is that the solution deals with the workload. To solve this problem we implemented a backtracked algorithm that explores all possible cluster configurations and returns the one that minimizes the energy consumption while dealing with the workload. There are many mechanisms to tackle this problem, like Linear Programming (LP), Tabu Search, GRASP and so on, but we are not concerned about how the solution is found here, hence doing it the simplest way possible. Our problem can be mathematically defined as:

Minimize:

$$P = \sum_{i=1}^{N} [\rho p_i(fop_i) + (1 - \rho)p_i(idle)] \tag{5.5}$$

Subject to:

$$x = \frac{\sum_{i=1}^{N} fop_i}{\sum_{i=1}^{N} max\_freq_i} \geq workload \tag{5.6}$$

Where,

$$\rho = \frac{workload}{x} \tag{5.7}$$

Of course, equation 5.5 assumes that the utilization of the system (ρ) is perfectly balanced among all active nodes. Although it is not true all the time, on average DWR policy tends to balance the utilization among the servers due to the way we assign weights to the servers. It means that every active server will tend to be under the same utilization as the system ($\rho_i = \rho$ for every server i). That is why we do not use ρ_i in this equation.

5.3.2 Meeting the QoS Constraint

Although we can deal with the incoming workload, we still must guarantee that the time restriction is met. The time restriction imposes that at least a fraction β of the requests' response time have to satisfy their deadlines. Even when the cluster is able to deal with the incoming requests the speed at which the requests are dispatched might not be enough. However, how fast we have to go to provide the desired response time is the question to be answered. If we do it too fast we meet the deadlines but we waste energy, since we are providing more QoS than needed. On the other hand, if we do it too slow we do not meet the deadlines and compromise the system functionality.

In order to determine how fast or slow we have to operate is not a trivial issue. Unlike the previous case, where we could define the cluster configuration based only on the monitored system load, here only this information is not enough. If we wish to provide service with QoS, we must either monitor the response times on-line or assume some kind of distribution of the inter-arrival and response times to control. In the light of this situation, we propose two solutions. The former is an off-line approach that consists of assuming that the inter-arrival times and execution times follow an Exponential distribution. Although it is known that an Exponential distribution does not characterize a Web traffic, we adopt it because of its simplicity and due to the fact that closed expressions can be obtained for several system's characteristics. Nonetheless, as we have more than one kind of requests and they differ in mean execution time and arrival rate, depending on its type, it characterizes a Hyper-exponential distribution. The latter is a on-line approach that adjust the cluster configuration based on runtime load and QoS monitoring.

The next two sections will discuss these approaches in further detail.

5.3.2.1 Off-line Approach

Assuming that the requests' inter-arrival and service times on each server follow an Exponential distribution, they can be modeled as N M/M/1 servers. In this model, the response

time distribution follows an Exponential distribution with c.d.f. $F(t) = 1 - e^{(\mu-\lambda)t}$.

Nonetheless, in our case there are R different request types in the system with μ_r mean service rates and λ_r mean request arrival rates, which characterizes a Hyper-exponential distribution. However, when the random variables for the inter-arrival and service times of a server follow a Hyper-exponential distribution, the response time distribution can not be treated analytically anymore. As a Hyper-exponential distribution is obtained as a weighted average of different Exponential distributions [20], our proposed solution is to assume that the c.d.f. for the response time in this case follows a Hyper-exponential distribution whose c.d.f. is obtained as an weighted average of the c.d.f. of the response times of each request type.

$$F(t) = \sum_{r=1}^{R} \alpha_r * (1 - e^{-(\mu_r - \lambda_r)t}) \tag{5.8}$$

As we can see in Expression 5.8, when $R = 1$ we obtain the same expression as for the case of M/M/1 servers, as it should be. Moreover, as mentioned earlier, μ_r is defined based on the fastest processor and λ_r is the mean request arrival rate of request of type r in the system and not in a specific server. For a server i, the service rate is related to its current processor speed and the mean request arrival rate is related to the weight of the server in the dispatching policy:

$$\mu_{ri} = \frac{\mu_r * fop_i}{max\_freq} \tag{5.9}$$

$$\lambda_{ri} = \lambda_r * q_i \tag{5.10}$$

Therefore, the percentage of requests whose response time is below t at server i $(W_i(t))$ is:

$$W_i(t) = \sum_{r=1}^{R} \alpha_r * (1 - e^{-(\mu_{ri} - \lambda_{ri})t}) \tag{5.11}$$

And the percentage of requests whose response time is below t in the system $(W(t))$ is calculated as an weighted average of Expression 5.11 for each server i that comprise the cluster.

$$W(t) = \sum_{i=1}^{N} \left(q_i \times \sum_{r=1}^{R} \alpha_r \times (1 - e^{-(\mu_{ri} - \lambda_{ri})t}) \right) \tag{5.12}$$

We must find a cluster configuration that asserts that $W(t) \geq \beta$ and apply this restriction along with Equations 5.5 and 5.6 as a second constraint. This solution assumes the Hyper-exponential nature of the incoming workload and approximately calculates the mean response time of the system. λ_{ri} is obtained though incoming workload monitoring, t is the server parameter that represents the deadline of the requests, α_r is a parameter defined in Section 5.1.2 and q_i is informed by the dispatching policy. Therefore, we adjust the servers' operating frequency to change the μ_{ri} values and find a configuration that minimizes the overall energy consumption and asserts that $W(t) \geq \beta$.

Finally, as in this model a serve configuration obtained by Equation 5.12 is for an specific deadline t and reliability factor β, we must have a different table for each QoS constraint we want to establish.

5.3.2.2 On-line Approach

Alternatively, instead of determining an equation for the cumulative density function of the response times, we can monitor them on-line. Then, we can increase or decrease the the computational power of the system based on the monitoring result. If the fraction of request that meet their deadlines is below the reliability factor β, we must increase the computational power of the system; on the other hand, if it is too high, we must decrease the computational power.

Nonetheless, since we are dealing with a soft real-time system, a small number of deadline misses will not jeopardize the system operation and can be tolerated. On the other hand, meeting more deadlines than established should not be a problem. However, in this case it means that the system is running faster than needed, and hence wasting energy. Therefore, we tolerate it only if the fraction of deadlines met is barely over β.

We increase or decrease the computational power of the system by multiplying it by a *fix factors*, which may be different when performing increasing or decreasing adjustments. If we must increase the computational power of the system, we multiply it by an up_factor; on the other hand, if we must decrease the computational power of the system, we multiply it by an $down\_factor$. Additionally, the values that the up_factor and $down\_factor$ parameters assume might be constant or variable. In case we change it over time, we call it a *mobile fix factors*. The only policy we implemented to the *mobile fix factors* is that the

value they assume is based on how far we are from the desired QoS level. For example, if we wish a reliability of 70% but obtain 60%, we increase the server capacity in $\frac{10}{60}$ by making $up\_factor = 1 + \frac{10}{60}$; or else, if we wished a reliability of 60% but obtained 70%, we would deacrease the server capacity in $\frac{10}{70}$ by making $down\_factor = 1 - \frac{10}{70}$. This strategy aims to converge quicker to the goal (β), as it assigns high values to the up_factor and $down\_factor$ parameters when we are far from the goal and low values as we get closer to it.

It is a simple approach nevertheless, and more complex approaches, like the usage of PID controllers, may be investigated. However, here we are only interested in show an alternative solution to the off-line approach that presents no assumption on the Web traffic nature.

Finally, whenever we wish to increase or decrease the computational power of the system we do it by looking up the table built off-line based on Equations 5.5 and 5.6, since it provides the most energy-efficient configuration for every computational power we want to get.

5.3.3 Applying the Solution

So far, for a given load condition, we have the tools in hand to determine which configuration to set up the system. However, we must keep in mind that although the Web traffic tends to present long-term stability, it presents high short-term fluctuations that must be filtered in order to avoid intermittent cluster reconfigurations. These intermittent reconfigurations are not desired because they are not for free from both time and energy standpoints. We can smooth it out, for example, by increasing the load monitoring periodicity or its moving average window. In this work, we choose to tackle it by demanding that the decision to increase or decrease the current cluster capacity must be taken a given number of times in a row before consummating it. Although the decision must be always to increase or decrease, they do not need to be the same decision.

