

ENERGY CONSERVATION FOR SERVER SYSTEMS

by

EDUARDO SOUZA DE ALBUQUERQUE PINHEIRO

A dissertation submitted to the  
Graduate School—New Brunswick  
Rutgers, The State University of New Jersey  
in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

written under the direction of

Ricardo Bianchini

and approved by

---

---

---

---

New Brunswick, New Jersey

May, 2005

## **ABSTRACT OF THE DISSERTATION**

### **Energy Conservation for Server Systems**

by **EDUARDO SOUZA DE ALBUQUERQUE PINHEIRO**

**Dissertation Director:**

**Ricardo Bianchini**

To date, energy consumption has only been a concern for mobile and embedded devices due to battery lifetime and heat dissipation concerns. In this thesis we make the case for energy conservation for server systems. We describe in details the current mechanisms for power management and the state-of-the-art policies for energy conservation for the main components of servers. We propose three novel energy conservation techniques for servers, namely Load Concentration, for entire servers; Popular Data Concentration, for disk arrays; and Diverted Accesses for distributed file servers. Our experiments, physical measurements, simulations, and models show that our techniques can greatly enhance the opportunities for power management and consequently accrue significant energy savings.

## Acknowledgements

The acknowledgement part of a thesis is probably the most boring part of it. However, it is also the only part that most people I care about will ever read. Therefore, I will attempt to please these people by writing something to them here.

Writing a comprehensive acknowledgment page is also challenging. With so many friends, colleagues and family members to thank, I do not know how to start, who to include, who to leave out. If I include too many people, I dilute my thanks and this becomes a list of names. If too few, people will be felt forgotten. If I include only those who contributed directly or indirectly to the making of this thesis I might be faced with the need to thank the mailman for delivering my books or Randy, the chef, for cooking some of my meals at Rutgers.

Therefore, I am going to thank only those who made the most important contributions to my thesis. So, my thanks go:

To my advisor, Ricardo Bianchini for all the help and friendship, for believing in me never giving up on me, even after I told him I wanted to quit and be a screenwriter. I will never forget all the endless nights in the lab, with or without the popcorn.

To my family: my mom Suelly, my dad Rogério and my brother Pedro, for being there for me, always, unconditionally. To my grandfather Geraldo, also for being there for me. To my cousin Mariana for also being there for me and helping

out with long chats when I needed.

To the people in the Panic Lab, who helped with comments, suggestions and hugs. In special, thanks to Matias Cuenca, Chris Peery, Xiaoyan Li and Kiran Nagaraja.

To professors at Rutgers who were directly or indirectly involved with my research: Uli Kremer, Rich Martin and Thu Nguyen.

To the people and friends in the Dark Lab who were the ones closest to me and helped me the most. This thesis is a reflection of your comments, ideas and your very own sweat and blood too. Special thanks to my friends Fabio Oliveira, Taliver Heath, Vinicio Erazo, Gustavo Gama, Diego Nogueira, Ana Paula Centeno and Bruno Diniz.

To my friends Tiberius and Mara Bonates for being there and saying “Móóóóóó” at the right times.

Finally, special thanks to my loved wife, Izabel, to whom I dedicate this thesis.

To my wife, Izabel.

# Table of Contents

<b>Abstract</b> . . . . .	ii
<b>List of Figures</b> . . . . .	x
<b>1. Introduction</b> . . . . .	1
1.1. Overview . . . . .	4
1.2. Contributions . . . . .	5
<b>2. Background</b> . . . . .	8
2.1. Mechanisms For Power Management . . . . .	8
2.1.1. CPU . . . . .	9
2.1.2. Disks . . . . .	12
2.1.3. Memory . . . . .	14
2.2. Network . . . . .	15
2.3. Characteristics of Servers . . . . .	16
2.4. Power Analysis: Where is all the power going? . . . . .	18
<b>3. Related Work</b> . . . . .	21
3.1. Entire Server . . . . .	22
3.2. Server Subsystems . . . . .	23
3.2.1. Processing System . . . . .	24
3.2.2. Main Memory System . . . . .	25

3.2.3.	Network Infrastructure . . . . .	26
3.2.4.	Storage System . . . . .	27
	Without Redundancy . . . . .	27
	With Redundancy . . . . .	29
3.3.	Other Related Energy Conservation Techniques . . . . .	30
<b>4.</b>	<b>Load Concentration . . . . .</b>	<b>32</b>
4.1.	Cluster Configuration and Load Distribution . . . . .	33
4.2.	Implementations . . . . .	39
4.3.	Methodology . . . . .	43
4.4.	Experimental Results . . . . .	44
4.5.	Conclusions . . . . .	50
<b>5.</b>	<b>Popular Data Concentration . . . . .</b>	<b>51</b>
5.1.	Popular Data Concentration . . . . .	53
5.2.	Nomad FS . . . . .	54
5.3.	Comparison Against Other Approaches . . . . .	58
5.4.	Methodology . . . . .	60
5.4.1.	Simulation . . . . .	61
5.4.2.	Simulator Validation . . . . .	64
5.4.3.	Parameter Space . . . . .	66
5.4.4.	Synthetic Trace Generator . . . . .	68
5.4.5.	Real Traces . . . . .	69
5.5.	Parameter Space Results . . . . .	70
5.5.1.	Arrays of Conventional Disks . . . . .	72

5.5.2.	Arrays of Two-Speed Disks . . . . .	73
	Workload Characteristics . . . . .	73
	File Server Characteristics . . . . .	78
5.6.	Real Trace Results . . . . .	80
5.7.	Conclusions . . . . .	82
<b>6.</b>	<b>Diverted Accesses . . . . .</b>	<b>85</b>
6.1.	Background . . . . .	87
6.1.1.	Redundancy . . . . .	87
6.1.2.	Disk Energy Conservation . . . . .	88
6.1.3.	Storage System Design . . . . .	89
6.2.	Diverted Accesses . . . . .	90
6.3.	Designing Real Systems . . . . .	92
6.3.1.	Reliability and Availability . . . . .	93
6.3.2.	Performance: Throughput . . . . .	94
6.3.3.	Putting It All Together . . . . .	95
6.4.	Modeling Energy . . . . .	95
6.4.1.	Overview . . . . .	95
6.4.2.	Energy Conservation Techniques . . . . .	97
6.5.	Modeling Results . . . . .	106
6.5.1.	Availability and Throughput . . . . .	108
6.5.2.	Energy . . . . .	110
6.5.3.	Defining a Redundancy Configuration . . . . .	116
6.5.4.	Summary . . . . .	118
6.6.	A Case Study: PAST . . . . .	119

6.6.1. Model Validation . . . . .	121
6.6.2. Real Workload Results . . . . .	122
6.7. Conclusions . . . . .	124
<b>7. Conclusions and Future Work . . . . .</b>	<b>126</b>
<b>References . . . . .</b>	<b>129</b>

## List of Figures

2.1. Typical popularity of files in Web servers. . . . .	18
2.2. Typical server workload pattern. . . . .	18
2.3. Power breakdown for our Pentium III server. . . . .	19
4.1. Cluster evolution, WWW server, $\text{degrad} = -30\%$ . . . . .	46
4.2. Power consumption, WWW server (static vs. dynamic) . . . . .	46
4.3. Cluster evolution, WWW server, $\text{degrad} = -15\%$ . . . . .	47
4.4. Cluster evolution, OS, $\text{degrad} = 0\%$ . . . . .	47
4.5. Power consumption, OS (static vs. dynamic) . . . . .	48
4.6. Cluster evolution, OS, $\text{degrad} = 20\%$ . . . . .	48
5.1. Energy savings per file request rate (conventional disks). . . . .	72
5.2. Energy savings per file request rate (two-speed disks). . . . .	73
5.3. Energy savings per file system coverage. . . . .	75
5.4. Energy savings per file popularity. . . . .	76
5.5. Energy savings per percentage of writes. . . . .	76
5.6. Energy savings per temporal correlation. . . . .	77
5.7. Energy savings per number of disks. . . . .	78
5.8. Energy savings per main memory cache size. . . . .	80
5.9. Energy savings per reconfiguration period. . . . .	80
6.1. Second scenario: $I \geq T + T_u$ . . . . .	99

6.2. Third scenario: $T \leq I < T + T_u$ . . . . .	99
6.3. Availability and throughput for $n = 16$ . . . . .	107
6.4. Availability and throughput for $m = 1$ . . . . .	107
6.5. Energy savings for $n = 16$ . . . . .	110
6.6. Energy savings for $m = 1$ . . . . .	110
6.7. Savings per request rate for $n = 8, m = 1$ . . . . .	112
6.8. Savings per write percent for $n = 8, m = 1$ . . . . .	112
6.9. PDC and PDC+DIV savings as a function of coverage. . . . .	112
6.10. MAID and MAID+DIV savings as a function of cache miss rate. . . . .	112

# Chapter 1

## Introduction

Power and energy consumption have always been critical concerns for laptop and hand-held devices, as these devices generally run on batteries and are not connected to the electrical power grid. Over the years, a large amount of research has been devoted to low-power and low-energy design and conservation (e.g. [32, 40, 48, 71, 104]).

In this thesis we tackle the energy consumption problem in server systems. Servers are network providers of services. They are typically housed in machines connected to the electrical power grid and thus the relevance of power and energy consumption for them is different. Servers typically are the backbone of many Internet services, data centers, and research facilities. For scalability, servers are usually used in clusters, in quantities ranging from hundreds to thousands. In these quantities, servers can collectively draw power comparable to that consumed by six oil refineries [27], the equivalent of the consumption of a small town (about 445 megawatts). This amount of power consumption can easily escalate when multiple data centers are geographically close, which is often the case (e.g. Silicon Valley, Research Triangle Park). Back in 2001, some planned data warehouse expansions in California were cancelled due to the limitation of power generation in the area [26].

Power generation and distribution can be even more limiting when we take

into account the extra power needed for the cooling equipments. Estimates from Sun Microsystems show that a typical 1.2 Ghz server with 2GB of memory and other standard server components consume 750W of power and dissipates 2560 BTU/h of heat [77]. An efficient air conditioner to cool ten of these machines would consume 862.5 W [37], which is more than 10% of the power budget to operate the ten machines and is consistent with the lower range of consumption in studies of water-cooled data center [78].

Perhaps more importantly, the energy cost of operating these clusters of servers can be a significant fraction of the total revenue the servers provide. Consider the example of a web hosting center. In this type of business, revenue comes from renting out disk space, networking, processing power, cooling and power supply to the end clients. A few different plans are usually available. They range from small startup packages that include a single shared host with limited network capacity and disk space (typically 50 MB/month and 5 MB total, respectively) to dedicated servers with capacity limited by hardware to co-location options, which include multiple servers.

The small startup packages vary widely in options and prices and also the number of other clients sharing a server is unknown. But for dedicated servers and co-located servers, the prices are more consistent. Looking at an example of co-located servers, we can estimate how much of the revenue is spent on energy for the servers alone.

As of this writing (March 2005), the typical price for a dedicated mid-size to large-size server is between \$325 and 350 [80, 99]. Included in the price is networking (50GB/month), power and 2 hours of technical support (1 hour emergency) for 4U to 7U servers (a “U” is a unit of height a server takes up on a rack, 1 U

being 1.75 inches).

A typical 3 U blade server with 18 blade computers (18 Pentium III CPU, a 4200 rpm laptop disk and 512 MB DRAM per blade) consumes altogether 1200 W. The monthly cost of operating this server, just accounting for the server itself, without networking, cooling or backup power generation, can be estimated (considering the current price per kWh of energy to be \$0.10) at  $1200 * 30 * 24 * 0.10/1000 = \$86.40$ . That is 24-26% of the price charged by the hosting center to operate this server. This range is consistent with ranges found by others [16, 79].

If we include air conditioning costs this percentage will go up even further. For example, assume an efficient 18000 BTU air conditioner, rated at 18 SEER (Seasonal Energy Efficiency Rating)<sup>1</sup>. Such an air conditioner can cool 18000 BTU per hour with 1 kWh of energy. Our 1200 W server produces  $1200 * 24 * 365/1000 = 10512$  kWh of energy in form of temperature. The cooling will then be required to consume approximately  $10512 * 3414/18000 = 1994$  kWh which is approximately 19% of the total energy budget to operate the server. This figure does not account for energy needed to circulate the cool air (fans).

Between cooling and operating costs, a web hosting center might be spending \$174.00 of their \$325-350.00 revenue on energy. That represents a staggering 50-54%.

Besides monetary costs, energy conservation is important due to the way power is generated. In many countries, the prevalent method for power generation is coal-fired or nuclear fission. Both of those are harmful to the environment either directly (smoke pollution) or indirectly (nuclear waste). According to a

---

<sup>1</sup>National appliance standards require room air conditioners built after January 1, 1990, to have an SEER of 8.0 or greater [95] and the most efficient ones are rated at about 18 SEER [107].

scientific article published by researchers at School of Public Health (SPH), two coal-fired power plants were linked to “more than 43,000 asthma attacks and nearly 300,000 incidents of upper respiratory symptoms per year in the region”. Also, the study “estimated that 159 premature deaths per year could be attributed to this pollution” [5]. The web is riddled with other similar reports on power plant pollution.

To alleviate these problems, some previously-proposed techniques for embedded and mobile devices could be used for servers, albeit with limited success. These previously-proposed techniques leverage long idle times to accrue energy savings. In servers, the resource usage patterns are markedly different than those of embedded and mobile devices. Typically, idle times are short or non-existent in servers, making the usefulness of previously-proposed techniques that exploit idle times very limited. It is therefore imperative that idle times be created or extended in servers in order for these techniques to be successful. Our work aims at doing exactly this.

## 1.1 Overview

In this thesis, we propose policies for resource management in servers, targeted at conserving energy without degrading performance beyond a pre-established acceptable level. Managing resources efficiently according to the imposed load on them is necessary for an efficient use of energy and thus necessary for energy conservation.

We first describe the current mechanisms available for energy conservation, regardless of whether they are targeted at servers or mobile devices. Then we

describe the related work, including some non-energy works that are related to our contributions (outlined in the next section). Then we propose three main techniques for resource management and their contributions. We finish with a conclusion and the future directions of our work.

## 1.2 Contributions

This thesis aims at reducing the energy consumption in server systems, without significant degrading performance. To that end, we focus on scheduling the offered load to increase idle time. By reducing the need for power-consuming resources when load is low, these resources can be turned off or switched to low-power modes and thus save energy.

To substantiate our contribution, we propose three techniques, namely 1) Load Concentration; 2) Popular Data Concentration; and 3) Diverted Accesses.

**Load Concentration.** Load Concentration targets entire servers in a cluster for maximum savings. It detects when global utilization of the cluster is below maximum capacity and reduces the number of servers online (powered up) by shifting load to a smaller subset of servers and powering the others off. Load, in this case, is resource-consuming applications that might consume all or a combination of disk bandwidth, CPU time, and memory. We find that applying Load Concentration to an eight-node, cycle-server cluster under a mixed workload can save between 32 and 40% of energy compared to an always-on, energy-oblivious cluster.

**Popular Data Concentration.** This technique targets an important component of server systems, the storage subsystem. More specifically, it targets disk

array-based file servers. The idea is to increase idle times on some disks at the expense of increasing utilization on others, similar in spirit to Load Concentration. Files are periodically ranked in popularity (based on frequency and recency of access) and the more popular ones are migrated to a subset of disks while the unpopular ones are migrated to another subset. These migrations increase or produce large idle times on the unpopular disks, allowing the system to conserve energy by transitioning them to low-power mode. Our experimental results show that Popular Data Concentration has limited use under low intensity workloads. However, simulation results using two-speed disks show that Popular Data Concentration can save between 27 and 43% energy with no more than 8% of the requests being delayed. Two-speed disks were used due to their reduced energy overhead in transitioning between the two speeds. The gains with PDC were achieved compared with two-speed disks without PDC.

**Diverted Accesses.** The previous technique, as well as all other independently-proposed techniques for storage systems, did not consider the wide use of redundancy in modern storage systems. Diverted Accesses fills this gap by taking advantage of redundant information to improve performance, improve data availability, and conserve energy. It does so by segregating redundant information from original information onto separate disks (or separate storage nodes altogether) and diverting accesses to the original set, while the redundant set remains idle and thus in low-power mode. Our mathematical models and detailed simulations show that Diverted Accesses can save from 20-60% of energy of a peer-to-peer storage system under realistic traces.

In summary, the contributions of this thesis are:

- A new focus onto energy conservation for servers;

- Three energy management techniques for dynamically regulating resources to imposed load in clustered server systems;
- Thorough evaluation of techniques over a wide range of parameters using mathematical models, implementations, and detailed simulations.

The rest of this thesis is divided as follows. In chapter 2 we discuss the important background necessary to contextualize our work. In chapter 3 we thoroughly analyze previously-proposed techniques and methods for energy conservation in server systems, as well as in individual components. In chapter 4 we detail our contribution on entire servers via the Load Concentration technique. In section 5 we study disk-array-based servers and the Popular Data Concentration technique. In chapter 6 we broadly study energy conservation techniques for redundant storage servers and propose the Diverted Access technique. Finally, in chapter 7 we draw conclusions and provide directions for future work.

## Chapter 2

# Background

In this chapter we present the required background to understand energy and power conservation in computer systems. In particular, we look at mechanisms for power management for CPU, disks, memory, and networking hardware. These mechanisms are the foundation for power reduction and therefore for energy conservation as well.

### 2.1 Mechanisms For Power Management

There are a few ways of managing power in computer systems, but almost all of them involve either powering off resources when they are idle to avoid wasting power in idle cycles or slowing down resources to adjust to imposed load and deliver just enough performance to accomplish the required work without exceeding performance requirements.

Other ways to conserve power are by not doing the work – by sending computation somewhere else and thus spending a remote power budget – or by doing less work with potentially less quality, such as in transcoding or eliminating work (e.g. skipping frames when playing a movie).

In other words, techniques either exploit idle times or hide power consumption with lower power but higher latency operations.

Power management, regardless of the nature of the resource (memory, CPU, etc), has to be applied in such a way that it does not hurt performance significantly, since energy (our final goal) is directly proportional to time. Moreover, any power management policy must take into account the overheads incurred when transitioning to and from low-power modes. There are two forms of transition overheads: time and energy. These overheads vary depending on the nature of resources, models, manufacturers, etc. Disks, for example, can take seconds and several dozen joules to transition between power modes, whereas memories take a few nanoseconds and only millijoules to transition between states. But for all resources, when transitions are done too often, performance is hurt and energy overhead becomes a significant part of the total energy consumption.

Besides transition costs, some resources have low-power modes that also deliver lower performance. If the lower performance does not match the imposed workload's performance requirements, total execution time stretches and thus can hurt energy conservation. Balancing all these factors is a challenge.

Next, we examine a few examples of techniques in the context of four important resources, CPU, disk drives, networking, and memory.

### **2.1.1 CPU**

CPUs consume a significant portion of the power budget of a computer system (laptop, desktop, or server). Typical power consumption varies with technology, clock speed, and whether or not it is intended for mobile or embedded devices, desktops, servers, or SMP servers. Typical values are listed on table 2.1.

From the table, we see that modern processors consume as little as 5 W or as much as 120 W. To minimize power consumption it does not suffice to pick the

Name/Model	Clock (Ghz)	System	Power (W)	
			Typical	Maximum
AMD MP-2800+	2.13	Desktop	47.2	60
AMD Athlon XP-3200+	2.2	Desktop	60.4	76.8
AMD Duron-Applebred	1.8	Desktop	53	57
AMD Athlon XP-2200+	1.8	Mobile		35
AMD Opteron-252	2.6	Server		92.6
AMD Athlon 64-4000+	2.4	Server		89
AMD mAthlon 64-3400+	2.2	Mobile		81.5
AMD mSempron 2800+ LA	1.6	Mobile		25
Intel PIII-S-1.4	1.4	Desktop		31.2
Intel mPIII Speedstep	0.85	Mobile		22.5
Intel mPIII-M Speedstep	1.33	Mobile		22
Intel P4-660	3.06	Desktop		115
Intel mP4-552	3.46	Mobile		88
Intel mCel	2.5	Mobile		35
Intel Xeon	3.6	Server	110	120
Transmeta TM5800	0.733	Mobile	0.419-1.049	5.5
Transmeta TM5800	0.8	Mobile	0.419-1.049	6

Table 2.1: CPU power consumption. “System” is the intended target system for the CPU. Power consumption (when available) is shown as stated by the manufacturer. Source: Chris Hare, <http://users.erols.com/chare/main.htm>.

CPU with lower power-to-clock ratio since not all CPUs shown have the same IPC (instructions per cycle). Also, it does not suffice either to pick the CPU with highest IPC-to-power ratio if that CPU is not a low-power one, because when idle, the CPU would still consume a lot of power for no useful purpose (the IPC when idle is zero). Powering down the CPU when it is idle is feasible in batching systems but requires interrupts to wake up the CPU. The other – more popular – way to manage power in CPUs is to do work more efficiently, i.e. consume less power when the demand permits it. That is the idea behind Dynamic Voltage Scaling [104]. Since power consumption is proportional to the square of the voltage ( $P \propto CV^2f$  – where  $C$  is a capacitance constant,  $f$  is clock frequency,  $P$  is power and  $V$  is voltage), a linear reduction in voltage reduces power quadratically and only slows down a CPU linearly (because  $f \propto V$ ).

So, power can be reduced if there is no need for a CPU to compute as fast as possible. For example, an MP3 player application only needs to decode audio streams every tenth of a second, approximately. That means a 32-bit CPU operating at 100 MHz could handle the job. By lowering the voltage and matching the clock frequency to meet the stream's requirements, power consumption is reduced as a side effect.

An example of a DVS-capable CPU is the AMD Athlon Mobile 4, used in [57]. Its voltage ranges in increments of 0.05 volts from 1.15 to 1.45 volts. Each increment in voltage corresponds to an increment of 100 Mhz, starting from 600 Mhz at 1.15 volts to 1200 Mhz at 1.45 volts. The time to transition is estimated to be under 100  $\mu$ s. Another example of DVS-capable CPU is the Transmeta TM5900-1000 processor [28]. Its clock frequency varies from 433 Mhz at 0.8 volts to 1000 Mhz at 1.25 volts. The time it takes to come back from its deepest sleep mode to active mode is rated at less than 25  $\mu$ s.

Another way of doing work more efficiently is perhaps by not doing any of the work itself, but have it done remotely, at the expense of handing it out and supervising its execution. In computing terms, this idea translates into remote execution and can be applied to mobile devices with limited resources. Similarly, transcoding is another way of limiting resource consumption to generate a lower quality of service at a reduced energy. For example, a successful transcoding technique would be if serving a black and white video (as opposed to full color) were to reduce the number of bytes transmitted by the network card.

### 2.1.2 Disks

To understand disk drive power consumption, it is helpful to understand how they are built. Disks have one or more platters whose surface is used to store data. On each surface, a disk head moves longitudinally. When powered up, the platters spin at a constant velocity, typically between 4 to 15 thousand RPMs.

Except on laptop drives, all drives' platters and head are powered by a 12-volt line while logic (for caches, I/O, scheduling) is powered by a 5-volt line. A sizeable portion of the power is consumed by the motor used to rotate the platters and the arm used to move the head to the desired position (seek). Our measurements, for example, show that on a 7200 rpm IDE drive, the motor and head movement consume 65% of the power when the disk is idle. On a 10,000 rpm SCSI disk, this fraction is 45% percent (see Table 2.2) <sup>1</sup>. So ideally, disks can take advantage of idle times to spin down the spinning platters and thus save a sizeable portion of the power consumption. Furthermore, most of the logic does not need to be powered on either, so most disks can accrue significant power savings when in spun down mode.

However, time and energy to transition to spun down mode and back to idle mode varies from about a second for the IDE drive to a dozen of seconds for the SCSI drive and in both cases more than 40 joules of energy are spent. These transition costs can be a significant fraction of total energy expenditure depending on the workload requirements and idle times available. Transition costs alone can ruin any energy savings accrued by the low-power mode if done excessively.

---

<sup>1</sup>Even though the total power consumption of the SCSI disk drive is higher, the rotating platters consume a smaller fraction of total power than in the IDE disk. We believe this happens because there is more cache memory and more logic needed in the SCSI disk and therefore the 5 V line consumes a higher fraction of the power than in the IDE disk.

Measurement	IDE	SCSI
12 V spinup	37.68 J	47.31 J
12 V spindown	3.78 J	2.97 J
12 V idle	2.90 W	2.33 W
12 V read	3.95 W	4.38 W
12 V write	4.15 W	N/A
12 V spundown	0.16 W	0.40 W
5V spinup	1.42 J	18.59 J
5V spindown	0.60 J	25.28 J
5V idle	1.50 W	2.93 W
5V read	3.40 W	3.77 W
5V write	2.40 W	N/A
5V spundown	0.80 W	1.46 W
spindown time	0.6324 s	11.24 s
spinup time	0.6286 s	6.12 s

Table 2.2: Measured power profile of typical IDE and SCSI disks. The IDE disk is an IBM Deskstar 7200 rpm 40Gb drive. SCSI is a 10,000 rpm Seagate Cheetah 9.1Gb drive. Writes on the SCSI drive were not measured directly but a mix of random reads and writes consumes 7.09 W on average.

Some disk models have multiple power states, especially laptop disks [60]. These extra power states accrue increasingly higher power savings by turning off parts of the disks logic. However, the platters stop rotating only in the last power state, thus limiting the savings of the different modes to small fractions of the total power consumption.

On the other hand, disks – like CPUs – can also be made to work more “efficiently” when load is low. By rotating the platters more slowly disks can consume less power, but also transfer data at slower rates. This type of disks are called multi-speed disks [21, 45]. They can be used to take advantage of their multiple rotating speeds with different latencies, transfer rates, and power consumption.

State	RAMBUS		MobileRAM	
	Power (mW)		Power (mW)	
Active	313		275	
Standby	225		75	
Nap	11		N/A	
Powerdown	7		1.75	
Transition	Time (ns)	Energy (nJ)	Time (ns)	Energy (nJ)
	Standby $\rightarrow$ Active	3	0.94	0
Nap $\rightarrow$ Active	228	71.4	N/A	N/A
Powerdown $\rightarrow$ Active	22513	7046.5	7.5	2.1

Table 2.3: Memory power consumptions, transition times, and transition energies. Transition energies are estimated by multiplying time to transition by power at active state. Sources: Fan *et al.* [38] and Huang *et al.* [59].

### 2.1.3 Memory

Memories typically have multiple power states, each consuming increasingly less power at the expense of taking more time and energy to switch back to active mode (in which data can be read) [59]. These power modes dictate how much logic is on at a time. Components like row and column decoders and clock signals can be turned off. The only necessary component is the refresh signal, without which the memory would lose its data.

For a typical RAMBUS memory [29], transition times vary from 22510 ns from powerdown to standby to 3 ns from standby to active. Power consumption per each memory bank varies from 7 mW at powerdown mode to 313 mW at active mode. Other types of memory have similar characteristics (see Table 2.3).

Memory has similar tradeoffs as disks or any other device. Care must be taken when implementing energy savings techniques to avoid having lower power modes delay accesses too much and therefore increase energy consumption. Also, frequent transitions can reduce savings by the overhead of transition.

## 2.2 Network

Network adaptors share similar power characteristics with memory and CPU, since both components are integral parts of typical adaptors. Therefore, most of the mechanisms for power management in network adaptors can borrow from the mechanisms used in memory and CPUs.

However, memory in network cards is limited to small I/O buffers and processing units are fairly simple and low-power. Not surprisingly, network adaptors typically consume very little power, compared to the rest of the server system and not much work is in the literature about energy conservation in network adaptors.

On the other hand, routers and switches are power-hungry machines. For them, packet switching efficiency is key to performance. The power consumed by routers and switches has historically been considered just a side effect of high performance and thus many opportunities exist for power management. For example, an Infiniband 8-port 12x (30 Gb/s) switch consumes 16.4 W just for the link circuitry [84].

The main mechanism for conserving router and switch energy is to power off unused or idle links and route around them. When these links are needed for high performance, they are reactivated according to the policy in use.

Better design is another way of reducing power consumption in routers and switches. For example, reducing resistance of the wires while shielding them from interference lowers the power envelope of the interconnect as a whole. Also, a careful choice of components and the layout of the various router buffers, cross-bars, arbiters, and links is central to the design of power-efficient networking hardware.

## 2.3 Characteristics of Servers

Energy conservation in server systems differs markedly from that in laptop, handheld, and embedded devices. In servers, there is typically no idle time readily available to be exploited. The typical workload on servers is intense enough that resources are either mostly utilized or their utilization varies so rapidly that powering off resources during idle times can become a serious problem depending on the energy overhead involved in the transition [21, 87].

Moreover, workloads on server systems are not interactive and limited by a single user's capacity to generate load. For example, on laptop systems, a single user might listen to music and surf the web, watch a DVD or edit code and compile, but not everything at the same time. This means that resources of the laptop are used at different times and with different intensities. On servers, however, it is possible for all subsystems to be stressed most of the time.

Given the different type of workload and lack of idle time on servers, energy conservation becomes a challenge. Idle time must be created or extended so that traditional power-management mechanisms can be used.

Replication of resources and redundancy in servers can be exploited to create or extend idle times. Because server systems are designed to support worst-case peaks in demand, the servers' resources are under-utilized for a large fraction of their lifetime. For example, internet services typically exhibit load peaks during the day and valleys at night time while for some commercial sites weekends show a similar pattern, but with lower intensity due to reduced commercial traffic. Figure 2.2 shows the load profile of the seven-day trace collected at AT&T. The daily patterns are clearly seen, though in this example the weekend effect is not

observed (the first peak is a Saturday) because it is a proxy mainly for home-based, dial-up users. Servers can take advantage of the predictability of this type of workload and reconfigure resources to meet demand. This under-utilization does not mean there is enough idle time left for energy conservation policies to employ the mechanisms discussed earlier, because load is typically spread evenly across all resources of a given type whenever possible, to avoid bottlenecks and single points of failure. However, done carefully, load manipulation can be used to both keep performance at appropriate levels and still increase idle times in some resources.

Another way that server workloads differ markedly from that of single-user workloads is in their locality and popularity of reference. In single-user systems, a few documents (local files, Web pages, database tables, etc) can be referenced a few times and the distinction between these documents in terms of locality of reference and popularity can be moot, since the number of accesses are few and limited by the intensity of what is generated by a single user. In servers, however, the effect of multiple users accumulate and cause documents to observe the collective effect of popularity and locality of reference. This means some documents will be more popular than others and some documents will be more recently accessed than others (these two sets need not be the same). Servers can take advantage of these characteristics of workloads by utilizing caches and performing data reorganization/migration to improve response time and also reduce power. For example, Web, file, and domain name servers (DNS) workloads have been found to follow a Zipf power-law in the distribution of accesses to their documents [12, 18, 66]. This means that a few documents are accessed frequently while others are very unpopular. Figure 2.1 shows the distribution of accesses

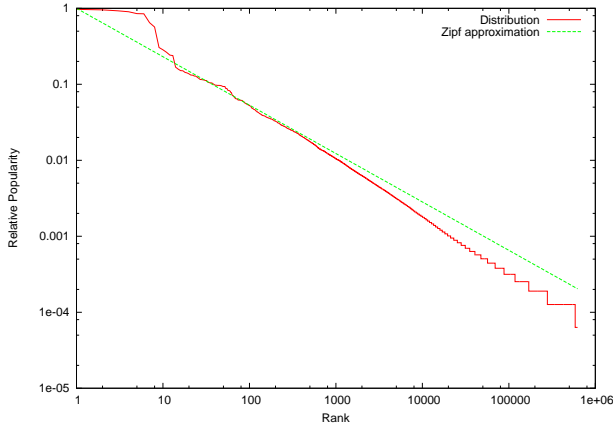


Figure 2.1: Typical popularity of files in a Web server versus the synthetic Zipf distribution with  $\alpha=0.63$ . Web proxy trace from AT&T.

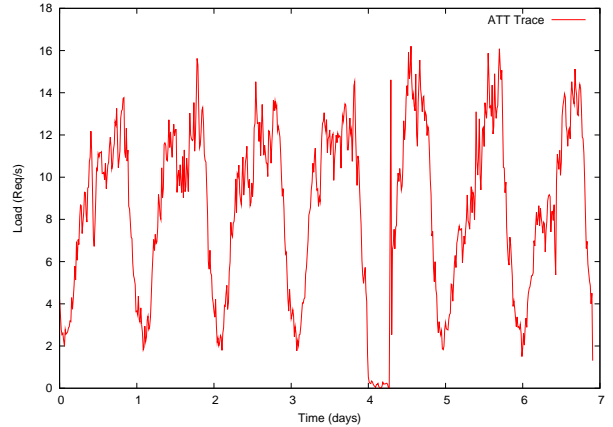


Figure 2.2: Typical server workload pattern. Web proxy trace from AT&T.

to files of a trace collected from Web proxies at AT&T from 01/16/99 through 01/22/99.

## 2.4 Power Analysis: Where is all the power going?

Since the focus of this thesis is on saving energy by reducing power consumption (rather than improving performance and thus reducing execution time, at constant power), we need to know where the power is consumed in typical servers.

A typical server node consists of one or more CPUs, several memory boards, a motherboard with various I/O components and caches, peripherals such as network cards, and one or more disks drives. To power everything, a regulated power supply must convert AC current to DC and provide enough excess power for changes in power demand by all these components.

Our particular server node is composed of an 800-Mhz Pentium III CPU, 512 MB memory in two PC133 ECC DIMMs, two 9-GB IBM DNES-309170W SCSI drives with a SCSI card, a PCI video card (but no monitor), one cLan Gigabit Ethernet card, and two Intel Ethernet Pro 100 Mb network cards. This machine

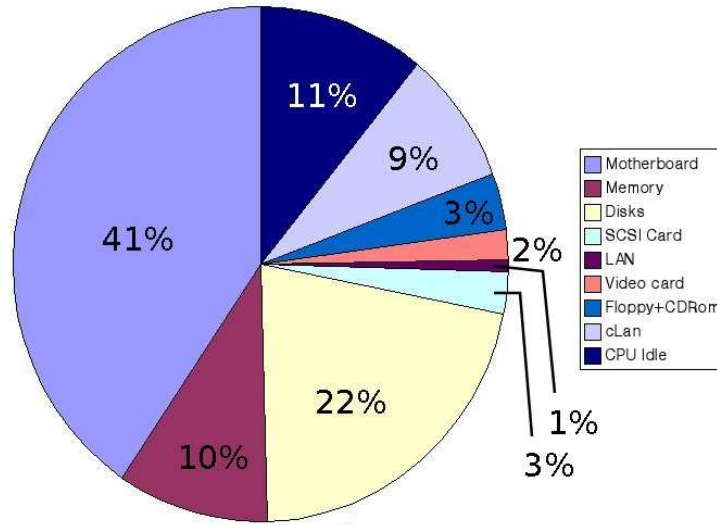


Figure 2.3: Power breakdown for our Pentium III server. Key goes counter-clockwise starting at the twelve o'clock position (41% is 'Motherboard').

consumes 4.3 Watts when it is powered off but plugged into the power grid. When it is on and fully functional but idle, it consumes 70 Watts. At maximum utilization – when all disks, memory and CPU are stressed – it consumes 94 Watts. Figure 2.3 shows the power breakdown per major component.

From the figure, we can see that the motherboard (plus the power supply's overhead) accounts for roughly 41% of total power consumption, followed by disks at 22%, and CPU and memory with 11 and 10% each, respectively.

Most interestingly, however, the difference in power consumption between idle and fully utilized is not great, only about 24 W. This means that the difference in power consumption between having two machines running below their maximum capacity and one machine running at peak capacity while the other machine is off is advantageous (i.e.  $94 + 0 < 70 + 70$ ).

A similar conclusion can be drawn from the disk profile. As an example, consider the Seagate SCSI disk we measured (see table 2.2). This disk consumes, on average, 5.26 W when idle and 7.09 W when fully utilized – doing seeks, reads

and writes (not shown in the table). In sleep mode (when the platters cease to rotate, the heads park, and most logic is powered off), it consumes only 1.86 W. This means that a single very busy disk plus a spun down disk consume less power than two mostly idle disks (i.e.  $7.09 + 1.86 < 5.26 + 5.26$ ).

Our resource management techniques (discussed in the next chapters) in part focus on shifting load towards subsets of the resources in order to create idleness and therefore more opportunity for power conservation.

## Chapter 3

### Related Work

A large body of previous work on energy conservation has focused on laptop computers, embedded, and hand-held devices. Research on these devices has included optimizations for the processor [48, 57, 58, 104], for the memory [31, 56, 71, 102], for the disk [32, 55, 73], and for offloading computation from them to non-battery-operated computers [70, 91].

Some of this research can be used to optimize each node of a cluster independently as well as individual components. Server systems can also benefit from these contributions. However, our research is orthogonal to these contributions in the sense that we focus on conservation that considers either all of the cluster resources or all resources of a certain subsystem (e.g. the storage subsystem), rather than individual nodes or devices in a subsystem. Moreover, we focus on creating or extending idle time in order to create opportunities for these standard techniques to work on servers.

In the next few sections, we categorize the related work into three broad categories, based on whether they focus on **entire server nodes**, **server subsystems** (e.g. array of disk drives, multiple memory banks) or **others**, which apply to single devices/components and are not server-specific.

Bianchini and Rajamony [16] surveyed these and other works on power and energy conservation for server systems.

### 3.1 Entire Server

In chapter 4 we introduce Load Concentration, a resource management technique for server clusters that concentrates load on a subset of the server nodes and reconfigures the cluster by powering off unused nodes. Perhaps the most closely-related research to our Load Concentration is that by Chase *et al.* [23]. They tackled the general problem of resource allocation in hosting centers using market-based policies. In terms of energy conservation, they evaluated a resource allocation policy for a clustered WWW server that is similar to the cluster configuration algorithm we study in chapter 4 and is also related to the system design methodology we present in chapter 6.

Load Concentration is similar in spirit to offloading computation from a battery-operated device to remote, non-battery-operated computers (e.g. [70, 91]). However, Load Concentration involves different challenges and tradeoffs, mainly because the load on the cluster and the effect of applying the technique must be determined before any action can be taken.

Our cluster reconfiguration and resource management work is inspired by previous work in cluster-wide load balancing (e.g. [11, 14, 24, 33, 42, 75, 86]). Some systems do use some form of load concentration, but only as a remedial technique like in systems that harvest idle workstations (e.g. [11, 75]) or as a management technique for manually excluding a cluster node. We use load concentration as a first-class technique for conserving energy in clusters.

A few other projects deal with cluster reconfiguration (e.g. [4, 41, 44, 101]). Even though these projects do not consider power and energy issues, they lend themselves nicely to the powering down of unused systems.

Other resource management techniques that consider the whole server are cluster reserves [6] and resource containers [10]. The latter is designed to compartmentalize resource allocation within a single node and the former broadens this view to an entire cluster of servers. In both cases, the objective is to provide performance isolation between the different services being offered by a single server or a cluster of servers. Our proposed resource management algorithm does not isolate the performance of different services nor does it differentiate between types of services, since we focus on servers that are dedicated to a single purpose or application (e.g. cycle servers, file servers, Web servers).

In terms of the load concentration algorithm, the most closely related work to ours is that of Skadron *et al.* [96]. They proposed the use of control-theoretic techniques for dynamic thermal management of microprocessors. We apply similar techniques to cluster reconfiguration for power and performance.

For clusters of heterogeneous servers, the challenge is to keep powered on only the nodes that consume the least amount of power per required throughput. With changes in load, different sets of machines provide the optimal power-to-performance ratio. In heterogeneous systems, Heath *et al.* [50] showed that a cluster-wide energy-saving modeling and request distribution strategy with reconfiguration can produce energy savings of up to 43% with minimal impact on performance.

### 3.2 Server Subsystems

We categorize subsystems in terms of the service they provide. For example, computation and storage are separate subsystems, provided by the CPUs and

disks, respectively.

### 3.2.1 Processing System

Elnozahy *et al.* [34] evaluated different combinations of cluster reconfiguration and dynamic voltage scaling [104] for clusters. They showed that the benefits of our technique can be increased by coupling it with coordinated (cluster-wide) voltage scaling. The same group had already shown that isolated voltage scaling was useful for servers [17].

For Web servers, Elnozahy *et al.* [35] also showed that request batching can save significant amounts of energy. This technique delays requests and processes them in batches, conserving CPU energy between batches by transitioning the CPU to low-power state. Batching was also combined with DVS for even further savings. Though studied for CPUs, batching is a generic policy and could be applied to network interfaces, disks, and even entire servers at a time.

Hsu and Kremer [57] proposed compiler transformations to exploit dynamic voltage scaling in CPUs during main memory stalls. If, at the time of a memory access, there is no parallelism that the CPU can exploit to continue execution, it should wait for the memory access to complete at reduced voltage and power and thus save CPU energy. The authors evaluated their technique for high-performance laptops and mentioned that the work is suitable to servers as well. It is not clear how effective the technique would be given that it was not tested in servers and the benchmarks were single applications from the Specfp95 suite and not a mix of applications, more likely to be found in server environments. However, at least in theory, the technique could be applied to servers.

### 3.2.2 Main Memory System

Most of the work in the memory system used mobile or single-user workloads to evaluate the impact of their policies [31, 59, 71]. They could be applied directly on servers. The results, however, would not be obvious, and depend on the level of stress imposed on the memory subsystem of servers.

For example, the Power-Aware Virtual Memory [59] works by shifting active pages to a subset of the memory nodes (a memory node is the minimum unit that is power-manageable in a memory system), while mostly-unused memory nodes are transitioned to low-power modes. With this policy, if a memory system is heavily-used, the savings might be null or insignificant. This work employs a similar mechanism (data movement) as our work on Popular Data Concentration, studied in chapter 5. However, their evaluation used single-user workloads, typically found in mobile computers and devices, not servers.

The work by Delaluz *et al.* [31] is similar in spirit to adaptive/predictive techniques used for disk drives. In these works, the inter-access times are monitored and a prediction is made based on past history. If the predicted future idle time is greater than a threshold, the memory nodes are transitioned to low-power modes. As we will show in chapters 5 and 6, idle times in servers are short and not suited for adaptive/predictive techniques.

Lebeck *et al.* [71] sought to minimize the energy-delay product and save energy without significant performance degradation. Their early work (2000) studied static and dynamic power state policies, various types of applications, and hinted at the benefits of dynamic page migration and the usefulness of hardware-assisted fixed-threshold power-down policies. But it did not consider more than one application at a time (which is the case of cycle servers) nor multiple clients

issuing requests (which is case of single-application servers), so applicability to server environments is unknown.

A few other works also looked at energy and power management of memory systems for mobile or embedded devices at the architectural and compiler perspectives, with similar limitations [47, 69].

Architectural support for energy-efficient caches [9] and memory hierarchy [8] have been proposed by Balasubramonian *et al.* for general-purpose use (desktop, mobile, or server). Although not designed exclusively for servers, these works were tested with a combination of workloads drawn from the Spec95, Spec2000 and Olden benchmarks, which can arguably be characteristics of servers workloads (of cycle servers, at least). However, a thorough examination of a mix of applications was not performed and thus a definite conclusion about the applicability of these works for servers cannot be made.

### 3.2.3 Network Infrastructure

Routers and switches are a vital part of server systems without which servers would not have meaningful purpose. Limiting peak power consumption on routers has been the focus of much research by Peh *et al.* [93, 97]. In these works, each routing node limits its peak power consumption by shutting down unused links and dynamically shifting traffic to under-utilized nodes (according to each nodes' power budget). Each router estimates current power usage and reacts to predictions of future power usage.

In [92], Shang *et al.* apply DVS to interconnect switches and show power savings of up to 6 times as compared to a power-oblivious switch, with only a 15% performance degradation. With these power savings and small performance

degradation, this technique is also an effective energy conservation technique.

Another work on switches, by Kim *et al.* [68], shows that DVS suffers from leakage power and can benefit from the Dynamic Link Shutdown (DLS) technique, which turns off entire links for more robust energy savings – up to 40% with 5% latency penalty – even when load is moderate to high (40 to 70% utilization).

Our work is orthogonal and complementary to these contributions, since routers and switches are necessary components to support servers.

### 3.2.4 Storage System

For the storage subsystem, we further divide categories into two, those that take redundancy into account and those that do not.

#### Without Redundancy

Several techniques have been proposed for disk energy conservation in storage systems. In the next few paragraphs, we divide them into three groups: threshold-based, data-movement, and other techniques.

**Threshold-Based Techniques.** The simplest threshold-based technique is *Fixed Threshold (FT)*. In FT, a disk is transitioned to low-power mode after a fixed threshold time has elapsed since the last access. Inspired by competitive policies, the threshold is usually set to the break-even time, i.e. the time a disk would have to be in low-power mode to conserve the same energy consumed by transitioning the disk down and back up to active mode.

Some studies have also considered adapting thresholds, based on past access patterns [32, 54]. We study in chapter 6 the best possible adaptive-threshold technique called *Oracle*. This technique transitions between power modes based

on future knowledge of idle times. Though not implementable without application support [52], we use Oracle as an idealistic basis for comparison.

**Data-Movement Techniques.** In this category are those techniques that migrate or copy data across disks. The *Massive Array of Idle Disks (MAID)* technique by Colarelli and Grunwald [25] uses extra disks to cache recently accessed data. On each access miss in the cache disks, the accessed block is copied from the original disk to one of the cache disks. If all cache disks are full, one of them evicts its LRU block to make space for the incoming block. The goal is to concentrate the accesses on the cache disks, so that the non-cache disks can remain mostly idle and, thus, be transitioned to low-power mode.

In contrast to the copy-based approach of MAID, *Popular Data Concentration (PDC)* [85], which we will discuss in chapter 5, migrates data across disks according to a combined metric of frequency and recency of access (popularity). The goal is to lay data out in such a way that popular and unpopular data are stored on different disks. This layout leaves the disks that store unpopular data mostly idle, so that they can be transitioned to low-power mode.

**Other Techniques.** Carrera *et al.* [21] and Gurumurthi *et al.* [45] proposed disks with more than one speed and showed that they can provide significant energy savings for different server workloads. Carrera *et al.* also showed that a combination of laptop and SCSI disks can be even more beneficial in terms of energy, but only for over-provisioned servers.

Zhu *et al.* [109, 110] proposed storage cache replacement algorithms that selectively keep blocks of data in main memory, so that certain disks can stay in low-power mode for longer periods of time.

Papathanasiou and Scott [82] proposed the direct replacement of laptop disks

for server-class disks, much in similarity with an earlier paper by Carrera *et al.* [21]. In contrast to the earlier paper, Papathanasiou and Scott showed that it might be possible to reduce power consumption without throughput degradation by using laptop disks to replace server-class disks, at the expense of decreased reliability and increased cost of hardware.

Our work is orthogonal to these contributions in that we seek to conserve energy in the context of disk array-based servers. For these systems, exploiting data movement or placement between disks can provide significant additional gains.

### **With Redundancy**

In terms of disk array-based servers, Gurumurthi *et al.* [46] considered the effect of different RAID parameters, such as RAID level, stripe size, and number of disks, on the performance and energy consumption of stand-alone database servers running transaction processing workloads. They also observed that it is not possible to exploit idleness in this context. However, they did not consider any other energy conservation techniques nor they exploited the use of redundancy explicitly to improve energy savings.

To our knowledge, the only work that explicitly takes redundancy into account for energy saving is Diverted Accesses, discussed in detail in chapter 6. The main idea behind Diverted Accesses is to segregate original blocks of data from the redundant blocks in such a way that entire disks are dedicated to redundant blocks and therefore are not needed under light load or when write/update traffic is low.

### 3.3 Other Related Energy Conservation Techniques

New storage devices, such as MEMS-based devices, have new characteristics that can be exploited for energy conservation, as in the work of Lin *et al.* [74].

As previously mentioned, handing off tasks to be performed remotely can alleviate the immediate energy problem (battery shortage) of a single mobile device, at the potential expense of increased overall energy consumption (that of the server plus mobile devices). This technique was first proposed by Rudenko *et al.* [91] and Kremer *et al.* [70].

The operating system (OS) has been the target of power and energy research as well (e.g. [13, 40, 71, 100, 104]). Vahdat *et al.* [100] suggest aspects that the OS should take into account when running on batteries and what could be done to avoid using energy unnecessarily, like turning off unnecessary portions of the memory subsystem. In [13], Benini *et al.* suggest that the OS should monitor resource usage so that shutdowns can be determined by the system more accurately than by applications or hardware alone.

Flinn and Satyanarayanan [40] developed a user-level middleware to filter and transcode data that applications fetch. Transcoding changes data quality in order for applications to use the minimum amount of energy when processing it. Vahdat *et al.* [22] and De Lara *et al.* [30] also concerned themselves with transcoding. A few previous papers considered application/OS interactions intended to optimize for power and energy [40, 76].

Heath *et al.* [52] used manual and compiler-assisted application transformations to show that increased burstiness in mobile disks saves considerable amounts of energy. They also used mathematical models to accurately predict power-mode

transitions on disk drives. We extend some of their models in our modeling of Diverted Accesses in chapter 6.

Papathanasiou and Scott [83] developed an efficient prefetching and caching mechanism for increasing burstiness on disk drives and save energy.

Fan *et al.* looked at main memory [71] and later [38] at the synergy between memory and processor voltage scaling and showed that the careful combination of techniques for memory and CPU voltage scaling outperformed either one alone.

For mobile computers, the network interface [98] and the wireless network [39, 65] have been the focus of energy research. Yan *et al.* [105, 106] showed that with modifications in mobile clients' networking protocol, energy savings can be accrued. These works generally do not apply to servers due to the low bandwidth and limited range of mobile network and wireless network cards.

## Chapter 4

# Load Concentration

In this section we propose and examine a resource management technique that focuses on entire server nodes at a time, in a clustered environment: Load Concentration.

In Load Concentration, our approach to conserving power and energy is to develop systems that can leverage the widespread replication of resources in clusters. In particular, we develop systems that can dynamically turn cluster nodes on – to be able to handle the load imposed on the system efficiently – and off – to save power under lighter load.

This research is inspired by previous work in cluster-wide load balancing (e.g. [11, 14, 24, 42, 75, 86]). When performing load balancing, the goal is to evenly spread the work over the available cluster resources and performance can be promoted. The inverse of the load balancing operation concentrates work in fewer nodes, idling other nodes that can be turned off. This *load concentration* or *unbalancing operation* saves the power consumed by the powered-down nodes, but can degrade the performance of the remaining nodes and potentially increase their power consumption. Thus, load concentration involves an interesting *performance vs. power tradeoff*.

Our systems exploit load concentration to conserve power. Their key component is an algorithm that makes load balancing and concentration decisions by

considering both the total load imposed on the cluster and the power and performance of different cluster configurations. In more detail, the algorithm uses a control-theoretic approach to determine whether nodes should be added to or removed from the cluster, and decides how the existing load should be re-distributed in case of a configuration change. To be able to understand the implications of our algorithm, we implemented it in two popular types of cluster-based systems: a network server and an operating system for clustered cycle servers. The implementations were performed in two ways: (1) at the application level for the network server; and (2) at the OS level for the cycle server. In a previous technical report [88], we also considered implementations that rely on application/OS interaction.

Even though we target power conservation primarily, our experimental results show that our secondary goal of saving energy is achieved as well. Our results show that the modified network server can reduce the total power consumption by as much as 71% and the energy consumption by 45% in comparison to the original server running on a static cluster configuration. The modified OS can reduce power consumption by as much as 88% for a synthetic workload, while attempting to avoid any performance degradation, again in comparison to the original system on a static cluster. The energy savings it accrues in this case is 32%. When a 20% performance degradation is acceptable, our system conserves 88% of the power and 40% of the energy consumed by the static system.