Finally, our last consideration pertain to how to use the solution presented here. Using the expressions to compute it on-line consumes much time that might compromise the the cluster service. Furthermore, these calculation must be done by the front-end, which can become a bottleneck under high load condition, where processing speed is critical. In order to avoid this processing overhead, we build a table off-line whose entries are the normalized discrete system load (that is, ranging from 0 to 1 in *granularity* steps) and the content of each entry is the cluster configuration to handle the respective load. The

granularity parameter must be chosen carefully, since low values incurs big tables that consume a lot of memory (a table has $\left\lceil \frac{1}{granularity} \right\rceil + 1$ entries) and high values lead to coarse-granularity control. Furthermore, if we adopt the off-line QoS solution, the time restriction is defined during table building phase, hence demanding a table for each pair of desirable response time t and reliability factor β.

Chapter 6

Results

In this chapter we describe the experiments we made in order to analyze the behavior of our PM mechanism. We considered two scenarios and through the experiments we adjust several configuration parameters to see how they impact on cluster performance. The configurable parameters in our PM mechanism are:

- **Load Periodicity** – Defines how often we calculate system load.
- **Load Moving Average Window** – Besides being calculated periodically, the load we attribute to the system at a given time is calculated as an average of the last system load values obtained through monitoring. We do it so because although the Web traffic presents high sudden fluctuations, it tends to stabilize over time. Therefore, we use this average to filter out these fluctuations, yet taking it into account. This parameter defines the number of elements we use in this average, or else, how long we look into the past.
- **Utilization Periodicity** – Defines how far in the past we have to look at when calculating the server utilization. As explained in Chapter 5, the server utilization is calculated as the fraction of the observation time that this server has been busy. The utilization periodicity defines both, how long is this observation time and how often we do the calculation.
- **Table Granularity** – For a given system load, we look up a table to obtain the proper system configuration. However, the system load is a continuous variable, hence being impossible to have an entry in the table for every value that the system load can assume; therefore, we discreteze the system load. The table granularity impacts on the table size, hence memory consumption, and on cluster reconfiguration due to the loss of accuracy.

- **Histogram Granularity** – A histogram is used only when performing on-line QoS monitoring and adaptation. The on-line approach utilizes a histogram to monitor the percentage of requests that miss their deadlines. Hence, we have to discreteze the time in order to make a histogram of the request's response time. When the time intervals are small, the histogram takes longer to compute; when it is larger we loose accuracy.

- **Request's Features** – Request's features encompasses the probability of requesting I/O operation, the mean inter-arrival time, the mean service time for CPU execution, the mean service time for I/O execution, the probability of requesting I/O and the deadline. Moreover, as the Web environment is a soft real-time system, it is not necessary to meet the deadlines of all requests. For this reason, the system is provided with a percentage β that defines how many of the requests must meet their deadline. These parameters impact on system load behavior and adjusting them provides several opportunities to analyze the cluster behavior under different load and constraint situations.

Given it is impossible to analyze all the combinations of the system parameters, we focus on the ones we judge the most important and watch their influence by varying them and fixing the other parameters. Nonetheless, these parameters and their behaviors are interconnected and the relationship among them should be further explored. The next sections describe the two scenarios we created, the experiments and the obtained results.

6.1 Scenario 1

For the experiments in the first scenario we assumed a server cluster with 8 machines, two of each type shown in table 6.1. This scenario is a simple version of the possible scenarios that the simulator can run. However, for the sake of simplicity, we exploit it to analyze the impact of some of the parameters presented above. So we only use the off-line approach here. We assume only one type of request, whose inter-arrival and execution times follow an Exponential distribution. The average inter-arrival time is dependent on the desired workload and the average execution time of each request is $0.01s$ (if executed at the highest frequency of the fastest processor). In each experiment, a total of 8×10^5 requests were simulated. The tables were built with a discrete load of granularity 0.01. The QoS requirement and reliability factor are set up accordingly to the specific experiment. Moreover, we used only the off-line approach to deal with the QoS restriction. Also, it

is assumed that load measurements are made every $1s$, that changes in the configuration are done after 5 consecutive load increases (decreases), and that the utilization ρ_i of each server is computed every $1s$ in the simulation. The moving average window for the load calculation has size 4. Finally, it should be mentioned that, in the simulator, the effect of switching on a server is taken into account. For the experiments described, switching on a server implies a $33s$ penalty and an additional 190J of energy consumption, as done in work [30].

Table 6.1: Processors specifications.

Processor	**Frequencies (MHz)**	**Respective power consumption (Watts)**
XScale	*idle*, 150.0, 400.0, 600.0, 800.0, 1000.0	0.355, 0.355, 0.445, 0.675, 1.175, 1.875
Power PC 750	*idle*, 4.125, 8.25, 16.5, 33, 99, 115.5, 132	1.150, 1.150, 1.369, 1.811, 2.661, 4.763, 5.269, 6.533
Power PC 1GHz 750GX	*idle*, 533, 600, 667, 733, 800, 867, 933, 1000	7.63, 7.63, 7.8, 7.97, 8.13, 8.30, 10.35, 12, 12.25
Power PC 405 GP	*idle*, 66, 133, 200, 266	0.74, 0.74, 1.09, 1.36, 1.58

To assess our method, we compared it to the one proposed in [31]. Figure 6.1 shows average power consumption of our method, for different QoS parameters and constant reliability factor equal to 0.8. For the method presented in [31] we only consider the case without QoS restriction because we want to compare both methods in the most energy efficient situation. For this comparison, we use a QoS parameter of $1s$ in our method because this value is high enough for the great majority of requests to be executed in time, resulting in the best energy efficiency. As we can see in the Figures, our method presents better results, even in some cases where there is a more tight response time restriction (see curves $QoS = 0.05s$ and $QoS = 0.07s$ for workload lower than 0.3). The reason for this is that in our method, the search algorithm finds the best configuration for each load level, while the method presented in [31] uses a predefined sequence of machines to be turned on and off, and this limits the optimization process. As expected, the smaller the QoS requirement, or the higher the workload, the processors will have to work faster to respond to the requests within the specified deadline, thus consuming more power. This behavior can be clearly seen in the figure. All curves, at some point, meet in the same line because the QoS constraint becomes so tight that the system must operate at full power whenever it is over a certain load. For example, for workloads greater than 0.7 in Figure 6.1, all the configurations with QoS response times of 0.02, 0.05, and 0.07 seconds

achieve the same power consumption, since the cluster operates at full power.

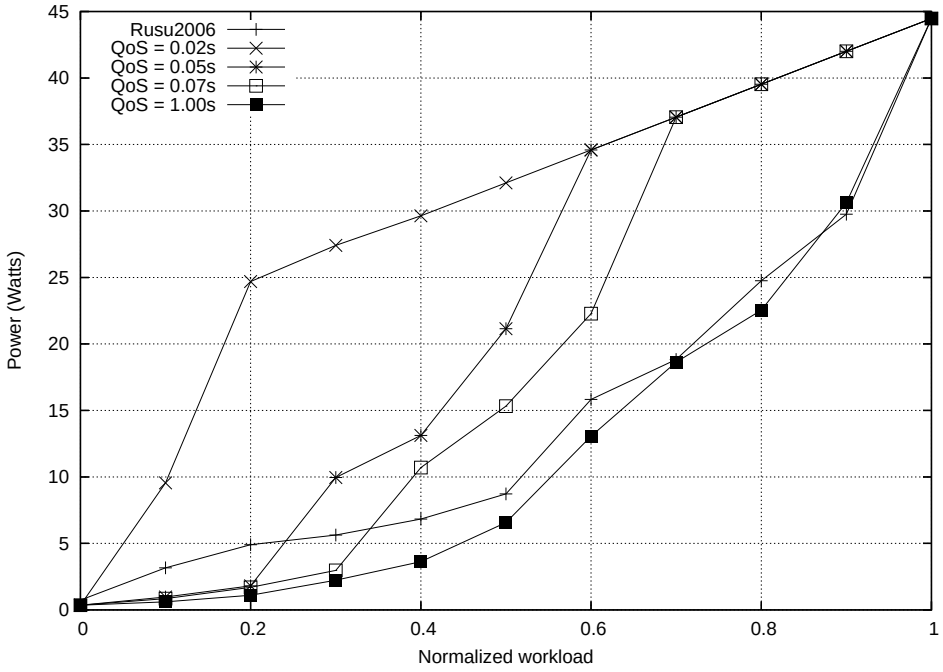


Figure 6.1: Cluster aggregate power for different QoS requirements, with $\beta = 0.8$.