## 4.1 Cluster Configuration and Load Distribution

**Power vs. performance.** We consider the tradeoff between power and two

types of performance, namely throughput and execution time performance. Throughput is the key issue for systems such as modern network servers, in which the goal is to service as many requests as possible; the latency of each request at the server is usually a small fraction of the overall latency of wide-area client-server communication. Execution time is key for systems such as cycle servers, as users may object to significant delays in the execution of their jobs.

The cluster configuration and load distribution algorithm we propose decides whether to add (turn on) or remove (turn off) nodes, according to the expected performance and power implications of the decision. Decisions are made dynamically for each cluster configuration and currently offered load.

For simplicity, the algorithm assumes that the cluster is comprised of homogeneous machines. Furthermore, the algorithm assumes that the removal of a node does not cripple the file system. This is a valid assumption, since: (1) in certain environments it is possible to replicate files at all nodes; and (2) when this is not the case, the file servers can transparently be run on machines that do not strictly belong to the cluster or that are not subject to the algorithm.

**Addition/removal decision.** To make node addition or removal decisions, the algorithm requires the ability to predict the performance and the power consumption of different cluster configurations. Exact power consumption predictions are not straightforward. The problem is that it is difficult to predict the power to be consumed by a node after it receives some arbitrary load. Conversely, it is difficult to predict the power to be consumed by a node after some of its load is moved elsewhere.

Nevertheless, exact power consumption predictions are not really necessary for the algorithm to achieve its main goal, namely to conserve power. The reason

for this is that each of our cluster nodes consumes approximately 70 Watts when idle and approximately 94 Watts when all resources, i.e. CPU, caches, memory, network interface, and disk, are stretched to the maximum. These measurements mean that: (a) there is a relatively small difference in power consumption between an idle node and a fully utilized node; and (b) the penalty for keeping a node powered on is high, even if it is idle. Thus, we find in practice that turning a node off always saves power, even if its load has to be moved to one or more other nodes. Thus, our algorithm always decreases the number of nodes, provided that the expected performance of applications is acceptable.

Performance predictions can also be difficult to make. We predict performance by keeping track of the *demand* for (not the utilization of) resources on all cluster nodes. With this information, our algorithm can estimate the performance degradation that can be imposed on a node when new load is sent to it. There is a caveat here, though. A degradation prediction is made based on past resource demand history of the load to be moved on its current node, so the prediction does not consider demand changes due to unexpected future behavior. In particular, the initial settle-down period during which the caches are warmed up with the new load is disregarded; we are more interested in steady-state performance.

A throughput prediction can easily be made based on the resource demand information. To see how this works, let us consider the throughput of a cluster-based network server. Suppose a scenario with 3 cluster nodes, each of which with demands for disk of 80%, 30%, and 20% of their nominal bandwidth. By adding up all of these disk demands (and disregarding other resources to simplify the example), we find that the server could run with no throughput degradation on 2 nodes ( $130 < 200$ ) and with a 30% throughput degradation on 1 node ( $130$

- 100 = 30). Our algorithm should decide to remove one at least; two nodes if a 30% degradation is acceptable.

Execution time predictions are much more complex, as they depend heavily on the specific characteristics of the applications and on the amount and timing of the demand imposed on the different resources. Therefore, we have to settle for optimistic execution time predictions based on the demand for resources. The predictions are optimistic because they assume that the use of resources is fully pipelined and overlapped. To see how this works, let us consider the execution time performance of applications running on a cluster of cycle servers. Suppose a scenario with 2 cluster nodes with demands for their CPUs of 80% and 40%. Our optimistic prediction strategy says that these applications could run with a 20% execution time degradation on 1 node ( $120 - 100 = 20$ ). Our algorithm should decide to remove one of the nodes, if a 20% degradation is acceptable. (In reality, 20% is a *lower bound* on the degradation.)

It is clear then that a key component of our algorithm is the demand for resources at each point in time. However, the decisions made by the algorithm must not be solely based on instantaneous demands to avoid reconfigurations triggered by transient load variations. The algorithm should also take into account the past history of demands and the speed of change in demands. *Control theory* provides a formal and well-understood approach to considering these properties. Thus, we use a Proportional-Integral-Differential (PID) feedback controller for each resource as the basis for our algorithm's decisions. The controller with the largest output (in absolute value) is used to determine the ideal cluster configuration at each point in time. The formula that describes the output,  $o(t)$ , of each controller is:

$$o(t) = k_p e(t) + k_i \sum_0^t e(t) + k_d \Delta e(t)$$

Each controller calculates the current excess demand (with respect to the current cluster configuration) for a resource, accumulates excess demand over time, and computes the rate of change in excess demands. These are the proportional, integral, and differential components of the controller, respectively. Each of these components is weighted with a tunable constant, which should reflect how much importance we want to give to each component. In our experiments, we used  $k_p = 0.7$ ,  $k_i = 0.15$ , and  $k_d = 0.15$ . Furthermore, we saturate the integral component at the resource capacity of a single node, i.e. 100% plus the acceptable performance degradation. (These constants and saturation value were chosen after some experimentation with our systems.) The output of the controller is the sum of the weighted components. The controller is executed every 10 seconds.

To guarantee stability, our algorithm computes excess demands with respect to the arithmetic mean of the resource capacities of  $N$  and  $N-1$  nodes for a configuration with  $N$  nodes. Moreover, the algorithm only triggers a reconfiguration if the absolute value of the controller's output is greater than half of the resource capacity of a single node plus 10% of this value. To decide how many nodes to add or remove, the algorithm divides the output of the controller minus half of the resource capacity of a node by the resource capacity of a node. For instance, for a 5-node configuration and a 20% acceptable performance degradation, excess demands would be computed with respect to 540% (the average of  $5 \times 120$  and  $4 \times 120$ ). In this scenario, a controller output of -65% means that the current configuration should not be altered ( $65 < 66$ ). An output of -300% should trigger the removal of two nodes ( $\lceil (|-300| - 60)/120 \rceil = 2$ ).

We refer to the acceptable degradation and the minimum time between reconfigurations as the `degrad` and `elapse` parameters of our algorithm. The `degrad` parameter can be specified by the cluster administrator or by each application (i.e. user). Ideally, the algorithm could also try to guarantee a *maximum* performance degradation. This is clearly not possible for execution time performance, but is conceivable for throughput performance. However, even in the case of throughput, such a strong guarantee cannot be made, given that the load on the cluster may increase faster than the system can react to such increase. Rather, we use our acceptable performance degradation parameter to *trigger* actions that can reduce or eliminate any degradation.

**Load (re-)distribution decision.** After an addition or removal decision is made, the load may have to be re-distributed. If the decision is to add one or more nodes, the algorithm must determine what part of the current load should be sent to the added nodes. Obviously, the load to be migrated should come from nodes undergoing excessive demand for resources.

If the decision is to remove one or more nodes, the algorithm must determine which nodes should be removed and, if necessary, where to send the load currently assigned to the soon-to-be-removed nodes. Obviously, the algorithm should give preference to lightly loaded victim nodes and destination nodes that would not undergo excessive demand for resources after receiving the new load.

The details of how to select victim nodes and of how to migrate load around the cluster depend on the system for which the algorithm is implemented, so we leave the description of these decisions for the next subsection.

## 4.2 Implementations

Our algorithm has been implemented with minor variations in two different environments: (1) at the application level for a network server that runs alone on a cluster; and (2) at the system level for a OS for clustered cycle servers.

In both implementations, the algorithm is run by a master node (node 0), which is a regular node except that it receives periodic resource demand messages from all other nodes and it cannot be turned off. We chose centralized implementations of the algorithm due to their simplicity and the fact that load messages can be infrequent. For fault tolerance, a distributed implementation would be best, but that is beyond the scope of this paper.

**Power-aware cluster-based network server.** We modified PRESS [20], a cluster-based, event-driven WWW server to implement our algorithm completely at the application level. The server is based on the observation that serving a request from any memory cache, even a remote cache, is substantially more efficient than serving it from a disk, even a local disk. Essentially, the server distributes HTTP requests across nodes based on cache locality and load balancing considerations, so that files are unlikely to be read from disk if there is a cached copy somewhere in the cluster. Since the cacheable files are static, each node stores a copy of all files on its local disk.

We implemented the cluster configuration and load distribution algorithm in the server making all nodes periodically inform the master node about their CPU, disk, and network interface demands. The CPU demand is computed by reading information from the `/proc` directory, whereas network and disk demands are computed based on internal server information. To smooth out short bursts of

activity, each of these demands is exponentially amortized over time using the following formula:  $\alpha \times \text{old\_demand} + (1 - \alpha) \times \text{current\_demand}$ . For our experiments,  $\alpha = 0.8$  and the interval between demand computations is 10 seconds. In case of the server, we are interested in throughput performance.

With information from all nodes, the master runs the cluster configuration and load distribution algorithm described in the previous section. If a removal decision is made, the master determines the maximum demand for any resource at each node and picks the node(s) with the lowest of the maximum demands as the victim. For the WWW server, it is not necessary to migrate load from a node to be excluded from the cluster. The load can be naturally redistributed among the remaining nodes, by the server's own HTTP request distribution algorithm and/or a load balancing front-end. Similarly, the addition of a new node to the cluster does not require migrating any load from other nodes to it.

Note that at the application level it is impossible to determine the demand for network interface (due to buffering in the kernel) and CPU (due to the fact that the server is single-process) resources, so our server cannot deal with a throughput degradation that is greater than 0%. In fact, in our experiments we assume that the resource capacity of a single node is either 70% or 85% of its actual capacity, i.e. we study `degrad` parameters of -30% and -15%. These values provide some slack to compensate for the time it takes for a node to be booted, approximately 100 seconds. We set the default value of the `elapse` parameter to 120 seconds. Given that the interval between demand computations is 10 seconds, this setting for `elapse` allows the server two observations of the state of a reconfigured cluster before another reconfiguration is permitted.

**Power-aware OS for clusters.** We modified Nomad [86], a Linux-based

single-system-image OS for clusters of uni and/or multiprocessor cycle servers. For the purposes of this paper, the most important characteristics of the OS are that (a) it has a shared file system; (b) it starts each application on the most lightly loaded node of the cluster at the moment; and (c) it performs dynamic checkpointing and migration of whole applications (with all its processes and state, including open file descriptors, static libraries, data, stack, registers and the like) between nodes to balance load. Resource demand is computed for each node in the OS, by checking the resource queues every second. Whenever the average CPU demand, the memory consumption, or the I/O demand observed locally at a node remains higher than a threshold for 5 seconds, the OS considers the node to be undergoing excessive demand and attempts to migrate some of its load out to a more lightly-loaded node with respect to the heavily demanded resource.

To avoid excessive migration activity, the migration of an application can only happen if a few conditions are verified. First, an application can only be migrated if it has already executed at least as long as the estimated time to migrate a process of its size. Second, a node that has just migrated an application elsewhere will not migrate another one until a period of stabilization, currently set to 30 seconds, has elapsed. Third, no incoming migration will be accepted by a node that has been either the source or the destination of a migration during the stabilization period. Finally, the OS was designed for clustered cycle servers, i.e. time-shared execution of sequential applications on uniprocessor nodes and of parallel applications on multiprocessor nodes, so applications that do not conform to these restrictions cannot be migrated by the system.

Again, we implemented the cluster configuration and load distribution algorithm in the OS making all nodes periodically inform the master node about their CPU, memory, and I/O demands. The CPU demand and the memory consumption are computed by reading information from `/proc`, whereas I/O demands are determined by instrumenting read and write system calls and getting swap information from `/proc`. To smooth out short bursts of activity, the demands are amortized using the same formula used by the WWW server. For our experiments,  $\alpha = 0.8$  and the interval between demand computations is 10 seconds. In case of the OS implementation of our algorithm, we are interested in execution time performance.

With information from all nodes, the master can run our algorithm. If a removal decision is made, the master selects the nodes with the lowest demands for each resource as candidate victims. Unlike the WWW server, in the OS case the load of the victim must be migrated to other nodes, so the master selects the two nodes with the lightest load with respect to each resource (CPU, I/O, and memory) and selects the source/destination pair that would lead to the lowest overall demand for resources. To simplify our prototype implementation, the destination node receives all applications that are running on the victim node. Any load imbalances are later corrected by the OS according to its load balancing policy.

In the modified OS, a node addition is not effected if only one application is responsible for the excessive demand. After a new node is turned on, the OS will start migrating applications to it, so that the load will be balanced again. Given that adding nodes takes a significant amount of time (about 100 seconds), it might take a while before the demand for resources becomes acceptable again,

after a long-lasting surge of activity. We experiment with two values for `degrad`: 0% and 20%. The `elapse` parameter is set to 150 seconds. This setting allows the system time to reconfigure, migrate applications to balance the load, and re-evaluate the resource demands before another reconfiguration is allowed.

### 4.3 Methodology

To study the performance of our algorithm and systems, we performed experiments with a cluster of 8 PCs connected by a Fast Ethernet switch and a Gigabit switch. Each of the nodes contains an 800-MHz Pentium III processor, 512 MBytes of memory, two 7200 rpm disks (only one disk is used in our experiments), and two network interfaces. Shutting a node down takes approximately 45 seconds and bringing it back up takes approximately 100 seconds.

All machines are connected to a power strip that allows for remote control of the outlets. Machines can be turned on and off by sending commands to the IP address of the power strip. The total amount of power consumed by the cluster nodes is then monitored by a multimeter connected to the power strip. The multimeter collects instantaneous power measurements 3-4 times per second and sends these measurement to another computer, which stores them in a log for later use. We obtain the power consumed by different cluster configurations by aligning the log and our systems' statistics.

**Network server experiments.** Besides the main cluster, we use another 14 Pentium-based machines to generate load for the modified WWW server. For simplicity, we did not experiment with a front-end device that would hide the powering down of cluster nodes from clients. Instead, clients poll all servers every

10 seconds and can thus detect cluster reconfigurations and adapt their behavior accordingly. The clients send requests to the available nodes of the server in randomized fashion according to a trace of the accesses to the World Cup '98 site from 12pm on 07/12 to 12pm on 07/14. The trace includes the day of the championship match. To shorten the duration of the experiments, we accelerated the trace 20 times.

**Distributed OS experiments.** The synthetic workload used for our modified OS experiments draws applications from a number of sources: all integer applications from the SPEC2000 benchmark, the Berkeley MPEG movie encoder, and two I/O benchmarks, IOcall and IOzone. IOcall is a benchmark to measure OS performance on I/O calls, especially file read system calls. IOzone is a file system benchmarking tool [61]; it generates and measures the performance of a variety of file operations. Applications are arbitrarily assigned to nodes and are run in arbitrary groups. Because the cluster size varies dynamically according to the resource demand imposed on it, we start with only one machine powered on (the master), which is responsible for launching all applications in the workload. The offered demand conforms to a bell-shaped curve. To shorten the length of the experiments, we generate significant changes in offered demand in very little time.

## 4.4 Experimental Results

**Power-aware cluster-based network server.** We describe two experiments with our server. In the first experiment, the parameters for the cluster reconfiguration algorithm are set to guarantee quick reaction to fluctuations in load, so

that we can tackle significant increases in load without performance degradation. More specifically, the algorithm tries to keep 30% spare resources for any cluster configuration. Figure 4.1 presents the evolution of the cluster configuration and demands for each resource as a function of time in seconds for this experiment. The demand for each resource is plotted as a percentage of the nominal throughput of the same resource in one node.

The figure shows that for this particular workload the network interface is the bottleneck resource throughout the whole execution of the experiment (140 minutes). We started the experiment with a two-node configuration. The traffic directed to the server initially increases slowly, triggering the addition of a node, before increasing substantially and triggering the addition of several new nodes in quick sequence. The traffic then subsides, until another period of high traffic occurs, which is followed by a substantial decline in traffic. Note that throughout the experiment the system reacts quickly to increases in traffic, because of the spare capacity it consistently retains. As a result of the spare capacity, the performance of the server is not affected by the dynamic reconfiguration of the cluster.

Figure 4.2 presents the power consumption of the whole cluster for two versions of the same experiment as a function of time. The lower curve (labeled “Dynamic Configuration”) represents the version in which we run the power-aware server, i.e. the cluster configuration is dynamically adapted to respond to variations in resource demand. The higher curve (labeled “Static Configuration”) represents a situation where we run the original server, i.e. the cluster configuration is fixed at 7 nodes. As can be seen in the figure, our modified WWW server can reduce power consumption significantly for most of the execution of our experiment.

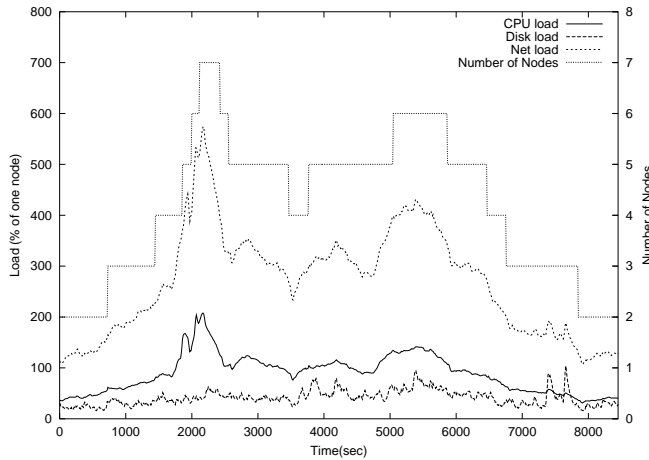


Figure 4.1: Cluster evolution and resource demands for the WWW server (dynamic configuration). `elapse = 120` seconds; `degrad = -30%`.

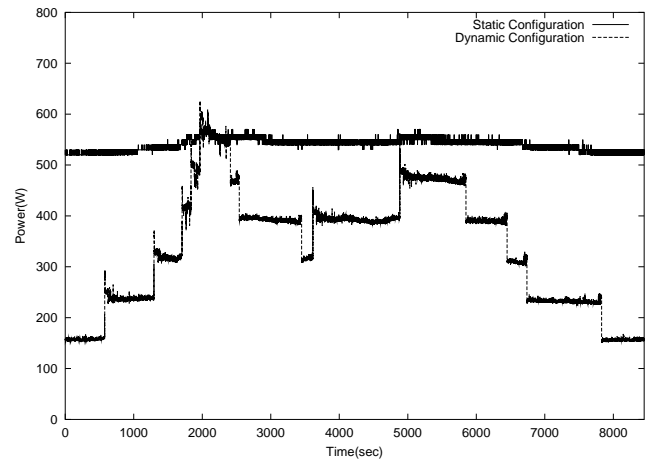


Figure 4.2: Power consumption for the WWW server under static and dynamic cluster configurations. `elapse = 120` seconds; `degrad = -30%`.

Power savings actually reach 71% when the resource demands require only two nodes. Our energy savings are also significant. Calculating the area below the two curves, we find that the power-aware server saves 38% in energy. Thus, the load on the cooling infrastructure over time is also reduced by 38%.

Even though these are significant gains, we can do better. The reason is that keeping spare capacity promotes performance at the cost of higher power and energy consumption. If we can estimate how fast the offered traffic can increase, we can reduce the spare capacity to the minimum required to avoid excessively long request latencies. For our trace, this minimum is 15%. Thus, figure 4.3 presents the evolution of the cluster configuration when the system attempts to retain this much spare capacity. Comparing figures 4.1 and 4.3 we can see that during most of the experiment the system now requires fewer active nodes to handle the offered load. In this case, the power and energy gains that can be achieved in comparison to a static system with 7 nodes are 71% and 45%, respectively.

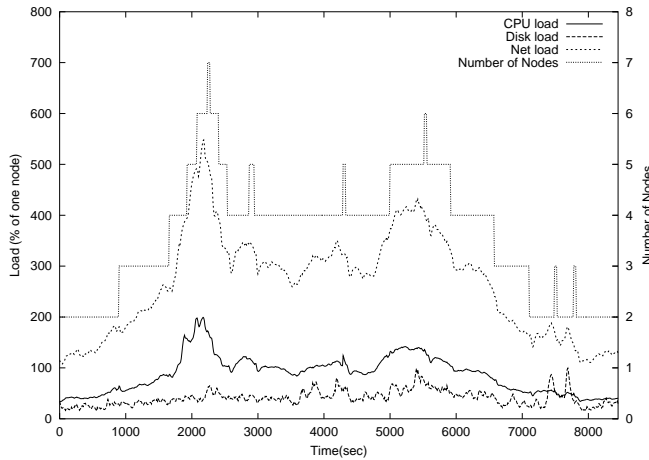


Figure 4.3: Cluster evolution and resource demands for the WWW server. `elapse` = 120 seconds; `degrad` = -15%.

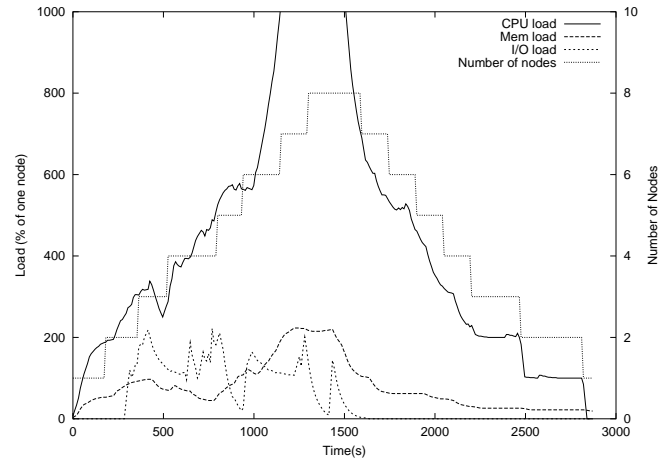


Figure 4.4: Cluster evolution and resource demands in the power-aware OS. `elapse` = 150 seconds; `degrad` = 0%.

In general, it might not be possible to determine the maximum rate of workload change a priori. In these cases, mismatches between the rate of workload change and cluster reconfiguration delays can be alleviated by adjusting the `elapse` and/or `degrad` parameters dynamically. We believe however that in practice values of a few minutes for `elapse` and a few percent for `degrad` should work just fine, since real network server workloads usually change more slowly than in our experiments.

**Power-aware OS for clusters.** Figure 4.4 presents the evolution of the cluster configuration and demands for each resource with `elapse` = 150 seconds and `degrad` = 0%, as a function of time. The experiment lasted about 46 minutes. The CPU is always the bottleneck resource during the experiment. The experiment starts with a single-node configuration. This node is responsible for starting all the applications in the workload. As new applications are started, the CPU demand increases and eventually triggers the addition of a new node. When the new node is added by the master, the OS attempts to balance the load

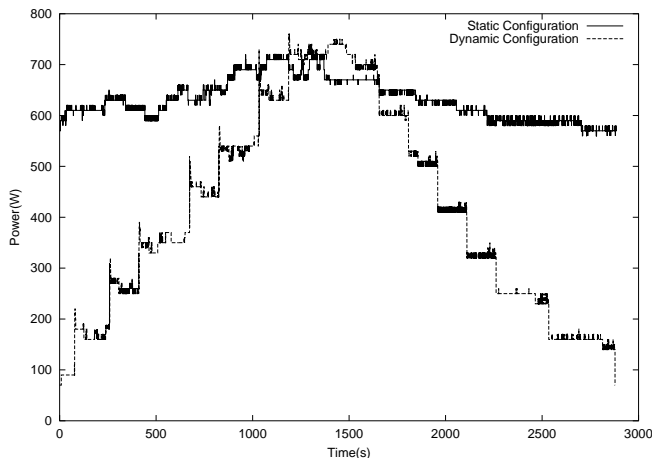


Figure 4.5: Power consumption for the power-aware OS under static and dynamic cluster configurations.  $\text{elapse} = 150$  seconds;  $\text{degrad} = 0\%$ .

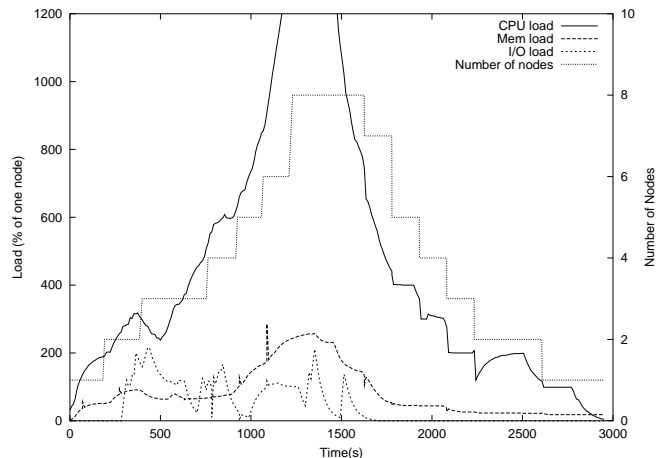


Figure 4.6: Cluster evolution and resource demands in the power-aware OS (dynamic configuration).  $\text{elapse} = 150$  seconds;  $\text{degrad} = 20\%$ .

by migrating some applications to the new node. As the number of applications started increases, they trigger the addition of other nodes, one at a time. The OS is able to track the demand increases fairly well by increasing the size of the cluster. At about half way through the experiment, the demand for CPU becomes much higher than can be managed by an 8-node cluster. Right after this peak in demand however, some applications start to finish and the demand for resources drops quickly. The master responds to this change in load by excluding the now idle nodes, one at a time. Again, the system does a good job of tracking the decrease in resource demand.

Figure 4.5 presents the power consumption of the whole cluster for two versions of the same experiment as a function of time. As can be seen in the figure, our power-aware OS can reduce power consumption significantly for most of the execution time of the experiment. Power savings <sup>1</sup> actually reach 88% when the

---

<sup>1</sup>Difference between maximum and minimum power dissipation during the execution of the experiment.

resource demands require only a single node. Energy savings are also significant. The area below the two curves indicates that the power-aware OS saves 35% in energy for this workload.

It is interesting to note that the workload used in this experiment finishes a little earlier on the static configuration (around 45 minutes) than on the dynamic one (around 46 minutes). If we compare the energy consumed by the static configuration during the first 45 minutes of the experiment against that of the dynamic configuration for the whole experiment, we find that our energy savings are only slightly smaller, 32%. (This comparison is not really fair however, since real, i.e. static, cycle servers are never turned off). In any case, it is clear that the load on the cooling infrastructure is reduced by at least 32% under the dynamic system.