Figure 6.2 shows the effect the workload granularity representation has on the energy saving method. As expected, fine granularities provide more energy saving than coarse ones, but at the expense of bigger tables and more sensitivity to cluster reconfiguration under unstable load (as usual in real life). This result was produced without taking into account QoS restrictions (simply setting the reliability factor to 0), as not to hinder the impact of the granularity factor. As we can see in this Figure, the impact of coarse granularities are hard. The greatest power consumption variation in percentage is 43%, when the load is 0.21 and the greatest variation in amount is 13.47 Watts, when the load is 0.91. Thus we decided to assume a table granularity of 0.01 throughout the experiments we performed.

Figure 6.3 shows the cluster power consumption considering different workloads. In this experiment, the QoS requirement was kept constant at $0.05s$. As it can be seen, the effect on power consumption of imposing a higher reliability factor is greater as the workload increases. In this situation, the system becomes rapidly saturated and starts to work at full power (best effort approach). For example, the curve for workload 0.8 becomes saturated for reliability 0.6, and the curve for workload 0.6 becomes saturated only for reliability 0.8.

Finally, Figure 6.4 shows the actual fraction of requests that have their time demands

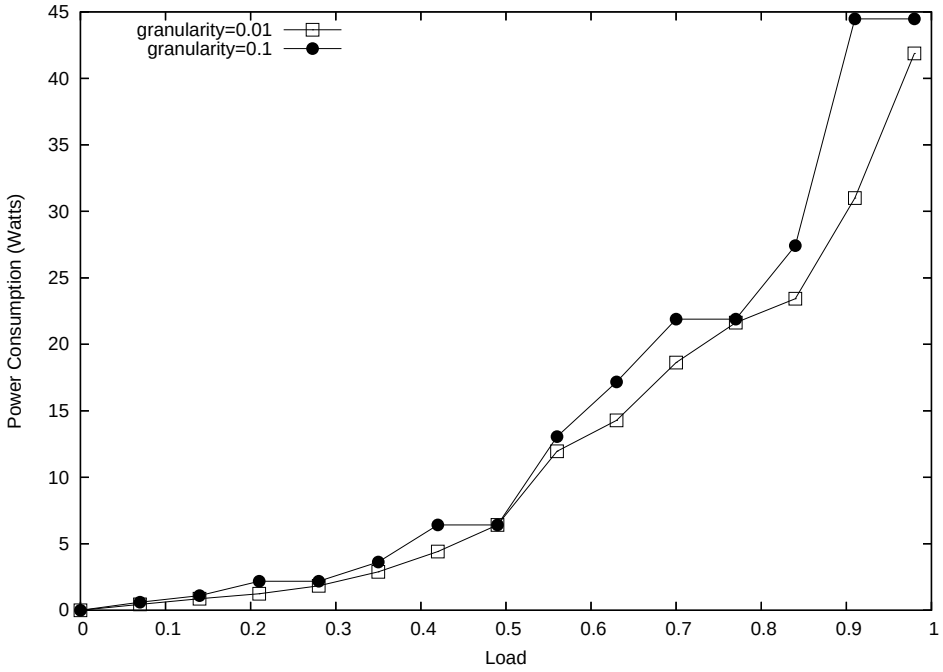


Figure 6.2: Power consumption versus load.

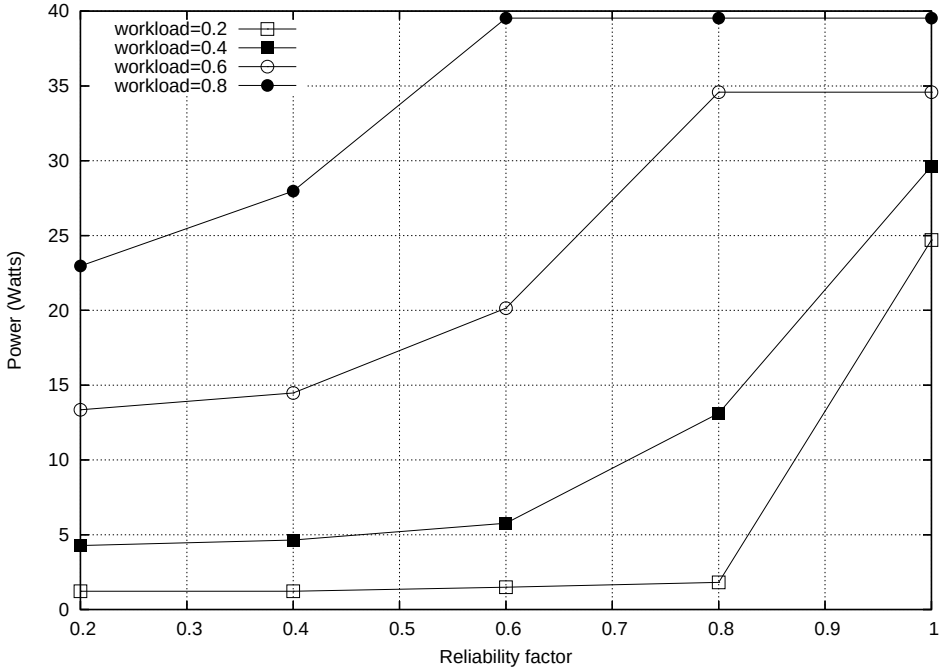


Figure 6.3: Power consumption of the server cluster as a function of β, for different workloads, with deadline $0.05s$.

satisfied, for different β and workloads. The previous Figures presenting the cluster aggregate power consumption were built using information generated during the table building phase, but this Figure was produced from results of the actual simulation process. Ideally,

this curve should follow the identity line (the identity line represents the situation where the actual percentage of deadlines met is exactly the same we impose with the parameter β), but it is easy to note that, as the workload increases, it is harder to ensure a high percentage of met deadlines (or even impossible, due to cluster saturation). As can be seen, with β increasing, the curves depart from the identity line, and, eventually, saturate (meaning that the system cannot satisfy the QoS requirement at the specified reliability level, shown as points below the identity line). Additionally, due to the discrete frequencies of the processors, as the workload and factor β decrease, the curves will bend upward. This is because the processors have a minimum operating frequency and the requests are being processed at a higher frequency than necessary. This can be clearly seen in the step-like curve for workload 0.2.

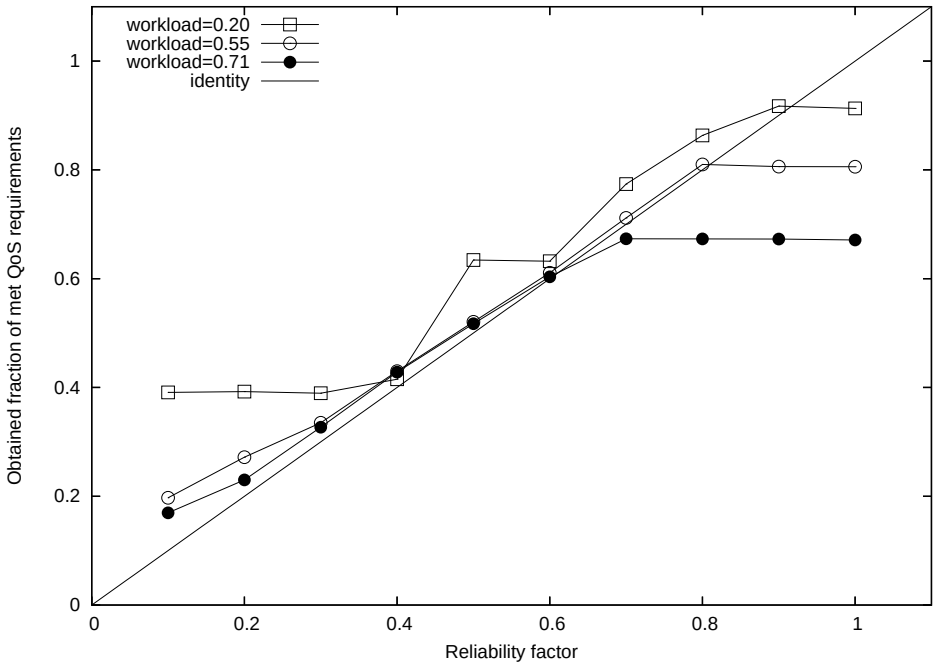


Figure 6.4: Actual fraction of QoS restrictions met, as a function of β, for different workloads, with deadline $0.05s$.

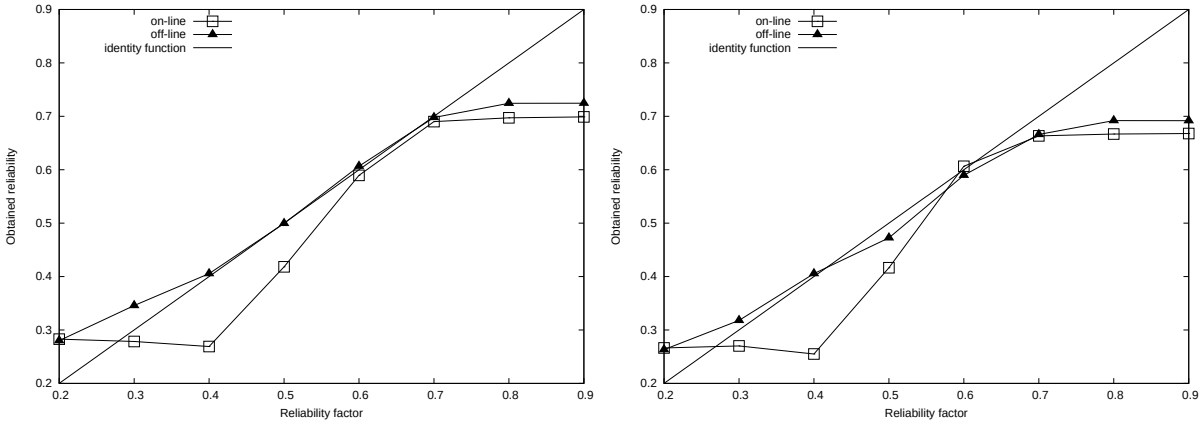
6.2 Scenario 2

For the second scenario we assumed a server cluster with 6 machines, three XScale and three Power PC 1GHz 750GX (see table 6.1). We consider only these machine types because the four machines presented in table 6.1 are too heterogeneous among them, what has a negative effect on the system behavior. For example, note that machine Power PC 405 GP is much slower than the other machines. Therefore, at the point where

the servers of this type must be brought on a small increase in the workload might imply another server of other type to be brought on, since the Power PC 405 GP machines can not deal with much load. Hence, the VOVO negatively impacts on the system behavior. This disparity in machine's performance also impacts on QoS provisioning. Additionally, we increase the scenario complexity with I/O operations, Web traffic following other type of distribution than the Exponential and more request types. Therefore, now we have a more unstable workload and cluster reconfigurations will be needed more often than in the previous scenario.

We assume two types of requests this time, a and b , whose execution times follow an Exponential distribution and inter-arrival times are experiment dependent. Each one comprises 50% of the total incoming workload of the system. The average inter-arrival time is dependent on the desired workload and the mean execution time of the requests a and b are respectively $0.01s$ and $0.005s$ when executed at the highest frequency of the fastest processor. The I/O mean execution time for them are $0.1s$ and $0.05s$, respectively. Each experiment simulates two hours of cluster operation. The tables were built with a discrete load of granularity 0.01 and the histogram with granularity of $0.001s$. The QoS requirement, reliability factor and probability of requesting I/O are set up accordingly to the specific experiment. The I/O intensity is always low because our method targets the CPU and we assumed from the scratch only CPU-bounded requests for this reason. Nonetheless, it is reasonable to assume that a request might need some local information to complete its execution and that this information can be cached. So we use a small I/O probability to simulate the cache miss ratio behavior and analyze the impact of it on the system. It simply makes no sense to assume high I/O demand as we have control only over the CPU. Moreover, we use both on-line and off-line approaches to deal with the QoS restriction, comparing their performance. The QoS restriction is always kept too loose in the experiments because otherwise only setting up the cluster to deal with the incoming workload would allow a great number of requests to meet their deadlines, hence spoiling the evaluational purpose of the experiments done. Additionally, it is assumed that load measurements are made every $4s$, that changes in the configuration are done after 10 consecutive load increases (decreases), and that the utilization ρ_i of each server is computed every $1s$ in the simulation. The moving average window for the load calculation has size 4. The effect of switching on a server remains the same as before. And finally, unlike in the previous scenario, all figures presented in this scenario were obtained from simulation results computed as an average of 10 experiments.

Figures 6.5 and 6.6 show the actual fraction of requests that have their time demands

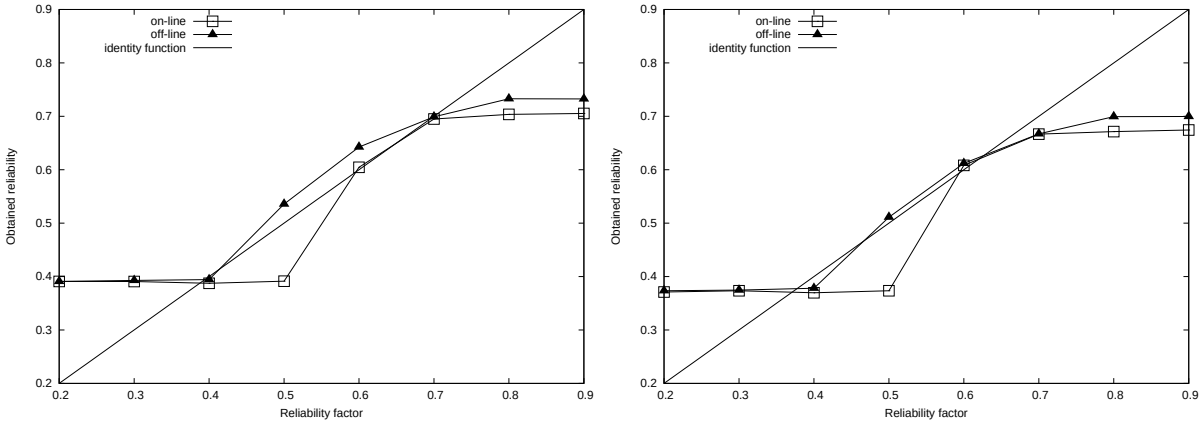


(a) (b)

Figure 6.5: Web traffic following an Exponential distribution with mean inter-arrival time $0.02s$ and deadline $0.01s$. Figure 6.5(a) is the behavior without I/O and Figure 6.5(b) with 5% of probability of requesting I/O operation.

satisfied, for different β and workloads. Figure 6.5 depicts an environment where the Web traffic has inter-arrival time $0.02s$ and Figure 6.6 an environment with inter-arrival time $0.03s$. Therefore, the former is under a heavier load than the latter. The effect is clearly seen in the earlier bend upward in the base of the curves of Figure 6.6 because the load is too low and the active processors are already operating at their lowest frequency. As we can see in Figure 6.6, when the system load is low at least 40% of the deadlines are met, even for lower reliability factors. On the other hand, as we can see in Figure 6.5, when the system is under a heavier load the fraction of requests that meet their deadlines keeps decreasing as β decreases below 40%, which demonstrates that the system is decreasing its capacity, and hence saving energy. The energy consumption of the experiments depicted in Figures 6.5(a) and 6.6(a) is shown in Figures 6.10(a) and 6.10(b), respectively, in the curves labeled *on-line exponential* and *off-line exponential* in both graphics. We can see that lower load provides better savings, but the saving looks small because the set of machines that are turned on are very economic (see the processor's specification for the machines used in this scenario in Table 6.1).