To investigate the tradeoff between performance and power, we also performed experiments in which our intended performance degradation is 20%. We kept `elapse` at 150 seconds. Figure 4.6 illustrates the evolution of the cluster configuration in this case. As one would expect, allowing for some performance degradation has the effect of slowing down the addition of new nodes and speeding up the removal of unnecessary nodes. As a result, the system decides to jump directly from 6 to 8 nodes when ramping up and to jump directly from 7 to 5 nodes when down-sizing. Overall, our system conserves 88% and 42% of the power and energy consumed by its static counterpart in this experiment. If we consider that the workload finishes 2 minutes later on the dynamic than on the static configuration, the energy gains are of 40%.

## 4.5 Conclusions

In this chapter, we addressed power and energy conservation for clusters. We proposed a control-theoretic cluster configuration and load distribution algorithm and applied it under two different scenarios. Our experiments showed that it is indeed possible to conserve significant power and energy in the context of clusters. Based on our experimental results, we conclude that exploiting periods of light load and matching the offered resources can provide tremendous gains for organizations and companies that rely on large clusters of servers

## Chapter 5

# Popular Data Concentration

In this chapter, we describe the Popular Data Concentration (PDC) technique and the Nomad File Server. PDC is a new energy conservation technique for disk array-based network servers. The idea behind PDC is to dynamically migrate the popular disk data (i.e., the most frequently *and* recently accessed data on disk) to a subset of the disks in the array, so that the load becomes skewed towards a few of the disks and others can be sent to low-power modes. PDC is based on the observation that network server workloads often exhibit files with widely different popularities. For instance, Web server workloads are known to exhibit highly skewed popularity towards a small set of files.

To demonstrate the benefits of PDC, we also propose a user-level file server, called Nomad FS, that takes advantage of our technique. We envision coupling Nomad FS with one or more front-end network servers. Nomad FS uses the MQ algorithm [108] for second-level cache replacement to guide the placement of files in the disk array dynamically. The current prototype of Nomad FS conserves energy by spinning disks down after a period of idleness. Unfortunately, spinning a disk back up to service a request increases the response time for the request significantly.

We also study Nomad FS in the context of disk arrays composed of two-speed disks [21]. These disks conserve energy by automatically reducing their rotational

speeds under light load. Thus, Nomad FS does not have to explicitly perform power mode transitions when two-speed disks are used. Furthermore, requests are only delayed during rotational speed transitions, which means that only a small percentage of requests is delayed for stable workloads.

Because our work targets network servers, degrading the response time of a small fraction of requests is acceptable for Nomad FS. In fact, the overhead of multiple message exchanges per request in network servers means that higher server response times can be accommodated. Moreover, network servers typically interact with people (or background processes), so even substantial degradations may not be noticeable. For example, a degradation from 10 to 100 milliseconds is usually acceptable for network servers.

To put PDC in context, we compare Nomad FS against a very similar version of it that is based on the Massive Array of Idle Disks (MAID) [25].

The behavior of the systems we consider is affected by several important parameters, such as the type of disks, the disk request rate, the size of the main memory cache, and the percentage of the stored files that is actually accessed. To understand the effect of these parameters and determine the “sweet spot” for each technique, we assess the parameter space extensively using a simulator and synthetic traces. We also simulate two real traces to confirm the observations from the parameter space exploration. To guarantee the accuracy of our results, we validate the simulator against real executions of our prototype implementation of Nomad FS.

Our results for arrays of conventional disks show that PDC and MAID can only conserve energy when the load on the server is extremely low. When arrays of two-speed disks are used, both PDC and MAID can conserve up to 30-40% of

the disk energy with only a small fraction of delayed requests. Overall, PDC is more consistent and robust than MAID; the behavior of MAID is highly dependent on the number of cache disks. Furthermore, PDC achieves these properties without the overhead of extra disks. However, the PDC energy savings degrade substantially for long migration intervals. Based on these results, we conclude that the techniques we study will be very useful for disk array-based servers when multi-speed disks become available.

## 5.1 Popular Data Concentration

Several types of network servers exhibit workloads with highly skewed file access frequencies. For example, it has been shown that the frequency of file access by a Web server conforms to a Zipf distribution [18] with high coefficient. The same is true of other servers as well, e.g. [63]. More formally, Zipf’s law predicts that the frequency of access or popularity  $\tau$  of a file is proportional to the inverse of its rank  $r$  raised to some coefficient  $\alpha$ , i.e.  $\tau = 1/r^\alpha$ . When  $\alpha$  is high (close to 1), a relatively small number of files is accessed frequently, whereas a large number of files is accessed rarely. Workloads with high  $\alpha$  often exhibit skewed popularity in terms of disk accesses as well (albeit with smaller  $\alpha$ ), even in the presence of large main memory caches [19].

PDC is inspired by such heavily clustered popularities. The idea behind PDC is to concentrate the most popular disk data (i.e., those that most frequently miss in the main memory cache) by migrating it to a subset of the disks. This concentration should skew the disk load towards this subset, while other disks become less loaded. These other disks can then be sent to low-power modes to

conserve energy. More specifically, the goal of PDC is to lay data out across the disk array so that the first disk stores the most popular disk data, the second disk stores the next set of most popular disk data, and so on. The least popular disk data and the data that are never accessed will then be stored on the last few disks. In fact, the last few disks will also include the data that most frequently hit in the main memory cache.

To avoid performance degradation, it is important not to overload the disks that store popular data. Thus, the expected access rate for each disk needs to be considered explicitly. This is done by estimating the future load (in MBytes/second) on each disk to be the sum of the recent load directed to the data to be stored on it. PDC should then only migrate data onto a disk until the expected load on the disk is close to its maximum bandwidth for the workload.

Because data popularity can change over time, PDC may have to be applied periodically. In such cases, it might be necessary to free up space on a disk by migrating data out of it, before more popular data can be migrated in.

## 5.2 Nomad FS

PDC has been implemented in Nomad FS, a prototype energy-aware file server consisting of approximately 13k lines of C++ code. Nomad FS is a user-level, event-driven server that works on top of the local file system. The server associates a helper thread with each disk. This thread is the only part of the server that directly touches the disk, either for read/write operations or migrations. Although the server receives requests for 8-KByte file blocks, entire files are migrated according to PDC. This approach works fine for the workloads we are interested

in, such as Web, proxy, ftp, and email server workloads, which access entire files at a time.

To conserve energy in disk arrays composed of conventional disks, PDC is used to idle disks, which are then spun down by the server after a fixed period of idleness (the idleness threshold). The server determines that it is time to spin down a disk by keeping the last access time per disk and periodically testing whether any disk has been idle for longer than the idleness threshold. A spun down disk is reactivated on the next access to it. When we consider disk arrays composed of two-speed disks, Nomad FS does not explicitly effect power mode transitions. Furthermore, Nomad FS does not migrate files out of disks that are already running in low speed to reduce the migration overhead.

The file server exports one single view of the file system, although multiple disks drives can be used. For simplicity of our prototype, each file is permanently stored on one disk only, i.e. no striping or mirroring is used at this time. The user has no control over where (on the disk array) files are stored, since that information can change dynamically. New files are created on any of the disks with enough free space. Metadata information is kept in a well-known location on the disk that stores the most popular files. The metadata for each file contains the file name (at most 128 bytes), the size of the file, the disk on which the file resides, the times of creation and last access, and bookkeeping information (pointers to next and previous files on the same disk, etc). Overall, each metadata entry consumes 168 bytes. The metadata for all files is kept in a large file that is memory mapped to the virtual address space of the server. Only when a file is actually opened is its metadata accessed and physical memory allocated for it.

Nomad FS caches data blocks in memory at the user level. Because the cache

is not a first-level cache (it sits behind the clients' and/or network servers' caches), the standard LRU replacement policy would not work well for this cache. Instead, Nomad FS uses the Multi Queue (MQ) algorithm [108] to manage its block cache. MQ has been shown superior to other algorithms, including sophisticated variations of LRU.

The MQ cache works as follows. There are multiple LRU queues numbered  $Q_0, Q_1, \dots, Q_{m-1}$ ; in our prototype  $m = 12$ , but the system behavior is typically similar for other large numbers of queues. Blocks stay in the LRU queues for a given *lifetime*. This lifetime is defined dynamically by the MQ algorithm to be the maximum temporal distance between two accesses to the same file or the number of cache blocks, whichever is larger. If a block has not been referenced within its lifetime, it is demoted from  $Q_i$  to  $Q_{i-1}$  or evicted from the cache if it is in  $Q_0$ . Each queue also has a maximum access count. If a block in queue  $Q_i$  is accessed more than  $2^i$  times, this block is then promoted to  $Q_{i+1}$  until it is accessed more than  $2^{i+1}$  times or its lifetime expires. Within a given queue, blocks are ranked by recency of access, according to LRU.

During operation of the server, information about the files referenced in the past are summarized on a list in main memory. The summary for each file stores the file descriptor, the number of disk accesses for the file, a pointer to its metadata, and some other bookkeeping data (such as list pointers), totaling 60 bytes. The amount of memory dedicated to the list of summary information should be large enough to hold at least the number of files in the working set of a given workload.

The ranking of file popularity is done incrementally every time a file access requires a disk access. We do this by reusing the same MQ cache algorithm

described above, but instead of using it on cache blocks, we use it on the summary information. More specifically, we keep the summary information on an MQ cache-like list. Every disk access causes the file's summary data to move on the list according to the MQ policy. More disk accesses cause the summary entry to move closer to the head of the list. After a while, popular files (in terms of disk accesses) will be close to the head of the list, while unpopular files will be towards the tail.

Periodically, files are migrated to disks based on this ranked list of disk accesses. The algorithm traverses the queues from  $Q_{m-1}$  to  $Q_0$ . Initially, files in queue  $Q_{m-1}$  are migrated to the first disk (numbered 0) until it is full or the expected load on the disk approaches its maximum bandwidth for the workload. The expected load associated with a file is estimated by dividing its size by its average inter-access time. The next set of files is migrated to the second disk (according to the same conditions) and so on. If the server needs to migrate a file  $F$  to disk drive  $i$ , there is no space available on  $i$ , and a file  $L$  on  $i$  is less popular than  $F$ , then file  $L$  is migrated to disk  $z$ . Disk  $z$  is selected as follows. First, the server tries the disks holding more popular files, starting from disk 0 and checking all disks until disk  $i - 1$ . If not enough space and bandwidth can be found there, the server checks the disks holding less popular data, starting from disk  $i + 1$  and checking all higher numbered disks. This algorithm should always find space for the file, as long as the overall disk space is not excessively tight for the size of the file system. We have not developed an algorithm to deal with this last scenario, because it has not been a problem for us.

### 5.3 Comparison Against Other Approaches

**MAID.** As previously said, MAID has been proposed as a replacement for old tape backup archives with hundreds or thousands of tapes. Because only a small part of the archive would be active at a time, the idea is to copy the required data to a set of “cache disks” and put all the other disks in low-power mode. Accesses to the archive may then find the data on the cache disk(s). Cache disk replacements are implemented using an LRU policy. Replaced data can simply be discarded if it is clean. If it is dirty, it has to be written back to the corresponding non-cache disk.

This same idea can be applied to file servers as well. In fact, we developed a version of Nomad FS that uses MAID to conserve energy. We used as much code from Nomad FS as possible for our MAID-based file server, so that the two servers can be directly compared. The main memory cache management, for instance, is exactly the same for both implementations. On an access to a cached file block, the access is performed on the corresponding cache disk. If the block accessed is not cached, the entire file is copied by the server from its location on one of the non-cache disks to one of the cache disks. If a replacement is required on a cache disk, an entire file that is large enough is replaced according to LRU.

The designers of MAID observed that cache disks can become overloaded [25]. To counter this problem, we optimize MAID by avoiding copying files to the cache disks when the recent load on these disks is approaching their maximum bandwidth. We find that the absence of this optimization always leads to a large percentage of delayed requests, as the cache disks become a bottleneck.

To conserve energy in configurations with conventional disks, the MAID version of our server explicitly spins disks down after a period of idleness. When considering two-speed disks, the server does not control power modes explicitly and does not copy data out of disks that are already at low speed.

In section 5.5, we quantitatively compare the MAID version of our file server against the PDC version. Here we focus on a qualitative comparison. PDC and MAID have the same objective, namely to increase idle times by moving data around the disk array and spinning disks down. As a result, both techniques sacrifice the access time of certain files in favor of energy conservation. However, instead of relying on file popularity and migration like PDC, MAID relies on temporal locality and copying to conserve energy. This has a few potential disadvantages in terms of energy savings: (1) the cache disk(s) represent energy overhead, since they are additional to the set of disks that are required to store the actual data; (2) the cache disk(s) might not have enough space to store the entire (dynamically changing) working set for a given workload, causing constant accesses to the non-cache disks; and (3) files are randomly spread on the non-cache disks, which may significantly reduce the opportunity for energy conservation if requests miss in the cache disk(s) frequently. Nevertheless, MAID does have a few potential advantages: (1) metadata management is significantly simpler, since it does not need to store and operate on file access information about all files, only those that are on cache disks; and (2) it adapts faster to changes in workload behavior, because file copying (and possibly replacement) take place on a per-file access basis.

**FT.** When considering disk arrays composed of conventional disks, we also study the fixed-threshold (FT) technique. We set the idleness threshold to the amount

of time for which the energy consumed in idle state is the same as that of powering the disk down and later powering the disk back up. The rationale for this definition is similar to the competitive argument about renting or buying skis [67]. A spun down disk is reactivated on the next access to it.

FT does not involve any data movement or re-organization and, thus, is not very effective at conserving energy in busy servers. In contrast, it is very useful when combined with PDC or MAID, as described above. We use an FT-based version of our file server as a basis for comparison in section 5.5. Again, we re-used most of the Nomad FS code to implement our FT-based file server.

## 5.4 Methodology

Our main goals in this chapter are to determine the conditions under which PDC is effective at conserving energy and how it compares to MAID and FT for two disk array types and a wide range of parameters. We explore this parameter space using the simulator that we describe in section 5.4.1.

The reason why we chose simulation rather than real experimentation is two-fold: (1) two-speed disks are not yet available on the market (although Sony already has a disk drive that allows for static speed setting [81]); and (2) using real experiments to perform such a comprehensive parameter space exploration would have taken extremely long. For example, each of our simulations involves 19 million requests at 750 requests/second. A real experiment with these parameters takes 7 hours, whereas a simulation takes less than 40 minutes. Note that accelerating a real execution and then scaling results back would not have worked, since it is not possible to accelerate real disk spin up/spin down operations, file

copying, or file migration.

Nevertheless, we strongly believe in realistic simulations, so we carefully validated the simulator against executions of our prototype file server implementations on our own server hardware. We discuss the validation of the simulator in subsection 5.4.2.

Another advantage of using simulation is that we can easily vary parameters, which is a key component of this paper. In subsection 5.4.3 we discuss the set of parameters that we have found most relevant in evaluating and comparing the different file servers.

We drive the simulator with both synthetic and real traces. Subsection 5.4.4 describes our generator of synthetic traces, whereas subsection 5.4.5 describes our real traces.

### 5.4.1 Simulation

We have developed an execution-driven simulator of our prototype file servers. The simulator mimics the implementation of our servers as closely as possible. In fact, several parts of the real servers were reused in the simulator to avoid deviations from the real implementations. The MQ replacement algorithm implementation, for example, was copied straight out of the server code.

At the file system level, files are assumed to be laid out in consecutive blocks of the same disk, whereas i-nodes are assumed to be cached in memory. At a lower level, the simulator models an array of conventional Cheetah disks or an array of two-speed disks. The main characteristics of the conventional disk are shown in table 5.1. The table also lists the idleness threshold for this disk, which is the sum of the energies to spin the disk up and down divided by the idle power.

Description	Value
Disk model	Seagate Cheetah ST39205LC
Standard interface	SCSI
Storage capacity	9.17 GBytes
Number of platters	1
Rotational speed	10000 rpm
Avg. seek time	5.4 msec
Avg. rotation time	3 msec
Transfer rate	31 MBytes/sec
Idle power	5.26 Watts
Down power	1.86 Watts
Active energy (8-KB read)	61 mJoules
Spin up energy	65.91 Joules
Spin down energy	28.25 Joules
Spin up time	6.12 secs
Spin down time	11.24 secs
Idleness threshold	17.9 secs

Table 5.1: Main characteristics and measured power, energy, and time statistics of our SCSI disk.

The main characteristics of the two-speed disks are shown in table 5.2. The high speed values are those of the Cheetah disk. These values were scaled down to produce the low speed values. In particular, the power and energy values were scaled down in quadratic fashion [45], assuming that  $power = c \times speed^2 + 1.86$  where  $c$  is the constant  $(5.26 - 1.86)/10K^2$ . The disk controller slows the disk down when the offered load becomes lower than 80% of the disk throughput at low speed. Conversely, the controller transitions the disk to high speed when this same threshold is exceeded. The controller can trigger these transitions by observing the actual utilization of the disk. We assume that no accesses can proceed when a disk is transitioning speeds. The speed transition costs are set to half of the corresponding costs for the Cheetah disk. For example, going from low to high speed costs half of the energy and time to spin up the Cheetah disk. These transition cost settings are admittedly somewhat arbitrary. However, note that

Description	Value
Storage capacity	9.17 GBytes
Avg. seek time	5.4 msec
High to low speed transition energy	14.13 Joules
High to low speed transition time	5.62 sec
Low to high speed transition energy	32.96 Joules
Low to high speed transition time	3.06 sec
High rotational speed	10000 rpm
Avg. rotation time at high speed	3 msec
Transfer rate at high speed	31 MBytes/sec
Idle power at high speed	5.26 Watts
Active energy at high speed (8-KB read)	61 mJoules
Low rotational speed	3000 rpm
Avg. rotation time at low speed	10 msec
Transfer rate at low speed	9.3 MBytes/sec
Idle power at low speed	2.17 Watts
Active energy at low speed (8-KB read)	43 mJoules

Table 5.2: Main characteristics and simulated power, energy, and time for our two-speed disk. The high speed values are those of our Cheetah disk (table 5.1). These values were scaled down to produce the low speed values. The speed transition costs are set to half of the corresponding costs in the Cheetah disk.

the transition costs make little difference in our simulations, since the frequency of speed transitions is low for the parameters we consider. This same effect has been observed in previous work [21].

Finally, the PDC and MAID-based servers involve four major sources of overhead that do not exist in the FT version of our server: migrations, copies, LRU processing, MQ processing. The CPU energy consumed by these operations has not been considered so far. To take this consumption into account, we determined these overheads by running three microbenchmarks. One of them isolates the CPU energy involved in each file block request to be about  $37 \mu\text{J}$  for both LRU and MQ processing. The other two microbenchmarks isolate the CPU energy consumed by each migration and copy operation, as a function of file size. We have approximated these costs to  $1.5 \mu\text{J}$  per byte moved. The simulator

Description	Value
Request rate	4 file requests/sec
Trace length	9033 secs (2.5 hours)
Number of files	43690 *
File size	20 KBytes *
Total file system size	853 MBytes *
Main memory cache size	64 MBytes *
Number of disks	4
Disk capacity	342 MBytes *
Idleness threshold	17.9 secs
Migration period	every 1.25 hours (PDC only)

Table 5.3: Values used in the simulator validation: workload characteristics (top), server settings and disk characteristics (bottom). Values marked with “\*” are used for fast validation.

Description	Total energy consumed			Files moved		MBytes moved		Spin downs/ups		Delayed requests	
	Sim	Real	Error	Sim	Real	Sim	Real	Sim	Real	Sim	Real
PDC	152023	172843	12.0%	9715	10943	227	283	336	326	1.3%	1.2%
MAID	141349	156335	9.6%	8738	8716	204	204	300	304	1.1%	1.0%
FT	190374	200479	5.0%	n/a	n/a	n/a	n/a	10	13	0.2%	0.2%
EO	190346	200833	5.2%	n/a	n/a	n/a	n/a	n/a	n/a	0.0%	0.0%

Table 5.4: Summary of simulator validation. Energy values are in Joules.

charges each request, migration, and copy with these CPU energy costs.

The simulator faithfully models the disk powers, energies, and times discussed above to determine disk energy consumption and response times. The CPU energy costs associated with PDC and MAID are added to their disk energy consumption.

## 5.4.2 Simulator Validation

It is important to build confidence in our simulator by validating it against real experiments. Unfortunately, we can only validate the simulations that assume conventional disks.

Our server hardware consists of an ASUS A7A133 motherboard with an Intel

Pentium-4 1.9-GHz processor, 512 MBytes of memory, 4 Seagate Cheetah 9.1-GByte Ultra SCSI disks, one Adaptec 29160 Ultra 160 SCSI controller, and one Gigabit Ethernet network adaptor. The energy consumed by the disks is monitored by a TDS 3014 oscilloscope. The oscilloscope is connected to a remote computer that collects power consumption data at 1.25 Gsamples/second. These data are averaged every 10 milliseconds. A 1.2-GHz PC connected to the server via a Gigabit Ethernet connection generates load for the server.

We used a synthetic workload to validate the simulator against our prototype file servers running on this server hardware. Our workload generator (explained in detail below) keeps selecting a file to access randomly out of a set of files for a user-specified amount of time. All the blocks of each chosen file are requested in sequence. The timing of the file requests follows an exponential distribution. The main characteristics of the synthetic workload and the server settings are summarized in table 5.3. Note that some of the workload characteristics and server settings (those marked with “\*”) are not realistic; they are meant exclusively for fast validation.

We collected results for our file servers, as well as an energy-oblivious server (EO) under which disks are never powered down. The MAID version used one of the 4 disks as a cache disk, so it did not incur any energy overhead due to the cache disk. Table 5.4 compares the servers in terms of disk energy consumption, files and bytes moved between disks, number of disk spin downs/ups, and the percentage of delayed requests. We consider a file request to be delayed if it takes longer than 200 milliseconds.

The results show that the total energy consumed by each server matches closely (within 5 to 12%) that of the simulated system. The results also match

the other metrics closely, especially the number of spin downs/ups, which is a key metric for energy conservation. These results give us confidence that our simulator can be used to illustrate the important trends across the parameter space.

### 5.4.3 Parameter Space

Three categories of parameters directly affect the energy conservation techniques and file servers we study: workload characteristics, server characteristics, and disk drive characteristics. Next, we discuss each of these categories in turn.

**Workload characteristics.** These are characteristics that are inherent to the workload imposed on the file server. Among the large set of parameters that describe a workload, we have identified five key characteristics: the request rate, the file popularity, the coverage of the file system, the percentage of writes, and the temporal correlation of accesses to files.

The *request rate* has to do with the rate at which file requests arrive at the server. We assume that inter-arrival times are exponentially distributed; the request rate is the average of the distribution.

File *popularity* corresponds to the frequency with which each file is accessed. We represent popularity by a value of the  $\alpha$  coefficient in Zipf's power law. This coefficient determines how skewed the distribution of accesses to files is. A smaller  $\alpha$  (close to 0) means that requests are more evenly distributed among files, whereas a larger  $\alpha$  (close to 1) means that requests are more skewed towards a few files.

The file system *coverage* has to do with the percentage of the entire file repository that is actually accessed by the workload. A coverage of 100% means that all files in the file system are accessed at least once.

The *percentage of writes* corresponds to the frequency of file write operations in the workload.

Temporal locality describes how far apart in time consecutive accesses to the same files are. In particular, workloads with high temporal locality exhibit consecutive accesses to the same file that are “close” in time, whereas workloads with low temporal locality exhibit consecutive accesses to the same file that are “far apart” in time. Temporal locality has a direct relationship with popularity, i.e., popular files have a short inter-arrival time due to their popularity. However, temporal locality can also be an effect of *temporal correlation*, as discussed in [64]. Files with high temporal correlation are those for which accesses are close together in time regardless of the absolute popularity of the file. In this study, we are interested in the effect of both aspects of temporal locality on energy conservation.

**File server characteristics.** The configuration of the file server can impact the energy savings significantly. Important configuration parameters are the main memory cache size, the cache replacement policy, and the number of disks used. We consider the *cache size* and the *number of disks* in this study. We keep the cache replacement policy fixed on the MQ algorithm, which has been shown to be the best policy for second-level caches such as the ones used in our file servers [108].

**Disk drive characteristics.** Several characteristics of physical disk drives affect energy consumption directly, especially the power consumption in each mode and

the spin up/down overheads. Rather than focusing on a detailed exploration of this space however, we concentrate on evaluating the systems we study for disk arrays composed of either *conventional* or *two-speed disks*.

#### 5.4.4 Synthetic Trace Generator

To understand the impact of the parameters we just discussed, we built a workload generator that can produce request traces with user-specified request rate, popularity, coverage, temporal correlation, number of requests, file system size, and average file size. Currently the generator assumes that all files are of the same size for simplicity.

To generate a request trace, the generator must first compute the total number of files in the file system, by dividing the file system size by the file size. With the total number of files, the generator can determine the number of files that must be requested, according to the desired coverage. Each of the files to be requested then receives a probability weight, based on Zipf's formula  $1/i^\alpha$ , where  $i$  is the rank of the file from 1 to the number of files to be requested. Based on these probabilities, the generator randomly selects the actual file requests that will make up the trace. All blocks of a selected file are requested in sequence. Each new request is assigned a timestamp that is equal to the time of the last access plus an exponentially distributed random increment based on the request rate.

The generator continues to select files based on their weights until the number of requests reaches the desired number. If the desired coverage is still missing  $N$  files when the trace is missing  $N$  requests, a request for each of these  $N$  files is appended to the trace sequentially. However, to avoid having a long sequence of

previously unreferenced files in the end of the trace, we randomize the trace once all the files have been selected. We have verified that this procedure produces the desired coverage without changing the desired popularity ( $\alpha$ ) by inspecting several generated workloads.