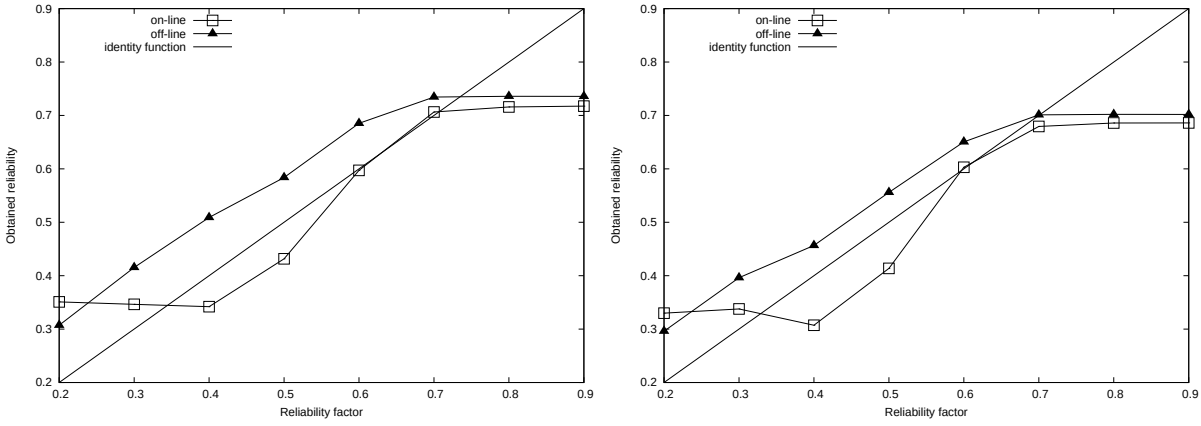
In both Figures 6.5 and 6.6 we show on the left side of the Figures an execution of the experiment assuming no I/O operation and on the right side an execution of the experiment assuming 5% of probability that an incoming request demands an I/O operation. As we can see in both cases, when assuming I/O the off-line curve only shifts downward, while the on-line curve reacts to the overhead imposed by the I/O activity and tries to keep up with the QoS (compare, for example, the on-line curve on Figures 6.5(a) and 6.5(b) when $\beta = 0.6$). However, as the I/O probability is too low, it is not able to jeopardize



(a) (b)

Figure 6.6: Web traffic following an Exponential distribution with mean inter-arrival time $0.03s$ and deadline $0.01s$. Figure 6.6(a) is the behavior without I/O and Figure 6.6(b) with 5% of probability of requesting I/O operation.

the obtained result. As we can see in the Figures, the off-line curve is always very close to the identity line. Nonetheless, the off-line approach presents good results because the model assumes an Exponential distribution for the inter-arrival times and that is exactly the same distribution that we are using in the Web traffic generator in these experiments. Yet, this result validates the assumptions and simplifications that we have made when building the expressions that enabled us to treat this problem analytically. However, it is well-known that it is not true in real life. Therefore, in the next experiments we analyze how the system behaves assuming that the Web traffic follows a Pareto distribution.

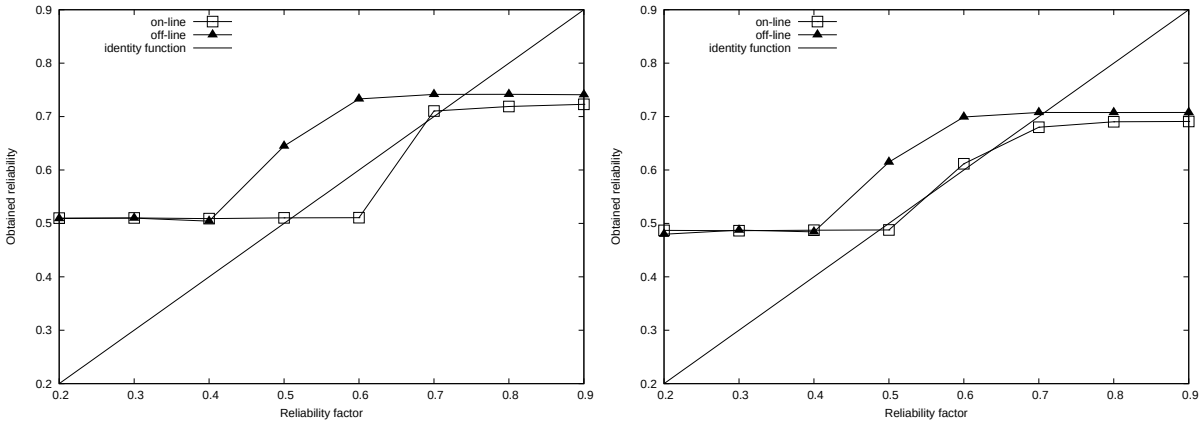


(a) (b)

Figure 6.7: Web traffic following a Pareto distribution with mean inter-arrival time $0.02s$ and deadline $0.01s$. Figure 6.7(a) is the behavior without I/O and Figure 6.7(b) with 5% of probability of requesting I/O operation.

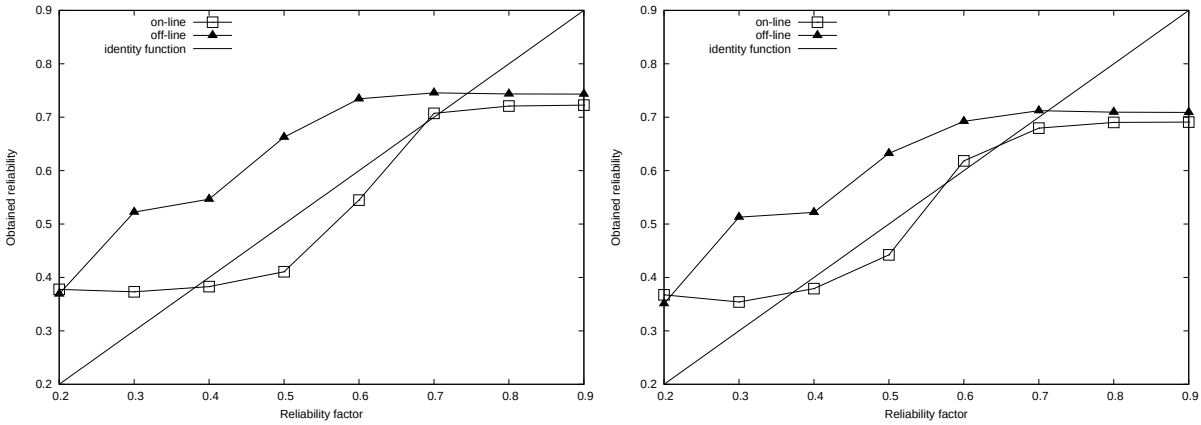
As we can see in Figure 6.7, now that we assume a Pareto distribution of the inter-arrival times the off-line curve is not so close to the identity line anymore. However, it

still presents a good behavior, since it is not so far from it and, besides, it is over the identity line, which means that we are not missing more deadlines than we are supposed too. Furthermore, when we assume I/O demand the downward shift of the curve indicates a better result. Furthermore, as we can see in Figure 6.10(a) in the curves *on-line pareto* and *off-line pareto*, the off-line approach consumes slightly more energy but postpones the energy peak when compared to the on-line approach. The only difference among the environments depicted in Figures 6.7, 6.8 and 6.9 is the load intensity. We consider these three situations to demonstrate that the behavior shown is not an isolated case, remaining the same even under different workload conditions.



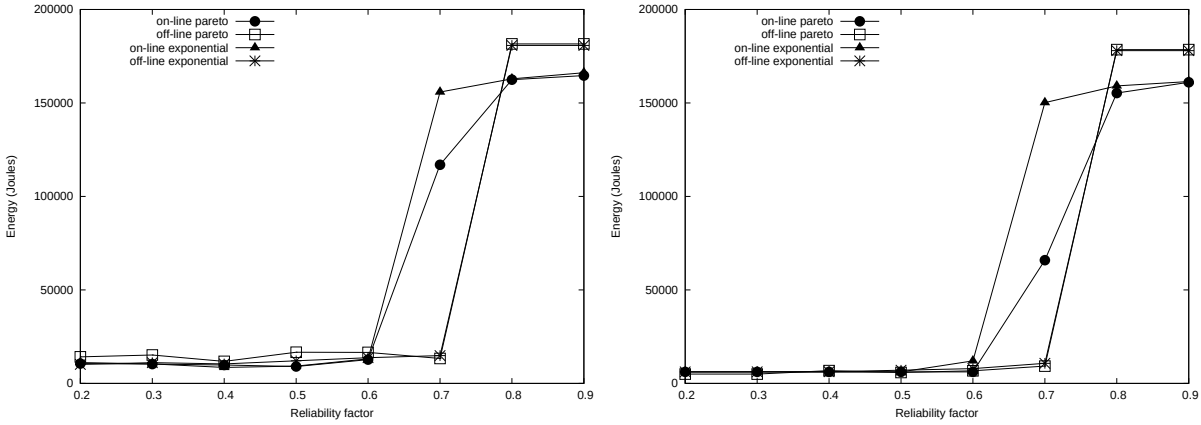
(a) (b)

Figure 6.8: Web traffic following a Pareto distribution with mean inter-arrival time $0.03s$ and deadline $0.01s$. Figure 6.8(a) is the behavior without I/O and Figure 6.8(b) with 5% of probability of requesting I/O operation.



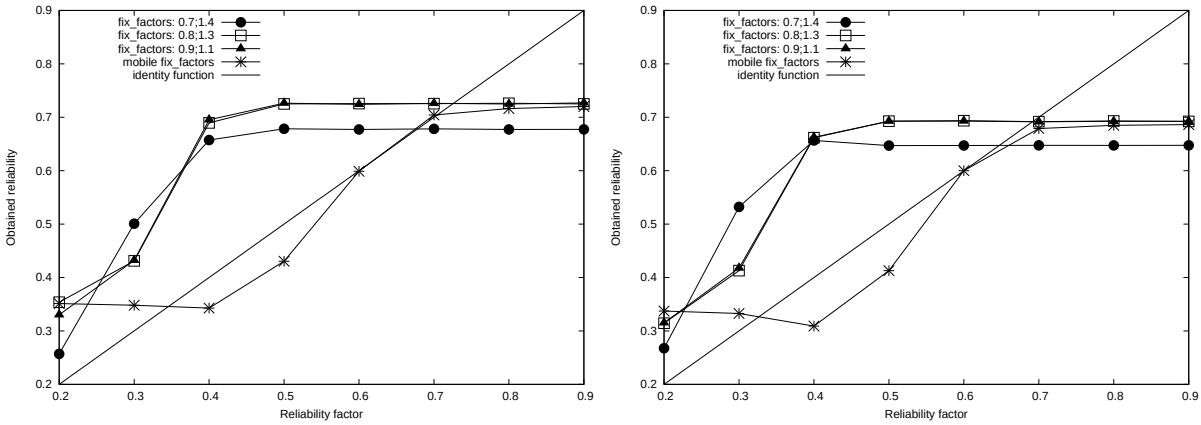
(a) (b)

Figure 6.9: Web traffic following a Pareto distribution with mean inter-arrival time $0.04s$ and deadline $0.01s$. Figure 6.9(a) is the behavior without I/O and Figure 6.9(b) with 5% of probability of requesting I/O operation.



(a) (b)

Figure 6.10: Energy consumption of the system assuming that there is no I/O operation. Figure 6.10(a) assumes a Web traffic with mean inter-arrival time $0.02s$ and deadline $0.01s$. Figure 6.10(b) assumes a Web traffic with mean inter-arrival time $0.03s$ and deadline $0.01s$.

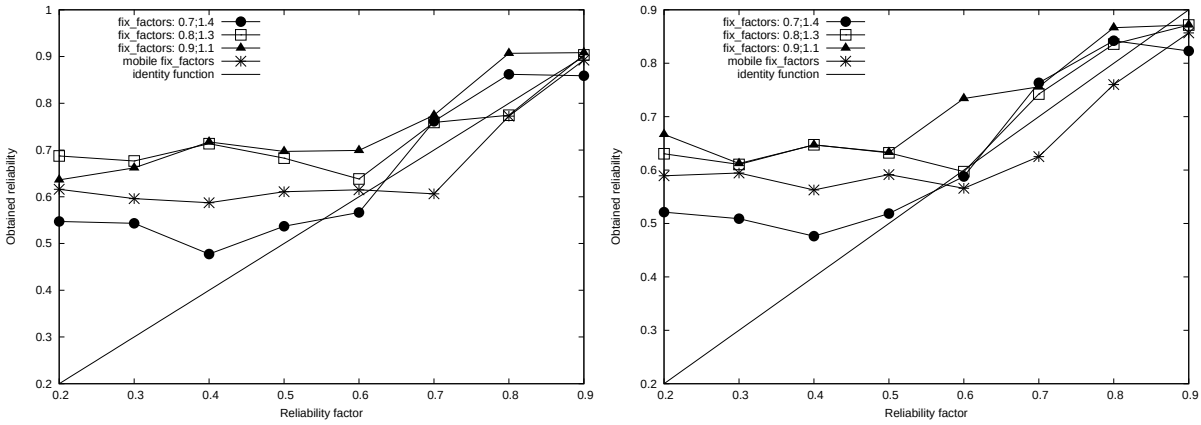


(a) (b)

Figure 6.11: Web traffic following a Pareto distribution with mean inter-arrival time $0.02s$ and deadline $0.01s$. Figure 6.11(a) is the behavior without I/O and Figure 6.11(b) with 5% of probability of requesting I/O operation.

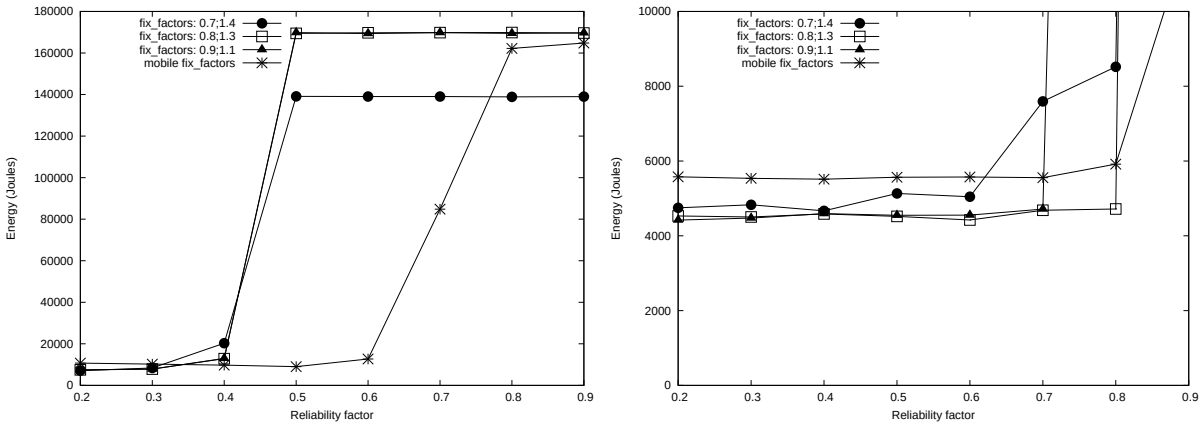
In the sequel, we analyze the influence of the *fix factors* parameters in the on-line approach. Figure 6.11 shows the curves for different values of up_factor and $down\_factor$ parameter and a curve for the *mobile fix factors* using the policy we presented in Section 5.3.2.2. As we can see analyzing this Figure and Figure 6.13(a), the mobile approach far outperforms the other techniques because it saves more energy and its curve for deadlines met in Figure 6.11 is very close to the identity line. However, it is not how it behaves always. During our experiments we also observed situations like the one depicted in Figure 6.12. Here the mobile approach does not perform so bad but loses to the constant approach with fix factors 0.7 and 1.4. Actually, when taking only the constant approaches into account, this option behaves better, even though one could think that high values would generate

instability as they try to adjust the server capacity in coarse-grained steps. Nonetheless, as we can see in Figure 6.13(b), the mobile approach still postpones the energy peak, but it slightly consumes more energy than all the other approaches for $\beta \leq 0.6$.



(a) (b)

Figure 6.12: Web traffic following a Pareto distribution with mean inter-arrival time $0.04s$ and deadline $0.02s$. Figure 6.12(a) is the behavior without I/O and Figure 6.12(b) with 5% of probability of requesting I/O operation.