However, this generation procedure does not consider temporal correlation explicitly. As described, the temporal locality of the workload is the one inherent to the popularity of files. The workloads we consider have this form of temporal locality, except when we explicitly address the effect of temporal correlation. To generate a set of workloads with varying degrees of temporal correlation, the generator does the following. The probability weights described by the Zipf are split into a fixed number of groups with the same number of files per group. Instead of generating requests that span the whole range of files in the coverage, the generator selects files from the first group first, for as many requests as the first group of files should receive. After generating all the requests for the first group, the generator moves to the second group and so on. Grouping files in this way forces files to be selected more closely together in time. For a number of groups larger than 1, the result is a workload with higher temporal correlation. For large numbers of groups, even unpopular files will have much higher temporal correlation. The resulting workload might be unrealistic for more than 1 group, but it does exhibit a controlled temporal correlation behavior, which suffices for our purpose of observing trends.

### 5.4.5 Real Traces

We also simulate two real (proxy server) traces, which we call Hummingbird (HUM) and Point of Presence (POP). HUM was collected at AT&T from 01/16/99

to 01/31/99. The total footprint of HUM consists of 76.9 GBytes with an average requested file size of 294 KBytes. POP was collected between 10/15/01 and 10/26/01 at the Federal University of Minas Gerais, Brazil. POP was filtered to eliminate the proxy misses, leading to a total footprint of 39.4 GBytes and an average requested file size of 17 KBytes.

## 5.5 Parameter Space Results

In this section we present the results of our parameter space exploration. We break the evaluation into two parts: arrays of conventional disks and arrays of two-speed disks. In each part, we start by analyzing workload characteristics and later move on to file server characteristics. To study the effect of each parameter, we assess the energy savings of the PDC, MAID, and FT-based versions of our file server (in comparison to the energy-oblivious version, EO), as a function of different settings for the parameter. To isolate the effect of a single parameter at a time, all other parameters are fixed at their default values. Throughout this section, we refer to the name of each technique and the server that exploits it interchangeably.

The default values for our simulation parameters are listed in table 5.5. Note that MAID-1+ is the MAID version of our file server that uses one extra disk as cache; MAID-2+ assumes two cache disks. The “+” refers to the optimization we made to MAID to avoid overloading the cache disk(s). The values for our simulated hardware are based on our own real server hardware, except for the main memory cache size. The values for the workload characteristics were chosen because they are “reasonable” values that represent areas of the parameter space

Description	Default Value
Request rate *	750 reqs/s (36 MBytes/sec)
File popularity *	0.85
Coverage *	40%
Percentage of writes *	0%
Temporal locality *	popularity-induced only
Trace length	19,000,000 requests
Main memory cache size *	1 GByte
Number of disks *	8 (PDC), 9 (MAID-1+), 10 (MAID-2+)
File size	48 KBytes
Total file system size	126 GBytes
Migration period *	every 1/2 hour (PDC only)
Cache disks	1 (MAID-1+), 2 (MAID-2+)
Disk characteristics	same as in tables 5.1 and 5.2

Table 5.5: Default parameters used in synthetic trace simulations: workload characteristics (top), server settings (middle), disk characteristics (bottom). We vary the parameters marked with “\*”.

where PDC and MAID perform well. In particular, the default request rate was set to half of the peak load we would expect for our real server. Common provisioning practice suggests that a server should be provisioned such that its peak load reaches 80% of its maximum throughput; the 20% extra capacity can handle any unexpected increases in load. We followed these suggestions and determined the maximum throughput of our server, 125 MBytes/sec, when all file requests can be served from the main memory cache. Given that the load peaks and valleys of servers deployed in the field can vary by factors ranging from 3 to 10 [23], our default request rate corresponds to a period of light load, exactly when energy savings are possible. Previous works have made similar assumptions, e.g., [21]. Nevertheless, we do vary the parameters marked with “\*” in the table in the next two subsections.

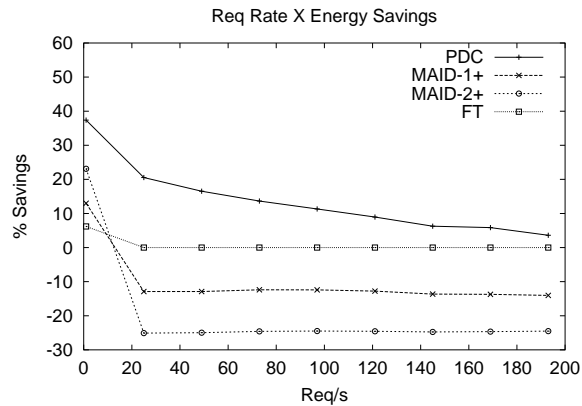


Figure 5.1: Energy savings per file request rate (conventional disks).

### 5.5.1 Arrays of Conventional Disks

Figure 5.1 depicts the energy savings that can be accrued by the four versions of our file server, as different average file request rates are imposed on the servers. The energy savings are computed in comparison to EO.

As expected, low request rates show better energy savings. The maximum savings are substantial, reaching almost 40% in the case of PDC. However, note that non-trivial energy savings can only be achieved for very low request rates, less than 100 requests/second. These request rates are less than 10% of our default request rate. A file request rate of 25 request/second, for instance, is equivalent to a throughput of only 1.2 MBytes/second. Even worse, when request rates are very low, a significant percentage of requests is delayed by disk spin up operations. Note also that the MAID systems actually produce negative energy gains, i.e. they consume more energy than EO, across most of the space of request rates. The reason is that the cache disks do not reduce the load on the non-cache disks enough for them to be spun down; they are simply energy overhead.

These request rate and response time results clearly show that conventional

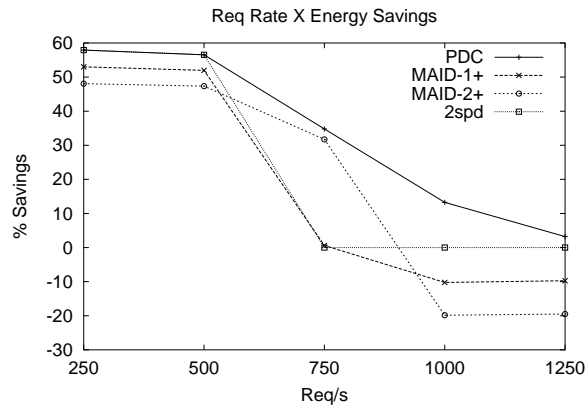


Figure 5.2: Energy savings per file request rate (two-speed disks).

disks are not useful when the goal is to conserve energy without noticeable performance degradation. When request rates are high, no energy savings can be accrued. When request rates are low, requests are frequently delayed by disk spin ups. These same negative effects are observed across the whole parameter space of workload and file server characteristics, so we do not present further results for arrays of conventional disks.

## 5.5.2 Arrays of Two-Speed Disks

### Workload Characteristics

**Request rate.** Figure 5.2 depicts the energy savings that can be accrued by the four versions of our file server, as a function of the file request rate. The curve labeled “2spd” represents the version that uses two-speed disks, but no data movement. All energy savings are computed with respect to EO running on an array of conventional (Cheetah) disks.

Again, lower request rates show better energy savings. In fact, the maximum savings in comparison to EO are very substantial, reaching almost 60%,

the energy differential between low and high speeds, for PDC and 2spd at 250 and 500 requests/second. In this part of the space, the MAID systems produce lower savings due to the cache disks, which are unnecessary. The energy savings decrease quickly as we increase the file request rate. At 750 requests/second, the 2spd and MAID-1+ systems stop producing energy gains, since all disks remain in high speed virtually all the time. At that request rate, PDC and MAID-2+ still produce gains of 30-40%, as data movement has the effect of reducing the load on several disks. At 1000 requests/second, the gains achievable by both MAID systems become negative, whereas PDC still produces gains of more than 10%. For higher request rates, the PDC gains are reduced and approach 3% at 1250 requests/second.

These results demonstrate that PDC is more consistent and robust than other techniques for the range of request rates that can actually provide energy savings. PDC achieves the highest gains across the range, without ever consuming more energy than EO. The behavior of the MAID systems is heavily influenced by the number of cache disks used. Moreover, the overhead of cache disks is pronounced on both ends of the range. 2spd does not provide gains in most cases.

In terms of response time, we find that all versions of our server exhibit less than 2% delayed requests, regardless of the request rate. In fact, the percentage of delayed requests is consistently very low for all except the most extreme (and clearly unrealistic) of parameters for two-speed disks. For this reason and the fact that network servers can accommodate substantial response time degradations, hereafter we do not discuss response times.

**File system coverage.** Figure 5.3 presents the energy savings accrued by the four servers, as a function of file system coverage. Recall that the default coverage

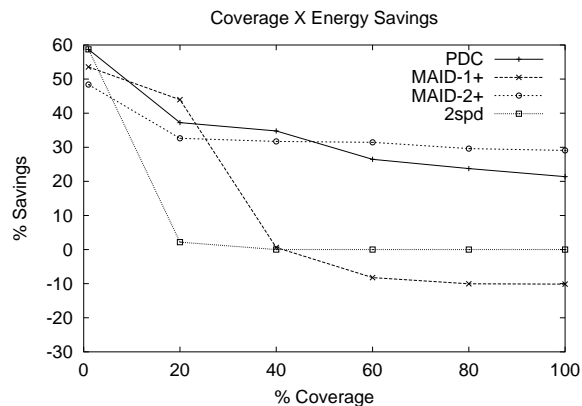


Figure 5.3: Energy savings per file system coverage.

we consider in our other analysis is 40%. As expected, higher coverages mean that disk loads increase and therefore the energy savings are reduced. On the other hand, higher coverage also allows the Nomad FS file placement algorithm (MQ) to place a larger number of files more intelligently in the array. As a result, the energy savings accrued by PDC degrade gracefully. MAID-2+ also degrades gracefully, since its two cache disks provide enough extra bandwidth at the default request rate to compensate for increases in coverage. In contrast, MAID-1+ suffers severely with increasing coverage, as a result of the extra pressure on the cache disk. When this disk becomes fully utilized, the non-cache disks also start operating at high speed.

**File popularity.** Figure 5.4 depicts the energy gains for different values of the Zipf coefficient ( $\alpha$ ). Alpha represents the file popularity as seen by the server, not the disk subsystem. Recall that the default popularity we assume in our other analysis is 0.85. The figure shows that all systems suffer as we decrease the popularity. The reason is that decreased popularity means lower main memory cache hit rates, which yield higher disk loads. Furthermore, the PDC file migration

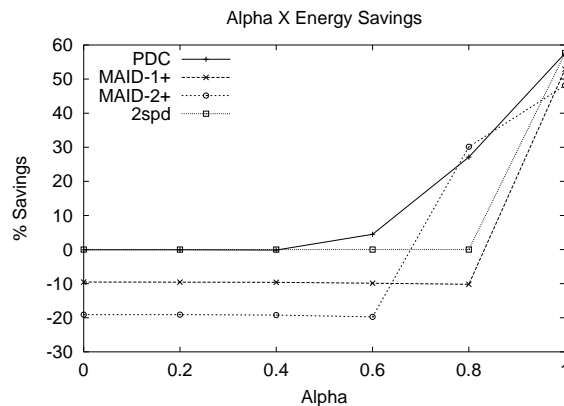


Figure 5.4: Energy savings per file popularity.

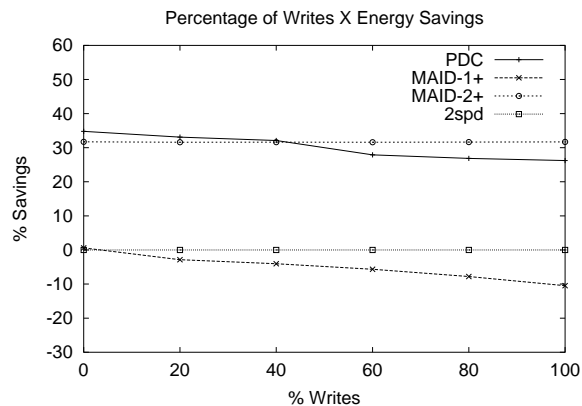


Figure 5.5: Energy savings per percentage of writes.

approach does not work as well when file popularity is decreased. Despite these problems, PDC degrades more gracefully than MAID or 2spd; it provides energy gains for popularities as low as 0.6. MAID does not behave as well as PDC because the files on its cache disks exhibit worse (popularity-induced) temporal locality with decreased popularity. This increases the load on the cache disks. When the cache disks become fully utilized, the traffic directed to the other disks increases further, forcing all disks to operate at high speed. As a result, the MAID systems consume more energy than EO for most of the parameter space.

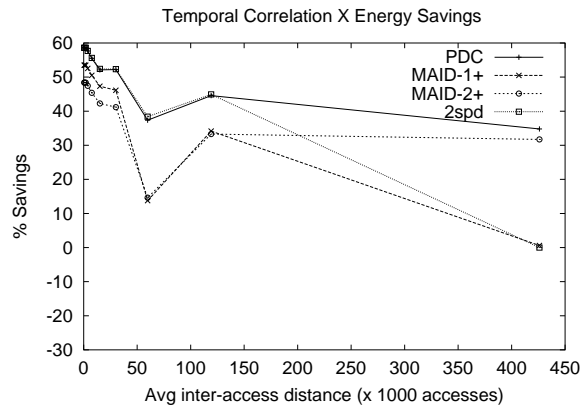


Figure 5.6: Energy savings per temporal correlation.

**Percentage of file write operations.** Figure 5.5 shows the effect of the percentage of writes on the energy savings provided by the servers we consider. Recall that we have been considering read-only workloads by default. The figure shows that the energy savings provided by the different techniques decrease slightly (if at all) with increases in the percentage of writes. The reason for the slight decreases is that more writes increase the disk traffic when dirty blocks are evicted from the main memory cache. In fact, in the MAID-based servers, writes can cause even more traffic when dirty disk cache blocks have to be written back to their corresponding non-cache disks.

**Temporal correlation.** Figure 5.6 shows the effect of temporal correlation on the energy savings of the servers we consider. We compute the average number of file accesses between two consecutive accesses to each file and compute the overall average for all files. Recall that we assume no temporal locality besides that induced by file popularity in our other analysis. For each curve in the figure, this means the rightmost point.

We can see that all systems benefit from higher temporal correlation (i.e.,

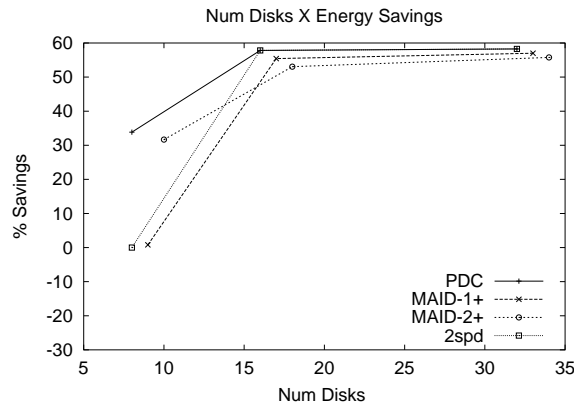


Figure 5.7: Energy savings per number of disks.

smaller average inter-access distance). The higher correlation increases the main memory cache hit rate and reduces the load on the disk subsystem. This effect is highly beneficial for 2spd. MAID has the added benefit that high temporal correlation causes the cache disks to work more effectively. However, the MAID systems have the energy overhead of the cache disks themselves. Again, PDC is the most consistent and robust system across the parameter space.

### File Server Characteristics

**Number of disks.** Figure 5.7 shows the energy savings of the different versions of our server, as a function of the number of disks. The total storage size is not changed, therefore simulations with more disks have fewer files per disk and do not require additional cache disks for the MAID systems. Recall that the default number of disks we assume in our other analysis is 8 for PDC and 2spd, 9 for MAID-1+, and 10 for MAID-2+.

We can see that increasing the number of disks increases savings quite significantly, as one would expect. The smaller set of accesses directed to each disk

allows all disks to operate at low speed. Note also that the energy overhead of cache disks is better amortized in larger array configurations. However, if we had assumed the number of cache disks to increase proportionally to the number of regular disks, we would have seen overheads of around 10% and 20% for the two MAID systems in these large configurations.

**Main memory cache size.** Figure 5.8 considers the effect of main memory cache size. Recall that the default size of the memory cache we assume in our other analysis is 1 GByte.

Cache size increases produce significant benefits for all servers. With caches of 5 GBytes or larger, both PDC and 2spd approach the maximum achievable energy savings of 60%, since the cache miss rate becomes so low that all disks start to operate at low speed. The MAID systems also benefit from larger main memory caches, but settle at lower energy savings with very large caches.

**Migration frequency.** Figure 5.9 shows the effect of changes in the migration period in Nomad FS. Recall that the default migration period we assume in our other analysis is every 1/2 hour. For very short migration periods, there may not be enough time to complete the migrations until the next round of migrations needs to start. In these cases, we simply stop the current migration process midway and start the next. Since MAID and 2spd do not involve migrations, we plot their energy gains as horizontal lines for comparison.

From the figure we can see that the frequency of migrations has a substantial impact on the gains achieved by PDC. With longer periods, PDC takes longer to adjust to access patterns. It is also interesting to note that short migration periods provide slightly higher energy savings than our default setting. The reason is that the energy cost of file placement changes is a small fraction of the energy

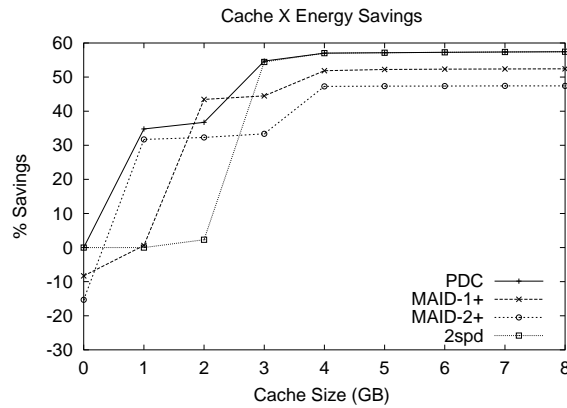


Figure 5.8: Energy savings per main memory cache size.

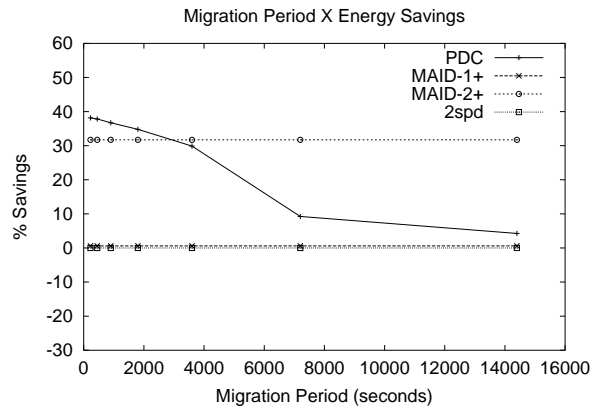


Figure 5.9: Energy savings per reconfiguration period.

consumed in between migrations. However, these shorter periods also come with substantial increases in the percentage of delayed requests, due to the intense file migration traffic.

## 5.6 Real Trace Results

The results so far have assumed synthetic traces and exponential request arrivals with a fixed mean arrival rate. In this section, we study real traces that exhibit pronounced load intensity variations, with load peaks in the afternoon and valleys

Description	HUM	POP
Avg. request rate	241 reqs/sec	263 reqs/sec
Coverage	100%	100%
File popularity	0.70	0.93
Percentage of writes	35%	0%
Temporal correlation	467280 reqs	94568 reqs
Cache size	1 GByte	64 MBytes
Migration period	672 minutes	168 minutes

Table 5.6: Parameters of real traces and server settings. Request rate values correspond to the accelerated traces. Migration periods correspond to one week in the non-accelerated traces.

Technique	HUM	POP
PDC Energy	1472438 J	147332 J
MAID-1+ Energy	1605547 J	170232 J
MAID-2+ Energy	1700585 J	181815 J
2spd Energy	1538228 J	160114 J
PDC Delayed	5.3%	8.0%
MAID-1+ Delayed	14.3%	8.5%
MAID-2+ Delayed	11.2%	8.1%
2spd Delayed	6.0%	7.6%

Table 5.7: Real trace results: energy and percentage of requests delayed.

at night. Note however that we accelerated the traces by factors of 15 (HUM) and 60 (POP), because even the peak loads in the original traces are relatively low for our disk array.

We simulate a tight coupling between the file server and the network server; the file server forms the “bottom half” of the network server on a single machine. We assume a “warm up” period corresponding to one week of the original, non-accelerated traces. Table 5.6 lists the characteristics of the traces and server settings with respect to the parameters we just explored. All simulations assume two-speed disks and 8 (PDC), 9 (MAID-1+), or 10 (MAID-2+) disks in the array.

Table 5.7 lists the energy and percentage of requests delayed (by more than

200 milliseconds) for each technique and trace after their warm up periods. EO consumes 2022095 J (HUM) and 256745 J (POP) with virtually no delayed requests.

These results confirm the main observations of the previous section, i.e. under light load, PDC conserves a significant amount of energy (27% for HUM and 43% for POP) while the cache disks in MAID reduce energy gains. The one exception is that 2spd conserves more energy than both MAID systems. 2spd performs well because, throughout most of the traces, the disk load is low enough that all disks can be at low speed. This characteristic makes these traces ideal for 2spd. Despite these good results, PDC still conserves more energy than 2spd (14% more for HUM and 13% more for POP) without an increase in the percentage of delayed requests.

Note also that the percentage of delayed requests is higher than in our parameter space exploration. The main reason for these results is that, going from each valley to the next peak, disk loads increase very quickly in the accelerated traces. Under this scenario, disk contention develops until the two-speed disks can transition to high speed. This should not be a problem in practice however, since load variations are substantially slower in real time.

## 5.7 Conclusions

We have introduced a new energy conservation technique called PDC for disk arrays. We have also designed and implemented an energy-aware file server called Nomad FS that exploits this technique. For comparison, we developed two versions of our server that use other energy conservation techniques, MAID and FT.

Using a validated simulator, we performed a comprehensive study of the parameters that affect energy conservation the most, pinpointing the scenarios under which each technique is useful, for disk arrays composed of either conventional or two-speed disks. The study allowed us to map the areas of the parameter space for which no technique can conserve energy. We also studied the behavior of the different techniques for two real traces.

In summary, we found that it is possible to conserve a substantial amount of energy during periods of light load on the server, as long as two-speed disks are used. We have also found that Nomad FS can deal more gracefully with increases in request rate and decreases in file popularity than the other servers. In addition, PDC achieved consistent energy savings for most of the parameter space. However, the PDC gains degrade substantially for long migration intervals. In comparison, the behavior of MAID is heavily dependent on the number of cache disks used. Furthermore, the energy overhead of these disks is pronounced in several parts of the parameter space. For the real traces, PDC achieved the best results, confirming the observations from the synthetic traces. We also showed that using two-speed disks without data movement behaves well when real disk loads remain light most of the time.

Based on these results, we conclude that the techniques we study will be very useful for disk array-based network servers when multi-speed disks become available.

Further research is required on the reliability implications of PDC. In fact, it is clear that PDC negatively affects the reliability of conventional disks due the large number of power mode transitions it causes. However, we have shown that PDC is not useful for conventional disks anyway. For multi-speed disks, we

expect PDC to have a negligible impact on reliability. The reason is that real network server workloads (which exhibit load peaks in the afternoon and valleys at night) should induce at most two speed transitions/disk/day in the presence of PDC.

Another interesting research issue is the interaction of PDC with storage system reliability approaches, such as mirroring and RAID. Extending Nomad FS to deal with mirroring should be easy, but combining PDC with data striping can be more challenging. We address both reliability and redundancy in PDC in chapter 6.

Finally, our parameter space exploration and experience with real traces suggest that PDC is highly sensitive to the length of the migration period and the estimation of the load to be imposed on disks after migrations. Another research topic is how to make Nomad FS automatically determine the best settings for these parameters.

## Chapter 6

### Diverted Accesses

In this chapter, we propose Diverted Accesses, a novel technique that exploits redundancy to conserve energy in storage servers. To fully understand and analyze the proposed technique, we develop mathematical models that estimate energy conservation for several previously-proposed algorithms. The models also estimate storage availability and performance, as a function of the its design and energy algorithm used.

Previously-proposed energy-conservation techniques are oblivious to redundancy and the effect of the energy techniques on service availability and performance. Redundancy is present in all practical storage systems, since it is mostly through redundancy that these systems achieve high reliability, availability, and throughput. Redundancy is typically implemented by replicating the “original data” (as in mirrored disk arrays, cluster-based storage systems [72], or wide-area storage systems [36, 89]) or by storing additional information that can be used to reconstruct the original data in case of disk failures (as in RAID-5 or erasure-code-based wide-area storage systems [43, 62]). We refer to the replicas and the additional information as “redundant data”.

Our approach is to leverage this redundancy to conserve disk energy. In particular, we propose a technique called *Diverted Accesses* that segregates the original and the redundant data on different disks. We refer to these disks as

original and redundant disks, respectively. The segregation allows the system to concentrate the requests on the original disks, leaving the redundant disks idle most of the time. During these idle periods, the disks can be sent to low-power mode to conserve energy. The redundant disks only need to be activated in three cases: (1) when the demand for disk bandwidth is high; (2) when one or more disks fail; and (3) periodically to reflect changes made to the original data. In this last case, the writes to the original disks need to be logged until the corresponding redundant disks are activated.

Because the benefits of our technique vary with the amount and type of redundancy built into the system, our evaluation analytically quantifies the effect of redundancy on several system characteristics, including disk energy consumption and the potential of different techniques to conserve energy. More specifically, we develop energy models for Diverted Accesses and previous redundancy-oblivious techniques, and couple them with well-known models of reliability, availability, and throughput. Our modeling results show that Diverted Accesses can provide substantial energy savings across a wide range of redundancy, request rate, and write percentage parameters. Other techniques are only useful in small parts of this parameter space.

Designers can use our models to determine the best redundancy configuration for new storage systems. Our approach is to select the configuration that achieves the required throughput, reliability, and availability but consumes the least amount of energy. Our results show that non-intuitive redundancy configurations are often the best ones when all metrics are considered.