(a) (b)

Figure 6.13: Energy consumption of the system assuming that there is no I/O operation. Figure 6.13(a) assumes a Web traffic following a Pareto distribution with mean inter-arrival time $0.02s$ and deadline $0.01s$. Figure 6.13(b) assumes a Web traffic following a Pareto distribution with mean inter-arrival time $0.04s$ and deadline $0.02s$.

So far we have assumed only I/O with 5% of probability because we judged it a reasonable cache miss ratio. Keep in mind that we are managing energy at server level, where it is more reasonable to assume a more intensive processing than data retrieving. If a request depends on massive data retrieving it should be delegated to another cluster level responsible for it. However, we investigated the impact of a heavier I/O demand on our methods and it is depicted in Figure 6.14. We analyzed situations of 25% and 40% of

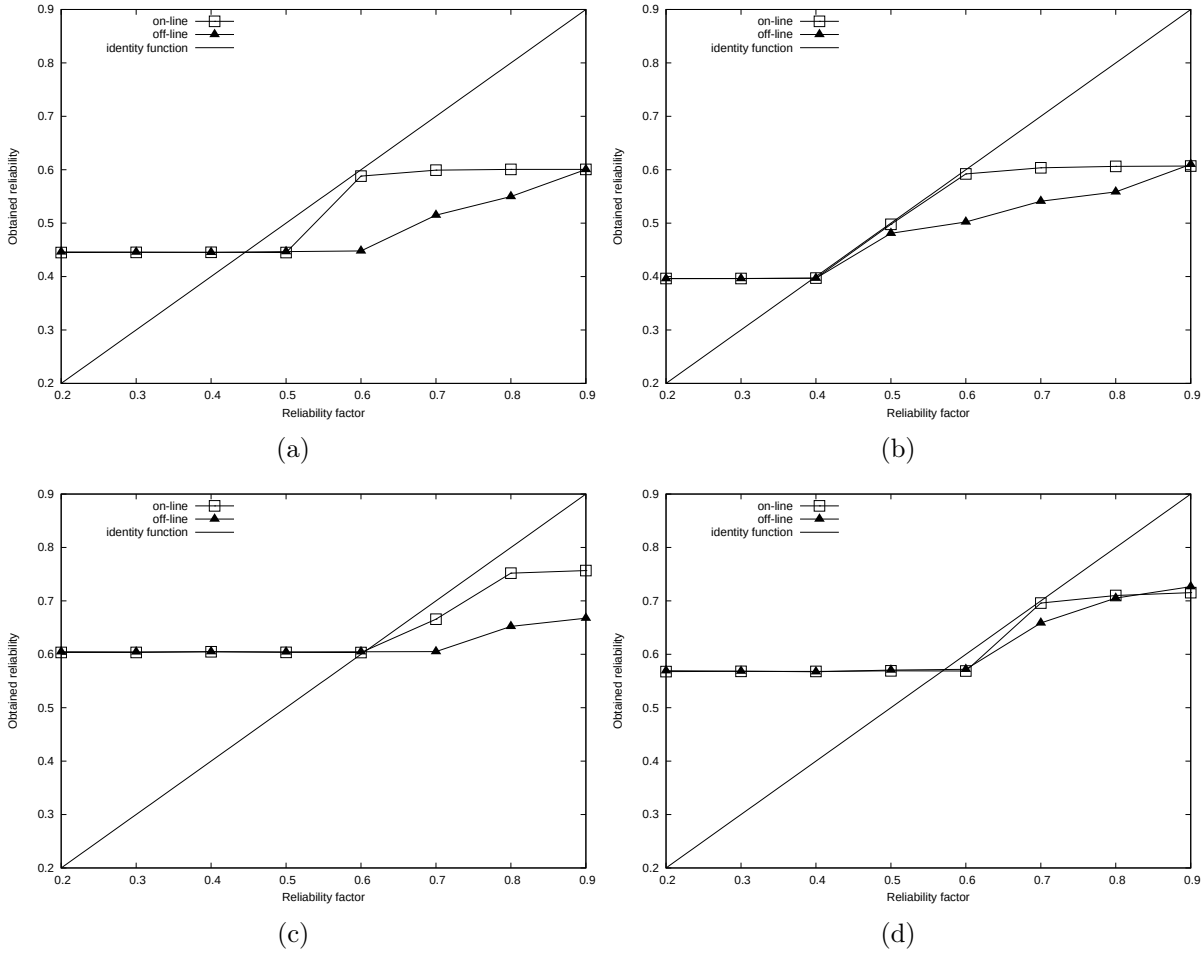


Figure 6.14: Web traffic following a Pareto distribution. Figure 6.14(a) with mean inter-arrival time $0.03s$, deadline $0.02s$ and 40% of probability of requesting I/O; Figure 6.14(b) with mean inter-arrival time $0.05s$, deadline $0.02s$ and 40% of probability of requesting I/O; Figure 6.14(c) with mean inter-arrival time $0.05s$, deadline $0.05s$ and 40% of probability of requesting I/O; and finally, Figure 6.14(d) with mean inter-arrival time $0.03s$, deadline $0.02s$ and 25% of probability of requesting I/O.

probability of I/O demand, since higher than this is beyond the control of our methods. We can only speed up processing to try to make up for the I/O serving, but as long as the I/O time is much larger than the processing time and the demand increases too much there is nothing left to do.

As we can see in the four graphics in Figure 6.14, when using the on-line approach the method is able to react and meet the QoS constraint. The four experiments vary in load intensity, deadline and I/O intensity. All curves are a straight line when β is low because the load is to low and the active machines are already running in the lowest frequency. But having a very low workload is the only way that the processing speed up can make up for the I/O delay, hence meeting the imposed deadline. However, notice that in graphics 6.14(a) and 6.14(b) the deadlines cannot be met for $\beta > 0.6$) even though the

workload is very low, reflecting that this kind of environment is beyond the control of a pure CPU aware PM strategy.

Chapter 7

Conclusion and Future Work

In this work we presented a technique to achieve minimization of energy consumption and adequate response times for soft real-time applications in Web server clusters. During system operation, accordingly to the offered load, to the QoS restriction (response times) and to the predefined proportion of requests that should have their deadlines met (soft real-time criterion), processors are switched on and off, and the ones active are set to an optimal frequency of operation. In our simulations and comparison to other proposals, the technique here described showed promising results.

We presented a solution based on analytic treatment for queueing system assuming that the inter-arrival and service time of the requests follow an Exponential distribution, because of the closed results that can be obtained for this model. Although the pattern for Web server requests does not follows a Poisson process, we show that it is a good approximation even though its simplicity. The Web traffic in our simulations comes from a generator we built rather than from real Web traces. In a first moment we generated a Web traffic following a Poisson process in order to validate the simulator, obtaining the expected results. Then, we extended the Web traffic generator, respecting the self-similarity nature of the requests arrival distribution by generating inter-arrival times that follow a Pareto distribution. Only the execution time remains following an Exponential distribution. For this latter case, the Exponential assumption seems to be a good approximation as we obtain good results in our experiments.

Nonetheless, we derive the analytic expressions used in our model through approximation, because when we have more than one type of requests the model cannot be easily treated analytically. Therefore, as the number of kinds of requests increases the quality of the result might degrade and this behavior must be analyzed more extensively.

In order to solve the optimization problem we used a dynamic programming approach

that scan all possibilities to look for the best result. However, this approach becomes unfeasible as the number of nodes increases because the execution time increases exponentially. Nonetheless, it is not hard to realize that a heuristic for this problem can be easily obtained, but we did not explored it because we focused on analyzing the feasibility of our analytic treatment to save energy yet providing QoS. We also compare the analytic treatment with an on-line approach for providing QoS and we notice that the analytic treatment presents a good performance. Of course, our on-line mechanism is very simple and more sophisticated mechanisms, like the use of, for example, PID controllers, might provide good results and should be further investigated.

Furthermore, we investigated how I/O operation due to cache miss ratio may impact on the system performance, concluding that it does not negatively impact on system behavior. We simulate the cache miss ratio in our experiments by using low I/O demand rates, but we compute only the time overhead of this operation, disregarding the energy onervhead. However, as the I/O demand increases, there is nothing our method can do as we operate only at CPU level. Under this situation, other mechanisms must be investigated. Nonetheless, the system must be tested more extensively in order to determine whether the Poisson assumption is really a good approximation or not. These tests should take into account different machine types, clusters of different sizes and analyze the system behavior as we vary the parameter presented in the beginning of Chapter 6.