Finally, to validate our analytical models and demonstrate the use of our technique in practice, we simulate a storage system based on PAST [89], a peer-to-peer

system with data replication, under two realistic access traces. Our simulation results show that a PAST system using Diverted Accesses can reduce disk energy consumption by 20-61%. These results are close to those predicted by our models.

We conclude that considering redundancy can provide significant disk energy savings beyond those of previous techniques. Furthermore, we conclude that designing a storage system requires quantifying several metrics, which are all affected by the redundancy configuration. Our models are key in this design process.

## 6.1 Background

Our work builds upon previous research on redundancy approaches, disk energy conservation, and storage system design. Next, we discuss each of these topics in turn.

### 6.1.1 Redundancy

Redundancy is typically implemented in storage systems through replication, parity schemes, or erasure codes. These methods can be defined in terms of their redundancy configurations by  $(n, m)$  tuples, where each block of data is striped, replicated, or encoded into  $n$  fragments, but only  $m$  fragments ( $m \leq n$ ) are needed to reconstruct the data. For instance, a RAID 1 storage system is represented by  $(n = 2, m = 1)$ , since there are two copies of each block of data but only one copy is enough to reconstruct the block.

Several papers have considered the properties of these different redundancy approaches, e.g. [7, 15, 103]. Replication typically requires more bandwidth

and storage space than the parity schemes. However, parity schemes can only tolerate small numbers of concurrent failures; they do work well for small to moderately sized disk arrays. Erasure codes require less bandwidth and storage than replication (for the same levels of reliability and availability), can tolerate more failures than parity schemes, but involve coding and decoding overheads.

**Contributions.** Our work complements these previous studies as we consider the impact of redundancy configuration on disk energy consumption and disk energy conservation techniques.

### 6.1.2 Disk Energy Conservation

Several techniques have been proposed for disk energy conservation in storage systems. We compare Diverted Accesses to several of these techniques, explained previously in chapter 3. To recap, we divided the techniques into **threshold-based** and **data-movement** techniques. The threshold-based ones we study in this chapter are Fixed Threshold (FT) and Oracle. The data-movement techniques we study are Massive Array of Idle Disks (MAID) and Popular Data Concentration (see chapter 5). When comparing these techniques to Diverted Accesses, we study both FT and Oracle as their primitive power-management technique.

**Contributions.** Our work introduces Diverted Accesses, the first disk energy conservation technique to explicitly leverage redundancy. Further, our work presents energy models for FT, Oracle, MAID, PDC, and Diverted Accesses that take redundancy configurations into account, and a case study of the application of Diverted Accesses in the context of the PAST storage system. The energy models build on our previous experience with accurately measuring and modeling

energy conservation techniques for disk drives [21, 51, 53, 52] and disk arrays [85].

### 6.1.3 Storage System Design

The literature on system design and resource allocation approaches based on utility functions and behavior models is extensive. Here, we focus solely on approaches that have targeted storage systems or that have considered energy as well as more traditional metrics.

Anderson *et al.* [3] proposed Ergastulum, a tool that quickly evaluates the space of possible data layouts and storage system configurations and finds a near-optimal design. Ergastulum takes a workload description, user-specified constraints, and an optimization goal as inputs. The goal is typically to minimize the cost of the storage system in terms of the price of hardware. Ergastulum uses performance models to determine whether a potential design is acceptable. Minerva [1] is similar but not as efficient as Ergastulum. Hippodrome [2] uses Ergastulum to adjust the design of the storage system as a result of dynamic changes in the workload.

Chase *et al.* [23], as discussed previously, have proposed a market-based resource allocation framework that dynamically tries to maximize the profits generated by shared hosting centers. Their framework includes the revenues created by dedicating resources to different services, the penalties for violating service-level agreements, and the cost (e.g., electricity) of the resource allocations.

Heath *et al.* [49, 50] have proposed a system that dynamically reconfigures heterogeneous server clusters to conserve energy. Their system makes decisions based on a description of the expected workload, models of throughput and power consumption, and the offered load at each point in time. Their goal is typically

to consume the least energy per request without degrading performance.

**Contributions.** We extend the previous work on Ergastulum, Minerva, and Hippodrome by considering the reliability, availability, and energy consumption of different designs. Since we only focus on selecting the redundancy configuration  $(n, m)$ , our space of possible designs is much more constrained than in these systems. For our purposes, an exhaustive search algorithm works fine.

We extend the previous energy work by considering storage systems, a wider range of behavior models, and different techniques for conserving disk energy.

## 6.2 Diverted Accesses

Our approach to reducing energy consumption in storage systems leverages their redundancy. The key observation is that the redundant data is only read under two scenarios: (1) during periods of high demand for disk bandwidth, to increase performance; and (2) when disk failures occur, to guarantee reliability and availability.

Given this observation, it is clear that the redundant data need not be readily accessible at all times. For example, storage systems used for data backup exhibit long periods of low read loads during the day in between periods of intense write activity at night. As another example, regular file system activity in engineering environments is high during the day and low at night. Regardless of the type of storage system, if one or more disks fail, the only redundant data needed is that corresponding to the failed disks.

Unfortunately, current storage systems cannot take advantage of these characteristics to conserve disk energy. Our Diverted Accesses technique addresses

this limitation by segregating the original data from the redundant data onto different subsets of disks: original and redundant disks, respectively. Even when all fragments contain redundant information, i.e. no fragment is strictly “original”, we can still store  $m$  fragments of each block on a subset of the disks, which would act as the original disks.

The goal is to keep redundant disks in low-power mode during periods of light and moderate offered disk load. We define the load as high when the set of original disks is not enough to provide the required disk bandwidth. More specifically, the handling of reads and writes depends on the offered load as follows:

- Reads are directed to the original disks only, unless the offered load is high enough that redundant disks need to be activated to help service it.
- Writes are performed on both the original and redundant data, when the load is high. When the load is light or moderate, writes are directed to the original disks and only propagated to the redundant disks periodically.

The handling of writes under light and moderate loads deserves further discussion, as it involves an interesting tradeoff between reliability and energy conservation. Updating the redundant data on all writes would promote reliability but prevent energy conservation for workloads with a non-trivial fraction of write accesses. Buffering these writes in memory for long periods would promote energy conservation but harm reliability. Our solution is to buffer writes to redundant data long enough to prevent frequent power-mode transitions but not any longer; only the data written during these short periods has a lower reliability. For systems that cannot accept this short window of lower reliability for a fraction of

their data, we can buffer these writes in non-volatile memory in single-node storage systems or multiple memories in distributed storage systems. In fact, in a distributed storage system with multi-disk nodes, we could also buffer writes on one of the disks at each node. Section 6.5 addresses this tradeoff in more detail.

Despite this buffering, Diverted Accesses does pose a problem for parity-based storage systems in the presence of “small writes”. For example, applying Diverted Accesses to RAID 5 would transform it into RAID 4. The bandwidth requirements of the parity disk in RAID 4 may degrade performance under small writes. Thus, the designer needs to understand the characteristics of the workload before applying Diverted Accesses in parity-based systems.

Finally, note that we focus solely on disk energy conservation in this paper. However, Diverted Accesses can be applied to entire servers in a distributed storage system. In this case, we could accrue energy savings by sending memories, processors, or even the entire nodes to low-power mode as well. Considering other components would magnify our potential savings, so we plan to address this issue in our future research.

### **6.3 Designing Real Systems**

In this section, we present well-known models for reliability, availability, and performance. Using these models along with our energy models, we can select the best redundancy configuration for new storage systems.

### 6.3.1 Reliability and Availability

In terms of reliability, we assume that disks lose or damage the data they store (e.g., due to a permanent mechanical drive failure) independently with probability  $1 - r$ , where  $r$  is the reliability of each disk. In terms of availability, we assume that disks become temporarily unavailable (e.g., due to a loose SCSI cable or a period of offline maintenance) independently with probability  $1 - a$ , where  $a$  is the availability of each disk.

Given these assumptions, the models that quantify reliability and availability as a function of the redundancy configuration  $(n, m)$  are very similar [94]. Equation 6.1 is the availability model when at least  $m$  fragments must be available upon a disk access. The reliability model is the same, except that  $a$  is replaced by  $r$ .

$$A = \sum_{i=0}^{n-m} \binom{n}{i} a^{n-i} (1-a)^i \quad (6.1)$$

The value of  $A$  is typically close to unity in highly available systems. For this reason, availability is often referred to in terms of the number of nines after the decimal period. For example,  $A = 0.999$  means three nines availability or about 9 hours of unavailability per year. Because disk reliability is substantially higher (by multiple nines) than disk availability, we do not consider reliability further in this paper.

### 6.3.2 Performance: Throughput

Our performance model is the aggregate maximum throughput of the disks in the system, given a fixed configuration. Recall that the number of disks is a function of the redundancy configuration,  $N = Dn/m$ .

The maximum throughput is defined by the redundancy configuration and the total disk bandwidth for the workload at hand:

$$\begin{aligned}
 fragSize &= blockSize/m \\
 delay &= S + R + fragSize/X \\
 width &= fragSize/delay \\
 width_r &= N \times width \\
 width_w &= N \times width(m/n) \\
 P = totalBW &= p_w width_w + (1.0 - p_w) width_r
 \end{aligned} \tag{6.2}$$

where *blockSize* is the maximum between the block size exported by the storage system interface and the weighted mean of a distribution of request sizes (each size being a multiple of the block size); *fragSize* is the size of each fragment; *width<sub>r</sub>* and *width<sub>w</sub>* represent the *effective* bandwidth for reads and writes, respectively; as previously defined, *S*, *R*, and *X* are the average seek time, the average rotational delay, and the disk transfer rate, respectively; and *p<sub>w</sub>* is the probability of writes. Note that the definition of *fragSize* may have to be changed for storage systems that exhibit fixed fragment size, e.g. in RAID 5,  $fragSize = stripeSize$ .

These equations apply to all energy conservation techniques, even though disks that are in low-power mode limit the maximum throughput of the system. However, all techniques activate all disks when the offered load requires it.

### 6.3.3 Putting It All Together

Determining the best redundancy configuration for a storage system involves meeting its storage capacity, availability, and throughput requirements for the least amount of energy. More specifically, we need to explore the space of potential configurations  $(n, m)$  and energy conservation techniques, trying to minimize energy (or average power), subject to the constraints on storage capacity, availability, and throughput. Given the relatively small search space of  $n * m$  possible configurations, for example in  $n \in [1..16]$  and  $m \in [1..16]$ , the problem becomes easy to solve by enumeration (and modeling, of course).

## 6.4 Modeling Energy

We model block-based storage systems comprised of identical disks. These disks may be attached to one or more servers, but our models are independent of such configuration issues. Besides the main characteristics of the disks, the main inputs to our models are the redundancy configuration  $(n, m)$  and the percentage of read and write accesses in the workload. We demonstrate that the models are very accurate in Section 5.5.

### 6.4.1 Overview

We define  $D$  as the number of disks required to store the data without any redundancy. Given a redundancy configuration, we define  $N$ , the number of required disks to store all data (original plus redundant),  $N(n, m) = (n/m)D$ . For simplicity, we refer to  $N$  instead of  $N(n, m)$ .

Each disk has a power consumption of  $P_h$  Watts when powered on and ready

to service requests (high power mode) and  $P_l$  when in standby mode, not able to service requests (low power mode). (Note that most high-performance, server-class disks – typically SCSI drives – do not offer more than one power-saving mode.) A disk spin up takes time  $T_u$  and energy  $E_u$ , whereas a disk spin down takes time  $T_d$  and energy  $E_d$ . A full transition from high-power mode to low-power mode and back to high consumes  $T_t = T_d + T_u$  time and  $E_t = E_d + E_u$  energy.

Each request to the storage system is assumed to have size  $blockSize$ . Internally, disk data is accessed in fragments of size  $fragSize$ , which is defined as  $blockSize/m$ . On each access, the disks take time  $S$  to seek to the appropriate track and time  $R$  to rotate to the desired sector. A fragment of data is transferred at a nominal rate  $X$ . We do not model the energy consumed by disk accesses, since previous works have demonstrated that it is a small fraction of the overall disk energy, even in busy systems [45, 46].

As in previous works [45], we only model the request traffic that reaches the storage system (i.e., beyond any main memory caches) and assume that request inter-arrival times are drawn from a Pareto distribution with average  $1/requestRate$ , even though our models work with any distribution. Block requests can be reads or writes with probabilities  $1 - p_w$  and  $p_w$ , respectively.

To estimate energy (average power) we do the following:

1. Draw a request inter-arrival time  $t$ ;
2. For each energy conservation technique, calculate the average idle time per disk, based on the inter-arrival time;
3. For each energy conservation technique, estimate the average power (as a

proxy for energy) for this idle time based on the underlying power management mechanism (FT or Oracle) and any other technique-specific parameters;

4. Repeat three previous steps until enough samples have been drawn to allow the average inter-arrival time to converge to  $1/requestRate$ ;
5. Compute the overall average power of each technique by dividing the sum of the average powers by the number of inter-arrival times drawn.

Note that each inter-arrival time  $t$  we draw in step 1 above does not represent a single disk access. Rather, it represents a steady-state period in which requests arrive uniformly with interval  $t$  and access disks in round-robin fashion. Our goal here is to model workloads as comprised by phases during which a large number of clients produce request rates at the storage system that are approximately constant. (Without these assumptions, it would have been difficult if not impossible to derive closed forms for our models.) The validation results of Section 6.6 show that our models are accurate, despite these assumptions.

Table 6.1 summarizes the model parameters and their meanings for easy reference.

## 6.4.2 Energy Conservation Techniques

We model six different energy conservation techniques: Fixed Threshold (FT), Oracle adaptive (Oracle), Popular Data Concentration (PDC), Massive Array of Inexpensive Disks (MAID), Diverted Accesses (DIV), and a technique combining PDC and DIV (PDC+DIV). We compute the energy savings of each technique

Symbol	Description
$D$	Number of required disks without redundancy
$N$	Number of required disks with redundancy
$P_h$	Disk power in high-power mode
$P_l$	Disk power in low-power mode
$T_u$	Time to transition to high-power mode
$T_d$	Time to transition to low-power mode
$T_t$	Time to transition down and up again
$E_t$	Energy to transition down and up again
$blockSize$	Size of blocks
$fragSize$	Size of fragments
$S$	Average disk seek time
$R$	Average disk rotation time
$X$	Disk transfer rate
$requestRate$	Arrival rate of block requests
$p_w$	Probability of block write requests
$D_{maid}$	Number of cache disks (MAID)
$m_{maid}$	Miss rate of cache disks (MAID)
$\beta$	Disk popularity coefficient (PDC)
$c$	Storage system coverage (PDC)
$wbSize$	Size of write buffer (DIV)
$batchSize$	Size of write batches (MAID+DIV)

Table 6.1: Summary of model parameters and their meanings.

with respect to an energy-oblivious (EO) system that keeps all disks at  $P_h$  at all times.

**FT.** In FT, disks are transitioned to low-power mode after an idleness threshold  $T$ . We set  $T$  to the break-even time, i.e.  $T = E_t/(P_h - P_l)$ . For each inter-arrival time  $t$ , we can approximate the idle time per disk  $I$  as:

$$\frac{Nt}{np_w + m(1 - p_w)} \quad (6.3)$$

Since the average number of disk accesses per second is  $(np_w + m(1 - p_w))/t$ , the idle time at each disk is simply the inverse of the access rate times the number of disks.

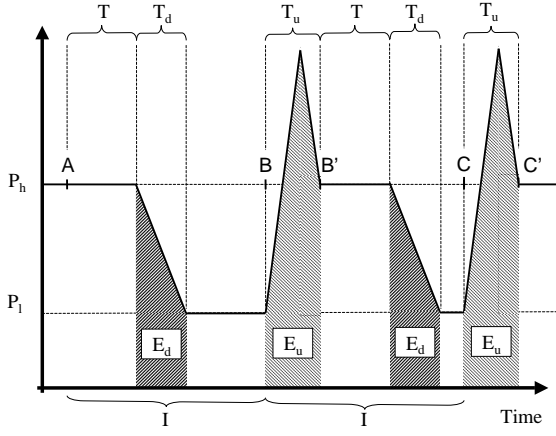


Figure 6.1: Second scenario:  $I \geq T + T_u$ . Accesses arrive at times A, B, and C. Accesses are actually performed at times A, B', and C'.

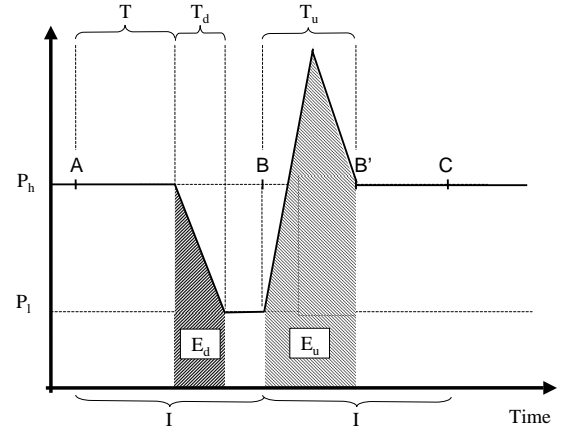


Figure 6.2: Third scenario:  $T \leq I < T + T_u$ . Accesses arrive at times A, B, and C. Accesses are actually performed at times A, B', and C.

The average power for each idle time is then defined by the three cases below:

$$\begin{aligned}
 &P_h N, && I < T \\
 &(T P_h + (I - T - T_t) P_l + E_t) N / I, && I \geq T + T_u \\
 &((I - T_u + T) P_h + (I - T - T_d) P_l + E_t) N / (2I), && \text{otherwise}
 \end{aligned} \tag{6.4}$$

The *otherwise* case above represents all other possibilities not included previously, i.e.  $T \leq I < T + T_u$ . To avoid more cases, our modeling extends idle times that end during a transition to low-power mode until the end of the transition.

More intuitively, the top equation represents the scenario in which idle times are too short and do not trigger a transition to low-power mode. The energy is simply that of all disks on, so the average power will be  $P_h N$ . The middle equation represents the scenario in which there is enough time for the disks to transition to low-power mode, perhaps spend some time in low-power mode, and transition back. In this case, all idle times except the first are effectively reduced by the spin up time. Figure 6.1 illustrates this scenario. The figure shows accesses arriving at times A, B, and C; actual disk accesses occur at times A, B', and C'. Note that, after the first idle time, the behavior between B and C will consistently be repeated while this idle time is in effect. Thus, we use the period between B and C in computing the average power. The last equation represents the scenario

in which there is not enough time for the full transition to and from low-power mode. In this case, the next idle time is shortened to a point that no power-mode transitions can happen. Figure 6.2 illustrates this scenario. The behavior between A and C will be consistently repeated while this idle time is in effect. For this reason, we use the entire period (two idle times) in computing the average power.

**Oracle.** In the Oracle technique, there is no need to wait for  $T$  before going to low-power mode; Oracle simply spins the disk down when the next idle time is longer than  $T$ . The disk can be spun back up so that it will be ready exactly when the next request arrives. Idle times are computed exactly like in FT. In terms of energy, only two cases apply: when the idle time will payoff in terms of energy savings ( $I \geq T$ ) or when it will not ( $I < T$ ). Therefore, the average power consumption of Oracle for each idle time is given by:

$$\begin{aligned} P_h N, & \quad I < T \\ ((I - T_t)P_l + E_t)N/I, & \quad I \geq T \end{aligned} \tag{6.5}$$

**MAID.** In this strategy,  $D_{maid}$  extra disks are used to cache recently-accessed files. Upon each block request, if the block is not yet on one of the cache disks, the corresponding fragments are accessed at the non-cache disks and also copied to one of the cache disks. To avoid high latencies, requests may bypass the cache disks during periods of high load. Thus, the maximum throughput of a MAID system is that of  $N + D_{maid}$  disks. In terms of energy, this high load scenario leads to extremely short idle times and an average power of  $(N + D_{maid})P_h$ . Below, we model MAID under light and moderate loads.

Modeling MAID requires knowledge of the temporal locality of accesses to files. As an approximation, we assume that we know the fragment cache miss

ratio  $m_{maid}$  of the cache disks. We can then estimate the idle times of the cache disks and the non-cache disks. To conserve energy, we can leverage the cache disks to accumulate the writes to a non-cache disk until a cache disk miss (a read) accesses it. At that point, the accumulated writes can be performed on the non-cache disk. This approach promotes energy conservation at the cost of lower reliability.

We need to calculate two distinct idle times: for the MAID caches ( $I_{cache}$ ) and for the non-cache disks ( $I_N$ ).

$$\begin{aligned} I_{cache} &= (D_{maid}t)/((np_w) + (m(1 - p_w))) \\ I_N &= (Nt)/(m(1 - p_w)m_{maid}) \end{aligned} \tag{6.6}$$

The average power consumed by the cache disks can be computed as in FT (equations 6.4), except that we need to replace  $I_{cache}$  for  $I$  and  $D_{maid}$  for  $N$ . The average power of the non-cache disks can also be computed using those equations, as long as we replace  $I_N$  for  $I$ . The overall average power is the sum of the cache and non-cache powers.

Note that our modeling of MAID is simplistic. In the extreme cases in which there are no read accesses ( $p_w = 1$ ) or all read accesses hit the cache disks ( $m_{maid} = 0$ ), the idle time of the non-cache disks is modeled as infinite. In other words, we assume the cache disks to have infinite write buffering capacity. Further, we assume that the energy spent in copying data to the cache disks is negligible. Our goal with these simplifications is to provide an upper bound on the energy conservation potential of MAID.

**PDC.** Like FT, Oracle, and MAID, the original PDC proposal [85] did not consider redundancy explicitly. However, unlike the other techniques, PDC can hurt

reliability significantly if it is applied to all fragments arbitrarily.

To avoid this problem, we modify PDC to migrate data in such a way that the  $n$  fragments of each block remain on different disks. Over time, the most popular fragments would then be stored on the “first” set of  $n$  disks, the second most popular fragments would be stored on the “second” set of  $n$  disks and so on.

In order to model PDC, we need to introduce the notions of disk popularity and file system coverage. Previous research has shown that the popularity distribution of various Web and network workloads follows a power law [12]. In particular, Zipf’s power law with coefficient  $\alpha$  to describe the popularity of files. Zipf’s law states that the probability of a file being accessed is proportional to  $1/r^\alpha$ , where  $r$  is the rank (popularity) of the file and  $\alpha$  is the degree of skewness of popularity. For example, when  $\alpha = 0$ , all files are equally likely to be accessed. The larger  $\alpha$  is, the more heavy-tailed the distribution is. Based on a similar idea, we use Zipf’s power law with coefficient  $\beta$  to describe the popularity of the groups of  $n$  disks in steady state, i.e. when all fragments have been migrated to their best locations.

File system coverage represents the percentage of blocks that are actually accessed in a given period of time (a day, a week, or the length of a workload).

To compute the idle times, we need to take the disk popularity and the coverage  $c$  into account. We do so using an “idleness weight”  $w$  for each set  $i$  of  $n$  disks:

$$w_i = (Nc/n) \left( 1 - \frac{1/i^\beta}{\sum_{j=1}^{\lceil Nc/n \rceil} 1/j^\beta} \right) \quad (6.7)$$

The idleness weight of a group of disks ranges from 0 to  $Nc/n$  and is proportional to the fraction of accesses that is not directed to the group. In other words, the most popular group will have a weight that tends to 0, whereas the least popular group will have a weight that approaches  $Nc/n$ .

These weighting factors can be used to weight the idle time  $I$  from equation 6.3 in computing the average power consumed by the disks for each idle time in PDC:

$$\sum_{i=1}^{\lceil Nc/n \rceil} \begin{cases} w_i P_h n, & Iw_i < T \\ (TP_h + (Iw_i - T - T_t)P_l + E_t)n/I, & Iw_i \geq T + T_u \\ ((Iw_i - T_u + T)P_h + (Iw_i - T - T_d)P_l + E_t)n/(2I), & \textit{otherwise} \end{cases} \quad (6.8)$$

Our modeling of PDC is optimistic for three reasons. First, our coverage parameter assumes that all write accesses are updates of existing data, rather than writes of new data. Second, we assume the energy consumed in data migration to be negligible. Third, due to the complexity of its data layout, PDC is essentially impractical in the presence of redundancy; it would be very hard to implement in single-node storage systems, and even harder in distributed storage systems. Nevertheless, our goal is to compute an upper bound on the potential benefits of PDC.

**DIV.** Diverted Accesses stores the original data on  $D$  disks and the redundant data on  $R = N - D$  disks. The redundant disks can be sent to low-power mode, until they are required for reliability or to provide higher bandwidth under high load. Again, idle times are short under high load, leading to an average power consumption of  $NP_h$ . Next, we model DIV under light and moderate loads.

First, we compute the idle times on the original disks,  $I_D$ :

$$I_D = \frac{Dt}{m} \quad (6.9)$$

Note that all (read and write) requests translate into accesses to the  $D$  disks. Writes are also buffered, so the expected idle time on the redundant disks ( $I_R$ ) is the expected time for the write buffer to fill up times  $R$ .

$$I_R = \frac{Rt * wbSize}{blockSize * (n - m)p_w}, \quad (6.10)$$

such that  $wbSize \geq blockSize$ .

With these idle times, the average power for DIV can be computed as the sum of the power consumed by the original and the redundant disks. These average powers can be computed the same way as in FT (equations 6.4) with minor differences. For the original disks, the average power is:

$$\begin{aligned} P_h D, & \quad I_D < T \\ (TP_h + (I_D - T - T_t)P_l + E_t)D/I_D, & \quad I_D \geq T + T_u \\ ((I_D - T_u + T)P_h + (I_D - T - T_d)P_l + E_t)D/(2I_D), & \quad otherwise \end{aligned} \quad (6.11)$$

For the redundant disks, the average power is:

$$\begin{aligned} P_h R, & \quad I_R < T \\ (TP_h + (I_R - T - T_t)P_l + E_t)R/I_R, & \quad I_R \geq T + T_u \\ ((I_R - T_u + T)P_h + (I_R - T - T_d)P_l + E_t)R/(2I_R), & \quad otherwise \end{aligned} \quad (6.12)$$

**MAID+DIV.** MAID can be combined with DIV. In MAID+DIV, the idea is to place a few cache disks in front of DIV-structured disks.  $D_{maid}$  extra disks are

used to cache recently accessed blocks like in MAID, whereas the  $D + R = N$  non-cache disks are organized as in DIV.

Given the extra cache disks, we can accumulate writes aggressively on them, as in MAID. The writes are propagated to the non-cache disks on read misses on the cache disks (original disks) or periodically in a large batch (redundant disks). The resulting idle times,  $I_{cache}$ ,  $I_D$ , and  $I_R$ , for MAID+DIV are calculated as follows:

$$\begin{aligned}
 I_{cache} &= (D_{maid}t)/((np_w) + (m(1 - p_w))) \\
 I_D &= (Dt)/(m(1 - p_w)m_{maid}) \\
 I_R &= (Rt * batchSize)/((n - m)p_w * blockSize),
 \end{aligned}
 \tag{6.13}$$

such that  $batchSize \geq blockSize$ .

The average power consumed by the cache disks in MAID+DIV is computed as in MAID, whereas the average power of the non-cache disks is computed as in DIV. The overall average power is the sum of these components, as the energy used by data copying is assumed negligible.

**PDC+DIV.** Here, we combine PDC with DIV. The idea is to segregate original and redundant fragments and only migrate the original ones according to popularity. Migration is performed in such a way that the  $m$  fragments remain on different disks. Over time, the most popular fragments would then be stored on the “first” set of  $m$  original disks, the second most popular fragments would be stored on the “second” set of  $m$  original disks and so on.

Due to the concentration of accesses, the computation of the idle times uses idleness weights  $w$  (just as our modeling of PDC) for each group  $i$  of  $m$  original disks:

$$w_i = (Dc/m) \left( 1 - \frac{1/i^\beta}{\sum_{j=1}^{\lceil Dc/m \rceil} 1/j^\beta} \right) \quad (6.14)$$

These weighting factors can be used to weight the idle time  $I_D$  from equation 6.9 in computing the average power consumed by the original disks for each idle time in PDC+DIV, according to the cases below.

$$\sum_{i=1}^{\lceil Dc/m \rceil} \begin{cases} w_i P_h m, & I_D w_i < T \\ (T P_h + (I_D w_i - T - T_t) P_l + E_t) m / I_D, & I_D w_i \geq T + T_u \\ ((I_D w_i - T_u + T) P_h + (I_D w_i - T - T_d) P_l + E_t) m / (2 I_D), & \textit{otherwise} \end{cases} \quad (6.15)$$

The average power consumed by the redundant disks can be computed exactly as in DIV. The overall average power is the sum of these two original and redundant powers. Again, we assume all writes to be updates to existing data and the energy of data migration to be negligible.

## 6.5 Modeling Results

In this section, we present our modeling results for a wide range of parameters. Specifically, we analyze the tradeoffs between different  $(n, m)$  configurations, in terms of energy, availability, and performance. Because the parameter space has at least 5 dimensions –  $n$ ,  $m$ , energy, availability, and performance –, it is impossible to visualize it all at the same time. Thus, we plot 2-D graphs showing the interesting parts of the space.

We computed the energy results in this section using a synthetic workload

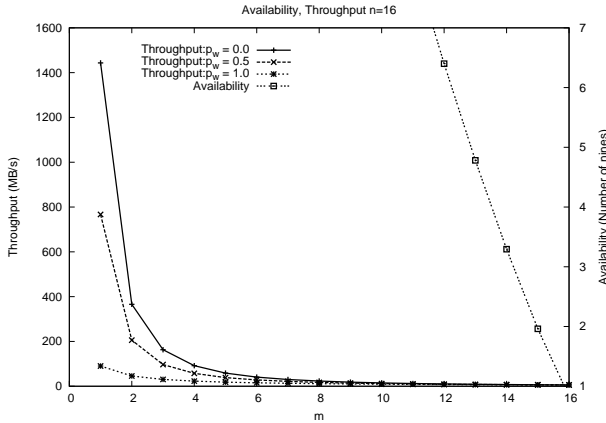


Figure 6.3: Availability and throughput for  $n = 16$ .

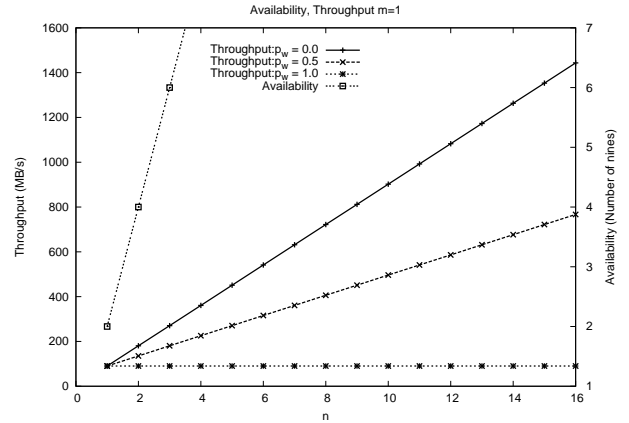


Figure 6.4: Availability and throughput for  $m = 1$ .

generated as follows. We draw 10,000 inter-arrival times from a Pareto distribution with a default average of 64 requests per second and an infinite variance. Requests are for 8-KB blocks. The disk parameters are based on the IBM 36Z15 Ultrastar model. Table 6.2 summarizes the default parameter values we used.

Although we study a wide range of parameter values, we carefully selected the default values for the workload-related parameters. In particular, the defaults for *requestRate* and  $p_w$  lie in the middle of the range created by our two realistic access traces (see Section 6.6). Further, we selected the default for  $D_{maid}$  based on simulations of the cache disk miss rate under our traces; 0.1*N* cache disks leads to the best tradeoff between miss rate and number of disks for our traces. The default value for  $m_{maid}$  lies in the middle of the range created by our two traces for the default number of cache disks. Our traces do not include information about coverage, so we arbitrarily chose 70% as its default value but quantify the effect of changes in this parameter explicitly. Finally, we do not have definitive information about  $\beta$  either; we set it to 1.0, but have found through extensive tests that this parameter has a negligible impact on the energy gains achieved by PDC and PDC+DIV.

Parameter	Default Value
Request Rate ( <i>requestRate</i> )	64 reqs/sec
Write Ratio ( $p_w$ )	33%
Disk Popularity ( $\beta$ , PDC)	1.0
File System Coverage ( $c$ , PDC)	70%
# MAID Cache Disks ( $D_{maid}$ )	$0.1N$
MAID Fragment Miss Ratio ( $m_{maid}$ )	40%
MAID+DIV Batch Size ( <i>batchSize</i> )	1 GB
DIV Write Buffer Size ( <i>wbSize</i> )	4 MB
Block size ( <i>blockSize</i> )	8 KB
# Disks without Redundancy ( $D$ )	64
# Disks with Redundancy ( $N$ )	Varies
Disk availability ( $a$ )	0.99
High Power ( $P_h$ )	10.2 W
Low Power ( $P_l$ )	2.5 W
Avg. Seek Time ( $S$ )	3.4 ms
Avg. Rot. Time ( $R$ )	2.0 ms
Transfer Rate ( $X$ )	55.0 MB/sec
Idleness Threshold ( $T$ )	19.2 secs
Spin up Time ( $T_u$ )	10.9 secs
Spin down Time ( $T_d$ )	1.5 secs
Energy transition down+up ( $E_t$ )	148.0 J

Table 6.2: Configurable parameters and their default values.

### 6.5.1 Availability and Throughput

Figures 6.3 and 6.4 show the modeled maximum throughput (Y-axis on the left) and availability (Y-axis on the right) of the system for different redundancy configurations. As throughput is dependent on the probability of writes, the figures show results for three different workloads. Availability is presented as the number of nines; we cut the curves at 7 nines, which means 3 seconds of down time per year. Figure 6.3 varies  $m$  for  $n = 16$ , whereas figure 6.4 varies  $n$  for  $m = 1$ . These figures assume that the number of disks  $N$  increases (decreases) as the amount of redundancy –  $n/m$  – in the system increases (decreases). When  $n = m$ ,  $N = D = 64$ .

Let us discuss availability first. Availability gets asymptotically close to 1 as

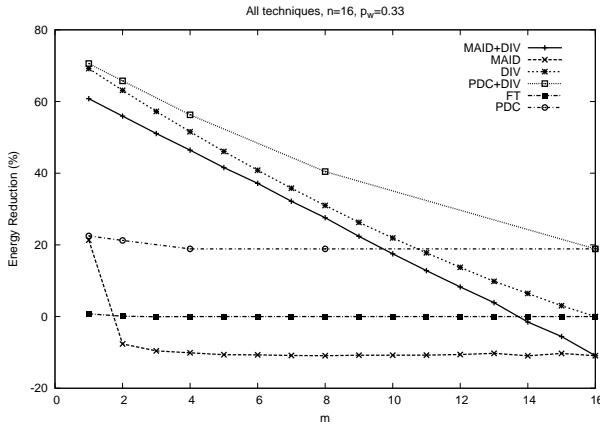
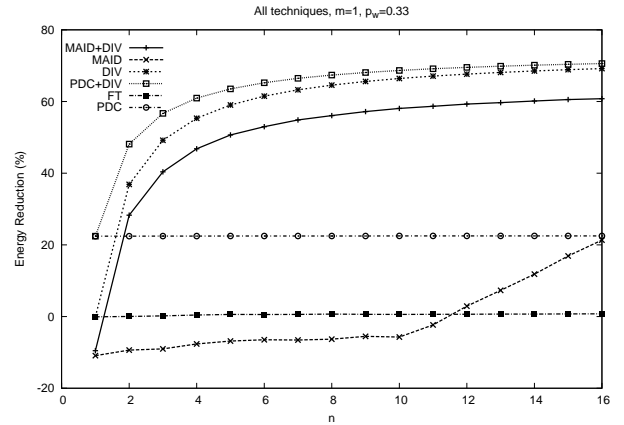
$m$  decreases and  $n$  increases. At  $n = 16, m = 1$ , the system availability is 16 nines. Clearly, this level of availability is overkill for most systems; 5 or 6 nines is a more reasonable goal (equivalent to the public telephone system). The figures show that the system can achieve this level of availability with  $n = 16, m = 12$  (figure 6.3) or  $n = 3, m = 1$  (figure 6.4).

In terms of performance, the throughput of each configuration is only limited by the number of disks used. In figure 6.3, the throughput drops approximately by  $1/m^2$  (compared to the throughput at  $m = 1$ ) with larger  $m$ . This happens for two reasons: (1) larger  $m$  implies smaller fragments, which reduce the throughput almost linearly; and (2) with  $n$  fixed, larger  $m$  means fewer disks need to be used to store the same dataset. For increasing  $n$ , figure 6.4 shows that throughput increases linearly, as more disks are used.

The throughput and availability trends suggest similar design decisions. For higher throughput and availability, we want larger  $n$  and smaller  $m$ . However, specific levels of desired throughput and availability might not be satisfiable at the same time.

**Impact of Fixed Number of Disks.** We just examined the effect of the redundancy configuration assuming a variable number of disks. Here, we evaluate the impact of using a fixed number of disks,  $N = 64$ . Under this assumption, all but one configuration ( $n = 16, m = 1$ ) will have unused storage space.

In this scenario, the availability does not change from the graphs in figures 6.3 and 6.4. The throughput curves drop only by a factor of  $1/m$  when  $m$  is increased, due to reason (1) listed above. When  $n$  is increased, write throughput decreases by a factor of  $1/n$  (compared to  $n = 1$ ), whereas the read throughput is unaffected. Overall, our observations from figures 6.3 and 6.4 still apply here.

Figure 6.5: Energy savings for  $n = 16$ .Figure 6.6: Energy savings for  $m = 1$ .

## 6.5.2 Energy

We now evaluate the effectiveness of the energy conservation techniques, as a function of the redundancy configuration. We start by assuming that FT is the primitive power-management technique used by MAID, PDC, DIV, MAID+DIV, and PDC+DIV and that the number of disks varies.

Figures 6.5 and 6.6 assess the impact of varying  $m$  (for  $n = 16$ ) and  $n$  (for  $m = 1$ ), respectively, on the energy savings produced by each technique. We compute the energy savings with respect to an energy-oblivious (EO) storage system. Note though that the percentages of savings represent different absolute energy consumptions, as the number of disks is not fixed. For example, EO consumes 10445 W average power for  $m = 1$  and 653 W for  $m = 16$ . Note also that some configurations are infeasible under PDC and PDC+DIV and are thus not included in the graphs. These configurations lead to fractional numbers of disk sets (see Section 6.4).

From figure 6.5, we can see that FT provides no energy savings across the space. Under the default request rate, there is not enough idle time for energy conservation in FT. In contrast, MAID conserves energy for small  $m$  but degrades

as we increase this parameter. In fact, MAID conserves a substantial amount of energy when  $m = 1$ , since the idle time on the non-cache disks is high enough under the default request rate and MAID fragment miss rate. However, when  $m \geq 2$ , MAID consumes more energy than EO because the cache disks do not filter enough accesses, and thus do not justify their energy overhead. PDC produces roughly the same energy savings, regardless of  $m$ . The reason is that these savings result from the constant file system coverage we assume.

DIV behaves very well until  $m$  starts approaching  $n$ ; larger  $m$  means that there is less redundancy, so fewer disks can be in low-power mode during read operations. Comparing the techniques, we find that DIV conserves more energy than MAID regardless of  $m$ . In comparison to PDC, DIV conserves more energy when the amount of redundancy is high, i.e.  $n \gg m$ .

Combining MAID or PDC with DIV improves these techniques substantially. MAID+DIV behaves similarly to DIV and significantly better than MAID for highly redundant configurations. With little redundancy, MAID+DIV behaves worse than DIV, consuming more energy than EO. PDC+DIV conserves significantly more energy than PDC for highly redundant systems. In contrast, PDC+DIV conserves more energy than DIV when redundancy is limited. Overall, PDC+DIV behaves best, as it combines the benefits of DIV and PDC under high and low redundancy, respectively.

From figure 6.6, we see that all techniques and combinations benefit from increases in redundancy, except for PDC and FT. MAID consumes more energy than EO for small  $n$ , but produces savings when the system becomes highly redundant. These savings come from the increase in the number of cache disks that results from increasing  $n$  (recall that we assume  $D_{maid}$  to be a fixed percentage of

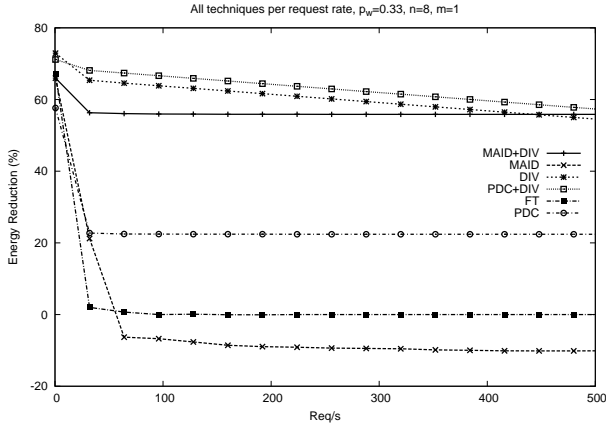


Figure 6.7: Savings per request rate for  $n = 8, m = 1$ .

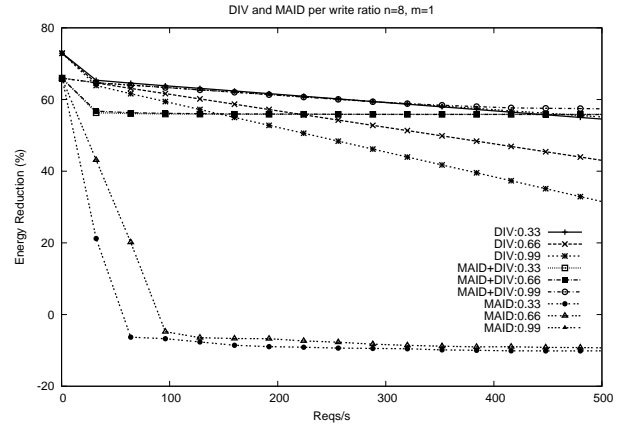


Figure 6.8: Savings per write percent for  $n = 8, m = 1$ .

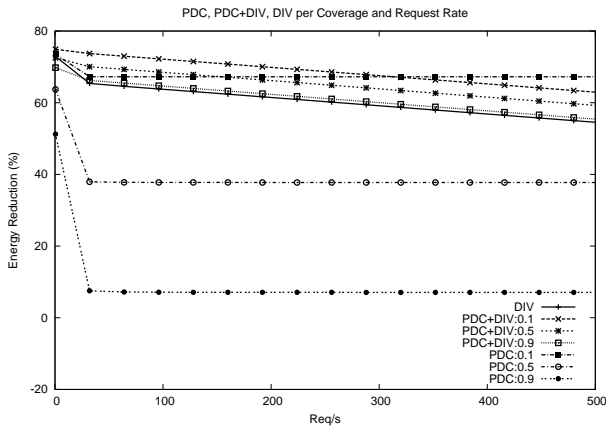


Figure 6.9: PDC and PDC+DIV savings as a function of coverage (10%, 50%, 90%) for  $n = 8, m = 1$ .

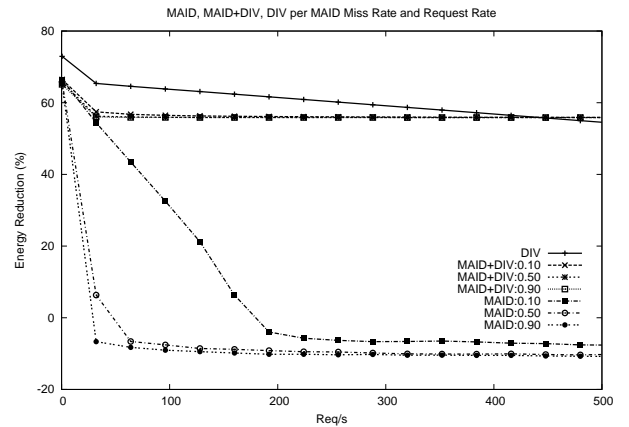


Figure 6.10: MAID and MAID+DIV savings as a function of cache miss rate (10%, 50%, 90%) for  $n = 8, m = 1$ .

$N$ ). DIV and its combinations produce the most significant and consistent energy savings, since DIV allows an increasing number of disks to be sent to low-power mode with increasing redundancy. Again, PDC+DIV performs best as explained above.

**Impact of Workload Characteristics.** We now study the impact of different workload characteristics on energy conservation, starting with the request rate. Figure 6.7 shows the effect of the request rate on a configuration with  $n = 8, m = 1$ . Recall that our previous results assumed the default request rate of 64 reqs/s.

The figure shows that FT and MAID only conserve energy for very low request rates (less than 32 and 64 reqs/s, respectively). For higher request rates, more sophisticated organizations are clearly necessary to increase idle times. Both PDC and DIV provide energy savings, but DIV conserves between two and three times more energy than PDC in this range of request rates. Adding DIV to MAID or PDC improves their behavior significantly, especially in the case of MAID. In fact, MAID+DIV degrades very slowly with the increase in request rate, due to our optimistic assumptions regarding the write accesses (infinite write buffering at cache disks and large write batch size). Under these assumptions, the cache disks and the original disks remain active, but the redundant disks can be in low-power mode most of the time. The overall trends we see in the figure continue until the request rate becomes so high that all disks are needed to fulfill the bandwidth requirements (not shown).

We now turn to the percentage of writes in the workload. Figure 6.8 shows a request-rate graph for the techniques that differentiate reads and writes, MAID and the DIV variants, with  $n = 8, m = 1$ . We omit PDC+DIV because its results follow the exact same trends as those of DIV, but with slightly larger savings. The figure plots results for three  $p_w$  settings: 0.33, 0.66, and 0.99. Recall that our other results assume  $p_w = 0.33$ .

These results are interesting in that MAID and DIV behave very differently. MAID conserves the most energy for write-dominated workloads, due to our optimistic assumptions for this technique; when reads are a significant fraction of the accesses, MAID actually consumes more energy than EO. In contrast, DIV does best for read-dominated workloads, as the redundant disks can be kept in

low-power mode most of the time. Nevertheless, DIV also does well for write-dominated workloads, since the redundant disks are only activated when the write buffer fills up. MAID+DIV behaves very well for all write rates by combining the best characteristics of MAID and DIV.

Figure 6.9 shows the behavior of PDC and PDC+DIV, as a function of coverage and request rate. Recall that coverage is the percentage of blocks referenced (read or written) over all stored blocks. By default, we have been assuming a coverage of 70% and a request rate of 64 reqs/s. PDC benefits significantly from smaller coverages. In fact, PDC becomes the best technique for very low coverage and high request rate. PDC+DIV also benefits from smaller coverages, but the improvements over DIV are meager when compared with the benefits a smaller coverage has on PDC alone. We can also see that the effect of data migration is small, since the energy savings of PDC+DIV with 90% coverage are very similar to those of DIV.

Figure 6.10 shows the behavior of MAID and MAID+DIV, as a function of cache miss and request rates. The miss rate depends on the workload and on the number of MAID caches used. Here, we study how the miss rate affects energy conservation, without concern for how it is achieved, i.e. without changing other parameters. The figure shows that the miss rate has a significant effect on MAID but not on MAID+DIV. MAID improves with lower miss rate, since lower rates increase the idle time at the non-cache disks, allowing them to be kept in low-power mode longer. MAID+DIV only benefits from extremely low miss rates (much lower than 10%), at which point the original (non-cache) disks can be sent to low-power mode for a long enough period. Again, this shows that MAID+DIV behaves well solely due to DIV and our favorable modeling assumptions.

**Impact of Write Buffer and Write Batch Sizes.** The size of the write buffer is tremendously important for DIV when workloads exhibit a non-trivial fraction of writes. For these workloads, DIV may not be able to conserve energy in the absence of a write buffer. As we increase the size of the write buffer, energy savings increase quickly at first but later taper off, as the redundant disks remain idle most of the time. Under our default parameters, the energy savings start to taper off with a 2-MB write buffer.

The behavior of MAID+DIV is significantly affected by a similar parameter, the write batch size. The batch size in MAID+DIV can be much larger than the buffer size in DIV, since writes are buffered on the cache disks in the former technique and main memory in the latter. When the batch size is the same as the buffer size, MAID+DIV conserves less energy than DIV, due to the energy overhead of the cache disks.

**Impact of Fixed Number of Disks.** Fixing  $N = 64$  across the spectrum of configurations leads to similar trends as in figures 6.5 and 6.6. Increases in  $n$  tend to increase the DIV savings initially, since larger  $n$  increases the number of redundant disks. At some point however, write accesses start limiting the idle time of a large number of disks. Increasing  $m$  reduces the DIV savings consistently, since this effectively increases the number of original disks.

**Impact of Oracle.** We also modeled the techniques assuming Oracle as the primitive power-management strategy, instead of FT. As one would expect, the result is that the techniques were able to conserve a little more energy at low request rates. For higher request rates, Oracle can block the power-mode transitions that would lead to higher energy consumption, but this feature did not affect the energy savings noticeably. When Oracle is used in isolation, it behaves

a little better than FT but for very low request rates only.

### 6.5.3 Defining a Redundancy Configuration

We illustrate the use of our models in the design of a redundancy configuration with a simple example. Suppose you need to design a system that requires: at least 20 disks to store all the data, at least 5 MB/s of throughput, and at least 0.99999 of availability.

We evaluated all possible combinations of  $n, m \in [1..16]$  for this example and our default model parameters. Table 6.3 summarizes the details of some of the candidate combinations. From left to right, the table lists the redundancy configuration, the number of disks, its throughput, its availability, the energy conservation technique, and the average power consumption.

In the table, the first group of rows shows the optimal configuration, (8, 5) with PDC+DIV for energy conservation. In this configuration, the two best techniques, DIV and PDC+DIV, consume 18% and 30% less energy than EO, respectively. It is interesting to note that the optimal configuration is somewhat non-intuitive; the intuitive ones are either invalid or sub-optimal. For example, the simplest redundant configuration ((2, 1), not shown) uses 40 disks, delivers enough throughput, but provides insufficient availability. The second group of rows shows results under another simple and intuitive mirrored configuration, (3, 1). For this configuration, DIV and PDC+DIV conserve 47% and 54% of the energy consumed by EO, respectively. The third group shows results for another intuitive configuration, (4, 2). Again, DIV and PDC+DIV are the best techniques for this configuration, conserving 32% and 44% of the energy, respectively. Finally, the fourth group shows results for yet another intuitive configuration, (8, 1). We

$(n, m)$	$N$	$P$ (MB/s)	$A$	Tech	$E$ (W)
(8, 5)	32	8.1	0.999999	EO	326
(8, 5)	32	8.1	0.999999	FT	326
(8, 5)	32	9.1	0.999999	MAID	367
(8, 5)	32	8.1	0.999999	PDC	265
(8, 5)	32	8.1	0.999999	DIV	267
(8, 5)	32	9.1	0.999999	MAID+DIV	275
(8, 5)	32	8.1	0.999999	PDC+DIV	228
(3, 1)	60	66.0	0.999999	EO	612
(3, 1)	60	66.0	0.999999	FT	612
(3, 1)	60	72.6	0.999999	MAID	674
(3, 1)	60	66.0	0.999999	PDC	473
(3, 1)	60	66.0	0.999999	DIV	326
(3, 1)	60	72.6	0.999999	MAID+DIV	365
(3, 1)	60	66.0	0.999999	PDC+DIV	280
(4, 2)	40	23.8	0.999996	EO	408
(4, 2)	40	23.8	0.999996	FT	408
(4, 2)	40	26.2	0.999996	MAID	452
(4, 2)	40	23.8	0.999996	PDC	316
(4, 2)	40	23.8	0.999996	DIV	276
(4, 2)	40	26.2	0.999996	MAID+DIV	295
(4, 2)	40	23.8	0.999996	PDC+DIV	230
(8, 1)	160	160.4	1.000000	EO	1632
(8, 1)	160	160.4	1.000000	FT	1633
(8, 1)	160	176.5	1.000000	MAID	1774
(8, 1)	160	160.4	1.000000	PDC	1262
(8, 1)	160	160.4	1.000000	DIV	631
(8, 1)	160	176.5	1.000000	MAID+DIV	717
(8, 1)	160	160.4	1.000000	PDC+DIV	585

Table 6.3: Sample candidate solutions for simple example.

might see this kind of large and highly redundant configuration in wide-area storage systems, in which many types of faults can cause parts of the system to become inaccessible. In this scenario, DIV and PDC+DIV consume 61% and 64% less energy than EO, respectively.