Although we only targets the CPU, a typical Web server is comprised of other energy critical elements, like storage devices, memories and other network components that might provide energy savings and contribute to attain a better overall energy saving. Besides, although we apply this PM mechanism to a Web server cluster, it is reasonable to think about small adjustments to adapt it to other types of clusters, like the GRID environment. However, when considering other types of clusters we must have their features properly analyzed in order to exploit energy savings capabilities.

Finally, the model presented was only assessed by simulation. Implementing it on a real cluster, would provide more realistic results about the mechanism efficiency. However, our work clearly shows the improvements in the reduction of energy consumption that can be obtained even using simplifying assumptions.

References

[1] Determining total cost of ownership for data center and network room infrastructure. http://www.apcmedia.com/salestools/CMRP-5T9PQG_R3_EN.pdf. American Power Conversion, APC White Paper #6.

[2] Transaction processing performance council. http://www.tpc.org/tpcw/.

[3] Advanced configuration and power interface specification. http://www.teleport.com/acpi/, August 2003. Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd. and Toshiba Corporation's report, White Paper.

[4] Intel's high-k/metal announcement. http://www.intel.com/technology/silicon/power/transistor.htm#Section3, June 2003.

[5] Nevine AbouGhazaleh, Bruce Childers, Daniel Mossé, and Rami Melhem. Energy conservation in memory hierarchies using power-aware cached-dram. University of Pittsburgh, Department of Computer Science, Technical Report: TR-05-123, http://www.cs.pitt.edu/ñevine/papers/TR-PACDRAM.pdf.

[6] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, pages 22–28, March-April 2003.

[7] Ricardo Bianchini and Ramakrishnan Rajamony. Power and energy management for server systems. *IEEE Computer*, 37(11):68–74, 2004.

[8] Pat Bohrer, Mootaz Elnozahy, Mike Kistler, Charles Lefurgy, Chandler McDowell, and Ramakrishnan Rajamony. The case for power management in web servers. In Robert Graybill and Rami Melhem, editors, *Power Aware Computing*, pages 261–289. Kluwer Academic Publishers, 2002.

[9] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[10] Jeff Chase, Darrell Anderson, Prachi Thakur, and Amin Vahdat. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles*, pages 103–116, Banff, Alberta, Canada, October 2001.

[11] Jeff Chase and Ron Doyle. Balance of power: Energy management for server clusters. In *Eighth Workshop on Hot Topics in Operating Systems*, page 165. IEEE Computer Society, May 2001.

[12] Wissam Chedid and Chansu Yu. Survey on power management techniques for energy efficient computer systems. Department of Eletrical and Computer Engineering, Cleveland State University, 2121 Euclid Avenue, SH 332, Cleveland, OH 44115.

[13] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transaction on Networking*, 5(6):835–846, December 1997.

[14] Steven Dropsho, Volkan Kursun, David H. Albonesi, Sandhya Dwarkadas, and Eby G. Friedman. Managing static leakage energy in microprocessor functional units. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 321–332, Istanbul, Turkey, 2002. IEEE Computer Society Press.

[15] Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, USA, March 2003.

[16] Mootaz Elnozahy, Mike Kistler, and Ram Rajamony. Energy-efficient server clusters. In *Second Workshop on Power Aware Computing Systems*, pages 179–196, Cambridge, MA, USA, February 2002.

[17] Raphael Guerra, Luciano Bertini, and J.C.B. Leite. Improving response time and energy efficiency in server clusters. In *VIII Workshop de Tempo Real*, pages 45–52, Curitiba, PR, Brazil, May 2006.

[18] Raphael Guerra, Luciano Bertini, and J.C.B. Leite. Managing energy and quality of service in heterogeneous server clusters. In *XXXII Conferencia Latinoamericana de Informática*, Santiago, Chile, August 2006.

[19] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 186–195, Chicago, IL, USA, June 2005.

[20] Leonard Kleinrock. *Queueing Systems*, volume 1. John Wiley and Sons, 1975.

[21] Heungki Lee, Gopinath Vageesan, and Eun Jung Kim. Power and locality aware request distribution. Technical report, Texas A&M University, Texas, USA, July 2004.

[22] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commercial servers. *IEEE Computer*, 36(12):39–48, December 2003.

[23] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A quantitative analysis of disk drive power management in portable computers. Computer Science Division, University of California, Technical Report, http://www.eecs.berkeley.edu/Pubs/TechRpts/1993/CSD-93-779.pdf, 1994.

[24] L. Li, I. Kadayif, Y-F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and A. Sivasubramaniam. Leakage energy management in cache hierarchies. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 131–140. IEEE Computer Society, 2002.

[25] Chia-Hung Lien, Ying-Wen Bai, Ming-Bo Lin, and Po-An Chen. The saving of energy in web server clusters by utilizing dynamic server management. In *12th IEEE International Conference on Networks*, volume 1, pages 253–257, Singapore, November 2004.

[26] Samuel K. Moore. Winner multimedia monster. *IEEE Spectrum*, 1(20):18–21, January 2006.

[27] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In *Compilers and Operating Systems for Low Power*, pages 75–93. Kluwer Academic Publishers, 2003.

[28] Ken Popovich. New Intel technology may clear way for faster chips. http://www.eweek.com, November 2001.

[29] Karthick Rajamani and Charles Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 111–122, Austin, Texas, USA, March 2003.

[30] Dakai Zhu Daniel Mossé Ruibin Xu, Cosmin Rusu and Rami Melhem. Practical energy-efficient policies for server clusters. In *6th Brazilian Workshop on Real-Time Systems*, Gramado, Rio Grande do Sul, Brazil, May 2004.

[31] Cosmin Rusu, Alexandre Ferreira, Claudio Scordino, Aaron Watson, Rami Melhem, and Daniel Mossé. Energy-efficient real-time heterogeneous server clusters. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 418–428, 2006.

[32] Cosmin Rusu, Ruibin Xu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for request-driven soft real-time systems. In *16th Euromicro Conference on Real-Time Systems*, pages 175–183, Catania, Italy, July 2004.

[33] Vivek Sharma, Arun Thomas, Tarek F. Abdelzaher, Kevin Skadron, and Zhijian Lu. Power-aware QoS management in web servers. In *24th IEEE Real-Time Systems Symposium*, pages 63–72, Cancun, Mexico, December 2003.

[34] Piia Simonen, Aarne Heinonen, Mika Kuulusa, and Jari Nurmi. Comparison of bulk and SOI CMOS technologies in a DSP processor circuit implementation. In *The 13th International Conference on Microelectronics*, pages 107–110, October 2001. Digital and Computer Systems Laboratory, Tampere University of Technology, Finland.

[35] Mircea R. Stan and Kevin Skadron. Power-aware computing. *IEEE Computer*, 36(12):35–38, December 2003.

[36] Andrew Tanenbaum. *Modern Operating Systems*. Prentice Hall, England, 1992.

[37] Patrick Thibodeau. Congress begins push for energy-efficient servers. http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9001753, July 2006. ComputerWorld Government.

[38] Arman Vassighi and Manoj Sachdev. *Thermal and Power Management of Integrated Circuits*. Springer, 2006.

[39] Vasanth Venkatachalam and Michael Fran. Power reduction techniques for microprocessor systems. *ACM Computing Surveys*, 37(3):195–237, September 2005.

[40] Mark Weiser, Brent Welch, and Scott Shenker. Scheduling for reduced cpu energy. In *Operating Systems Design and Implementation*, pages 13–23, 1994.

[41] Jane Wright. Blades have the edge: Superslim machines are fomenting a quiet revolution in the server room. *IEEE Spectrum*, pages 25–29, April 2005.

[42] R. Wright. The man who invented the Web: Tim Berners-Lee started a revolution, but it didn't go exactly as planned. In *Time*, pages 64–69. May 1997.

[43] Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for embedded clusters. *SIGPLAN Notices*, 40(7):1–10, 2005.