### 6.5.4 Summary

From the results above, it is clear that DIV (independently or in combination with other techniques) is an effective technique. In most of the parameter space, the DIV energy savings are significant and consistent. DIV is particularly effective for high  $n$ , low  $m$ , and read-mostly workloads.

DIV is the very reason why MAID+DIV and PDC+DIV behave well; MAID and PDC independently are neither robust nor energy-efficient in most cases. MAID+DIV behaves better than DIV in part of the space, mostly due to our highly favorable modeling of write accesses in MAID+DIV (and MAID). However, in other parts of the space, the cache disks contribute little besides energy overhead; in those scenarios, MAID+DIV consumes more energy than EO.

PDC+DIV conserves more energy than DIV when redundancy is limited. However, we also modeled PDC+DIV (and PDC) under favorable assumptions: perfect popularity categorization and no migration costs. Furthermore, PDC+DIV has one major drawback: it is very complex to implement in practice, especially in the context of a distributed storage system. In fact, determining the best data layout for energy and bandwidth is clearly NP-hard.

Based on these observations, we argue that DIV is the only effective, robust, and practical redundancy-aware energy conservation technique.

It is also clear from our results that the task of a storage system designer is not simple. Choosing the right redundancy configuration requires making informed decisions based on all the system requirements. Our simple example showed that non-intuitive redundancy configurations may actually lead to the best results.

$k$	N	ReqRate (reqs/s)	$p_w$	Tech	Buffer Size (MB)	Energy Savings		Error (%)
						Sim (%)	Model (%)	
3	3	0.01	0.25	DIV	0	62.1	62.4	0.8
3	3	0.01	0.25	DIV	1	66.6	66.7	0.4
3	3	0.01	0.25	DIV	8	66.6	66.8	0.4
3	3	0.01	0.25	DIV	$\infty$	66.6	66.8	0.4
3	6	10	0.50	DIV	0	0.0	0.0	0.0
3	6	10	0.50	DIV	1	15.6	16.8	1.0
3	6	10	0.50	DIV	8	45.7	46.1	0.6
3	6	10	0.50	DIV	$\infty$	49.3	50.3	2.0
3	15	100	0.75	DIV	0	0.0	0.0	-0.0
3	15	100	0.75	DIV	1	0.0	2.6	2.6
3	15	100	0.75	DIV	8	24.5	24.8	-0.3
3	15	100	0.75	DIV	$\infty$	49.6	50.1	1.0
5	20	100	0.75	DIV	0	0.0	0.0	0.0
5	20	100	0.75	DIV	1	0.0	2.5	2.5
5	20	100	0.75	DIV	8	21.7	22.2	0.8
5	20	100	0.75	DIV	$\infty$	59.8	60.1	0.8
3	3	0.1	0.0	FT	0	13.0	12.8	-0.3
3	9	0.1	0.0	FT	0	46.0	46.5	0.9
3	15	0.1	0.0	FT	0	57.8	58.1	0.7
3	3	10	0.0	FT	0	0.0	0.0	0.0
3	9	10	0.0	FT	0	0.0	-0.1	-0.1
3	15	10	0.0	FT	0	0.0	0.0	0.0

Table 6.4: Sample of validation results.

## 6.6 A Case Study: PAST

We now study similar tradeoffs in the PAST storage system [89]. This study serves two purposes: (1) to validate our energy models using simulation of a distributed storage system; and (2) to demonstrate DIV in the context of both real and synthetic workloads.

PAST is a large-scale, peer-to-peer file system that provides scalability, high availability, and high reliability (strong persistence). For availability and persistence, PAST keeps at least  $k$  copies of every file on different geographically distributed nodes. PAST stores files of any size in their entirety on each node.

Each file is identified uniquely by a 128-bit key, called `fileId`. File access requests are routed to the appropriate nodes using a peer-to-peer routing infrastructure, called Pastry [90]. Given a `fileId`, Pastry routes an associated message to the node whose node identification is numerically closest to the key, among all live nodes. Routing is done in  $\lceil \log_{2^b} N \rceil$  steps, where  $N$  is the number of nodes and  $b$  is a configuration parameter with typical value 4. Once the first copy of the file (out of the  $k$  available) is reached, PAST returns the entire file contents to the requesting node using the reverse path. Along the way, extra copies of the file might be cached to improve performance and increase availability.

To model PAST, we map fixed-sized blocks onto PAST files without loss of generality. We can then describe PAST using  $n = k, m = 1$ , since  $k$  copies are always available and only one is required to retrieve the original data. Even though we could consider power-managing entire PAST nodes, we continue focusing on disks (one disk per node, for simplicity).

To simulate PAST, we have developed a trace-based simulator that implements the relevant aspects of the system for this work: routing and replication. We did not simulate the caching of blocks along access routes to promote energy conservation (at the cost of potentially higher latency). Furthermore, cached blocks could be written to disks when they were active, so this decision should not affect our energy results at all. We simulate DIV, FT, and EO. The simulator selects a random node to receive each request in the trace. The request is then routed according to Pastry to the destination node and the reply is routed back. At each node, a fixed network latency of 50 ms is added. We did not experiment with variable network latency because we did not want to add sources of noise to the energy computation, which would make it hard to isolate where benefits

come from. We simulate the same IBM Ultrastar disks we have been studying.

### 6.6.1 Model Validation

In this section, we validate our energy models against the PAST simulator using synthetic workloads. Our synthetic workload generator takes the request rate and percentage of writes as input and produces a trace with 10,000 request arrivals drawn from a Pareto distribution with infinite variance. Each request is directed to a different disk (in round-robin fashion) and accesses a block of 8 KB. Note that the generation of our synthetic traces differs markedly from our modeling approach. In particular, each request arrival corresponds to a single disk access in our synthetic traces.

We executed a large number of simulations varying six different system and workload parameters:  $p_w \in [0, 0.25, 0.5, 0.75, 1]$ ,  $k \in [3, 5]$ ,  $req\_rate \in [0.01, 0.1, 10, 100, 1000]$ , energy conservation technique  $\in [FT, DIV]$ ,  $N \in [3, 6, 9, 10, 15, 20]$ , and  $wbSize \in [0, 1, 8, \infty]$ . Combinations of parameters that either do not make sense (e.g.,  $k > N$ ) or require more bandwidth than  $N$  disks can produce were discarded. We then compared the disk energy results of the remaining 795 simulations with the corresponding modeling results.

Table 6.4 shows a selected fraction of our validation results. The last column of the table shows the percentage difference between the energy consumption predicted by model and simulator. Our modeling results match the simulation results closely; the average error is 1.3%, the standard deviation is 2.8%, and the maximum error is 18%. If requests are directed to disks randomly (rather than in round-robin fashion), these values become 3.4%, 8.1%, and 28%, respectively.

The simulation and modeling trends match very closely. Again, DIV is most

effective for high redundancy (large  $k$ ), read-mostly workloads, and larger write buffer sizes. Also, FT is again only effective for very low request rates. These results build confidence on the parameter space study we present in Section 5.5.

## 6.6.2 Real Workload Results

We also wanted to simulate PAST for real traces. Unfortunately, real peer-to-peer *file access* traces are not available in the public domain. As an approximation, we used two proxy traces from AT&T and the IRCache project. The AT&T trace was collected between 01/16/99 and 01/22/99, whereas the IRCache trace was collected at three locations from 09/29/04 to 10/05/04. To mimic a system in which files are stored on disk and later accessed by peers, we pre-processed the traces to transform all file accesses into file read operations. We also introduced a write access for each unique file at a random time before the first access to it. After pre-processing, the AT&T trace exhibits 21,150,244 block requests, a 34% write percentage, an average request rate of 35 reqs/s, and a peak request rate of 2266 reqs/s. The IRCache trace includes 42,976,431 block requests, a 34% write percentage, an average request rate of 71 reqs/s, and a peak request rate of 10635 reqs/s. The traces are small ( $\leq 109$  GB each), so even one disk would be enough to store all the accessed data.

Before simulating, we need to define the ideal redundancy configuration for these workloads. First, we set the maximum throughput requirements as their peak request rates. Second, we set the target availability for the system at two different levels: 6 nines (IRCache) and 9 nines (AT&T). The latter is the same availability as in the original PAST paper. As a final constraint, we set  $m = 1$ , since this is the only setting that PAST handles.

Buffer Size (MB)	Energy (MJ)	Spin Downs	Avg Idle Time (s)	Reqs Delayed (%)	Avg Hop Count	Energy Savings (%)
0	120.3	5068	0.3	0.0	2.1	2.5
1	99.0	168524	27.9	0.8	0.9	19.7
2	78.1	110744	60.2	0.5	0.9	36.7
4	63.4	57072	128.7	0.3	0.9	48.6
8	55.9	29104	266.1	0.2	0.9	54.7
16	52.1	15024	541.1	0.1	0.9	57.7
32	50.2	7984	1090.9	0.1	0.9	59.3
64	49.3	4464	2190.6	0.0	0.9	60.0
128	48.8	2704	4389.9	0.0	0.9	60.4
$\infty$	48.4	1152	37209.9	0.0	0.9	60.8

Table 6.5: DIV results for AT&T trace. Average hop count is for one-way messages only.

Buffer Size (MB)	Energy (MJ)	Spin Downs	Avg Idle Time (s)	Reqs Delayed (%)	Avg Hop Count	Energy Savings (%)
0	499.6	2133	0.9	0.0	2.7	0.0
1	307.6	220699	89.9	0.5	1.6	38.4
2	278.2	111791	186.8	0.3	1.6	44.3
4	263.2	56225	380.6	0.1	1.6	47.3
8	255.7	28199	768.2	0.1	1.6	48.8
16	251.9	14159	1543.6	0.0	1.6	49.6
32	250.0	7139	3094.4	0.0	1.6	49.9
64	249.1	3629	6195.9	0.0	1.6	50.1
128	248.6	1847	12592.8	0.0	1.6	50.2
$\infty$	248.2	335	100792.6	0.0	1.6	50.3

Table 6.6: DIV results for IRCache trace. Average hop count is for one-way messages only.

Assuming these constraints and parameters, our optimization procedure finds  $N = 20$  disks and  $n = k = 5$  (AT&T) and  $N = 81$  disks and  $n = k = 3$  (IRCache) as the best configurations. We assess the DIV behavior on these realistic traces by simulating the system with this configuration and different write buffer sizes. We list these results in tables 6.6.2 and 6.6. From left to right, the table lists the write buffer sizes, the amount of energy consumed during the trace, the number of disk spin downs, the average idle times, the percentage of requests that were delayed by disk spin ups, the average number of hops traversed per one-way message, and the DIV energy savings with respect to EO. The tables show that DIV conserves between 20% and 61% of the energy, depending on the size of the write buffers.

When we cripple DIV by eliminating its write buffers, it conserves little if any energy. As we saw in Section 5.5, increasing the size of the write buffers increases savings significantly at first, but the gains progressively taper off. Although we do not include this information in the tables, our DIV energy model matches the simulation results closely; the average errors are 3.4% (AT&T) and 1.8% (IRCache), whereas the maximum errors are 10% (AT&T) and 8% (IRCache).

## 6.7 Conclusions

In this chapter, we introduced Diverted Accesses, the first energy conservation technique designed to leverage the redundancy in storage systems. We also introduced models that predict the disk energy consumption of Diverted Accesses and the previous techniques, as a function of the system's redundancy configuration. Our evaluation coupled a wide parameter space exploration with simulations of a real storage system under two realistic workloads. Our modeling and simulation results showed that Diverted Accesses is very effective and robust throughout most of the parameter space; other techniques are either not robust or impractical. Furthermore, we found non-intuitive redundancy configurations to be ideal in a couple of different examples. For our real system, realistic workloads, and ideal configuration, Diverted Accesses was able to conserve 20-61% of the disk energy consumed by an energy-oblivious storage system.

We conclude that considering redundancy can provide significant disk energy savings beyond those of previous techniques. Furthermore, we conclude that designing a storage system requires quantifying and trading off several different metrics, which are all affected by the redundancy configuration. Our models are

useful in this design process.

Finally, Diverted Accesses should be extremely useful for large-scale storage systems, such as outsourced storage services or wide-area publish-subscribe, backup, and archival systems. In fact, we believe that our technique would be even more beneficial (in absolute energy consumption terms) if applied to entire storage nodes rather than just their disks. We intend to address this issue in the near future.

## Chapter 7

### Conclusions and Future Work

In this thesis we made the case for power management and energy conservation in server systems. We are among the first researchers to propose and study this new research direction.

We have shown that careful resource management, adjusted to the imposed load on a server system, can conserve energy without significant performance degradation. Our Load Concentration technique conserves around 40% of energy of a cluster by turning entire nodes off. This kind of coarse-grain control has the highest absolute savings and is applicable to a variety of environments, from data centers and scientific computing clusters to research and academic environments. Load Concentration has to be implemented carefully however, since booting up a node to handle an increase in demand can take more than a minute.

Our Popular Data Concentration and Diverted Accesses techniques are more fine-grained, providing smaller absolute savings, but still a significant fraction of the energy consumed by servers. Their finer control provides for more flexibility and agility in how and when other components of the server system will be power-managed. For example, if a server system needs prompt availability of the processing resource (CPU), then power-managing disks and CPU differently (and individually) is better than having to bring up an entire node and thus pay the price of the long boot up delay.

We showed that Popular Data Concentration needs to be applied to an array of multi-speed disks to be effective under realistic loads. It can save a substantial amount of energy depending on the parameters of the workload, such as coverage, request rate, and write ratio as long as two-speed disks are used.

We also showed that Diverted Accesses can save significant amounts of energy depending on the redundancy parameters used and the intensity of the workload. We suggested that Diverted Accesses can be applied to entire servers, making DIV a very flexible technique that can be adjusted to tailor individual needs.

Although our techniques are important first steps, several areas exist for future research. Energy consumption in the memory subsystem of servers, for example, is a problem that remains open. There might be characteristics of servers' memory access patterns that could be exploited in a similar vein as the characteristics we exploit for entire servers and the storage subsystem. For example, popular memory blocks might be migrated to a subset of the banks in a memory module, similar to what PDC does.

Further, temperature management and its relationship to power and energy management need more study. Since temperature is directly related to power, we need to find out if resource management techniques for power can also be applied to manage temperature. We have early indication that managing temperature is not the same as managing power, since there is inertia in temperature movement that does not occur with power expenditure.

Our Load Concentration technique can be extended to take into account the physical layout of a machine room. It could then handle temperature emergencies – partial breakdown in cooling equipment – by shifting load towards the cooler set of cluster nodes while shutting down the affected nodes. The ramifications of

this study and its tradeoffs are our next step beyond this thesis.

Finally, the interplay between hardware reliability, performance, and temperature management is not clear yet. It is well-known that increases in temperature cause hardware components to fail. However, some devices may show soft-failure (recoverable) behavior with increases in temperature before they break, whereas others might show a binary behavior in which they work perfectly (perhaps only slower) and then break when a maximum temperature is reached. If their mode of operation is known and can be analytically modeled, we can create a policy that optimizes temperature management, performance, and reliability. The goal is to obtain the “sweet spot” in the tradeoff curve between performance and temperature. That is, we seek to balance small increases in temperature (to save the cooling energy/money) at the expense of slightly longer latencies (due to soft-failures/slower performance) and decreased reliability (just to the point where components will not break significantly more often than they would otherwise). We believe this is an attainable short-term goal.

## References

- [1] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems. *ACM Transactions on Computer Systems*, 19(4):483–518, November 2001.
- [2] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles Around Storage Administration. In *Proceedings of the International Conference on File and Storage Technology*, pages 175–188, January 2002.
- [3] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang. Ergastulum: Quickly Finding Near-Optimal Storage System Designs. Technical Report HPL-SSP-2001-05, HP Laboratories SSP, June 2002.
- [4] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rochwerger. Oceano - SLA Based Management of a Computing Utility. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001.
- [5] Harvard Gazette Archives. Report focuses on impact of power plant pollution. May 2000. <http://www.news.harvard.edu/gazette/2000/05.11/air.html>.
- [6] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS00)*, pages 90–101, 2000.
- [7] M. Bakkaloglu, J. Wylie, C. Wang, and G. Ganger. On Correlated Failures in Survivable Storage Systems. Technical Report CMU-CS-02-129, Carnegie Mellon University, School of Computer Science, May 2002.
- [8] R. Balasubramonian, David H. Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *International Symposium on Microarchitecture (MICRO-33)*, pages 245–257, 2000.

- [9] R. Balasubramonian, V. Srinivasan, S. Dwarkadas, and A. Buyuktosunoglu. Hot-and-cold: Using criticality in the design of energy-efficient caches. In *3rd Workshop on Power-Aware Computer Systems (PACS03)*, December 2003.
- [10] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI99)*, pages 45–58, 1999.
- [11] Amnon Barak and Oren La’adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Journal of Future Generation Computer Systems*, 13(4-5):361–372, March 1998.
- [12] P. Barford and M. Crovella. An Architecture for a WWW Workload Generator. In *Proceedings of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS98)*, 1998.
- [13] Luca Benini, Alessandro Bogliolo, Stefano Cavallucci, and Bruno Riccó. Monitoring System Activity for OS-Directed Dynamic Power Management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED98)*, pages 185–190, 1998.
- [14] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed Packet Rewriting and its Application to Scalable Server Architectures. In *Proceedings of the International Conference on Network Protocols*, October 1998.
- [15] R. Bhagwan, D. Moore, S. Savage, and G. M. Voelker. Replication Strategies for Highly Available Peer-to-Peer Storage. In *Proceedings of International Workshop on Future Directions in Distributed Computing*, May 2002.
- [16] R. Bianchini and R. Rajamony. Power and Energy Management for Server Systems. *IEEE Computer*, 37(11), November 2004.
- [17] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The Case for Power Management in Web Servers. In Graybill and Melhem, editors, *Power-Aware Computing*. Kluwer Academic Publishers, January 2002.
- [18] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of IEEE INFOCOM*, March 1999.

- [19] E. V. Carrera and R. Bianchini. Improving Disk Throughput in Data-Intensive Servers. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture (HPCA04)*, February 2004.
- [20] E. V. Carrera and R. Bianchini. Efficiency vs. Portability in Cluster-Based Network Servers. In *Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP01)*, June 2001.
- [21] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *Proceedings of the 17th International Conference on Supercomputing (ICS03)*, June 2003.
- [22] Surendar Chandra, Carla Schlatter Ellis, and Amin Vahdat. Differentiated Multimedia Web Services Using Quality Aware Transcoding. In *Proceedings of INFOCOM 2000 - Nineteenth Annual Joint Conference of the IEEE Computer And Communications Societies*, 2000.
- [23] J. Chase, D. Anderson, P. Thacker, A. Vahdat, and R. Boyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP01)*, October 2001.
- [24] Cisco LocalDirector. <http://www.cisco.com/>, 2000.
- [25] D. Colarelli and D. Grunwald. Massive Arrays of Idle Disks For Storage Archives. In *Proceedings of the 15th High Performance Networking and Computing Conference*, November 2002.
- [26] ComputerWorld. Net Blamed as Crisis Roils California. January 2001. <http://www.computerworld.com/managementtopics/outsourcing/story/0,10801,56341,00.html>.
- [27] John Cook. Internet data gain is a major power drain on local utilities. *Seattle Post Intelligencer Report*, September 2000. <http://seattlepi.nwsourc.com/business/data05.shtml>.
- [28] Transmeta Corporation. Tm5900 databook. [http://transmeta.com/crusoe\\_docs/tm5900\\_databook\\_040204.pdf](http://transmeta.com/crusoe_docs/tm5900_databook_040204.pdf).
- [29] R. Crisp. Direct rambus technology: The new main memory standard. *IEEE Micro*, 17(6):18–28, November 1997.
- [30] E. de Lara, D. Wallach, and W. Zwaenepoel. Puppeteer: Component-Based Adaptation for Mobile Computing. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS01)*, March 2001.

- [31] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. DRAM Energy Management Using Software and Hardware Directed Power Mode Control. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA01)*, January 2001.
- [32] F. Douglis and P. Krishnan. Adaptive Disk Spin-Down Policies for Mobile Computers. *Computing Systems*, 8(4):381–413, 1995.
- [33] Fred Douglis and J. Ousterhout. Transparent Process Migration: Design and Alternatives and the Sprite Implementation. *Software: Practice and Experience*, 21(8):757–785, August 1991.
- [34] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-Efficient Server Clusters. In *Proceedings of the 2nd Workshop on Power-Aware Computing Systems (PACS02)*, February 2002.
- [35] M. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS03)*, March 2003.
- [36] A. Adya et al. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI02)*, December 2002.
- [37] G. Elert et al. Power of an air conditioner. *The Physics Factbook*. <http://hypertextbook.com/facts/2001/KaFungKan.shtml>.
- [38] X. Fan, C S. Ellis, and A. R. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. In Springer-Verlag, editor, *Proceedings of Power Aware Computer Systems (PACS03)*, December 2003.
- [39] L. Feeney and M. Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *Proceedings of IEEE INFOCOM*, 2001.
- [40] Jason Flinn and M. Satyanarayanan. Energy-Aware Adaptation for Mobile Applications. In *Proceedings of the 17th Symposium on Operating Systems Principles (SOSP99)*, pages 48–63, 1999.

- [41] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. In *Proceedings of the International Symposium on Operating Systems Principles (SOSP97)*, pages 78–91, October 1997.
- [42] D. Ghormley, D. Petrou, S. Rodrigues, A. Vahdat, and T. Anderson. GLUnix: a Global Layer Unix for a Network of Workstations. *Software: Practice and Experience*, February 1998.
- [43] A. Goldberg and P. N. Yianilos. Towards an Archival Intermemory. In *Proceedings of IEEE Advances in Digital Libraries (ADL98)*, 1998.
- [44] G. Goldszmidt and G. Hunt. Scaling Internet Services by Dynamic Allocation of Connections. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*, pages 171–184, 1999.
- [45] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proceedings of the International Symposium on Computer Architecture*, June 2003.
- [46] S. Gurumurthi, J. Zhang, A. Sivasubramaniam, M. Kandemir, H. Franke, N. Vijaykrishnan, and M. J. Irwin. Interplay of Energy and Performance for Disk Arrays Running Transaction Processing Workloads. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS03)*, March 2003.
- [47] N. Hajj, C. Polychronopoulos, and G. Stamoulist. Architectural and compiler support for energy reduction in the memory hierarchy of high performance microprocessors. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED98)*, 1998.
- [48] T. Halfhill. Transmeta Breaks the x86 Low-Power Barrier. In *Microprocessor Report*, February 2000.
- [49] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Self-Configuring Heterogeneous Server Clusters. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP03)*, September 2003.
- [50] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Energy Conservation in Heterogeneous Server Clusters. In *Proceedings of the*

*10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP05)*, June 2005.

- [51] T. Heath, E. Pinheiro, and Ricardo Bianchini. Application-Supported Device Management. In *Proceeding of the 2002 Workshop on Power-Aware Computer Systems (PACS02)*, pages 114–123, February 2002.
- [52] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Code Transformations for Energy-Efficient Device Management. *IEEE Transactions on Computers*, 53(8), August 2004.
- [53] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Application Transformations for Energy and Performance-Aware Device Management. In *Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques (PACT02)*, September 2002.
- [54] David P. Helmbold, Darrell D. E. Long, Tracey L. Sconyers, and Bruce Sherrod. Adaptive disk spin-down for mobile computers. *Mobile Networks and Applications*, 5(4):285–297, 2000.
- [55] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A Dynamic Disk Spin-Down Technique for Mobile Computing. In *Proceedings of the 2nd International Conference on Mobile Computing (MOBICOM96)*, pages 130–142, 1996.
- [56] J. Hom and U. Kremer. Energy Management of Virtual Memory on Diskless Devices. In L. Benini, M. Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2003.
- [57] C-H. Hsu and U. Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *ACM SIGPLAN Conference on Programming Languages, Design, and Implementation (PLDI03)*, San Diego, CA, June 2003.
- [58] C-H. Hsu, U. Kremer, and M. Hsiao. Compiler-Directed Dynamic Frequency and Voltage Scaling. In *Proceedings of the Workshop on Power-Aware Computer Systems (PACS00)*, November 2000.
- [59] H. Huang, P. Pillai, and K. G. Shin. Design and implementation of power-aware virtual memory. In *Proceedings of USENIX Technical Conference (USENIX03)*, June 2003.

- [60] IBM. IBM A.B.L.E. Technology. <http://www.storage.ibm.com/hdd/library/able.htm>.
- [61] IOzone. IOzone Filesystem Benchmark. November 2000. <http://www.iozone.org>.
- [62] J. Kubiawicz et al. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS00)*, November 2000.
- [63] S. Jin and A. Bestavros. GISMO: A Generator of Internet Streaming Media Objects and Workloads. *ACM SIGMETRICS Performance Evaluation Review*, 29(3), November 2001.
- [64] S. Jin and A. Bestavros. Sources and Characteristics of Web Temporal Locality. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MAS-COTS00)*, August 2000.
- [65] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh-Cheng Chen. A survey of energy efficient network protocols for wireless networks. *Wireless Networks*, 7(4):343–358, 2001.
- [66] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. Dns performance and the effectiveness of caching. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.
- [67] A. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1):79–119, 1988.
- [68] E. Kim, K.H. Yum, G. Link, N. Vijaykrishnan, M. Kandemir, M. Yousif, M. J. Irwin, and C. R. Das. Energy optimization techniques in cluster interconnects. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED03)*, August 2003.
- [69] Johnson Kin, Munish Gupta, and William H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *International Symposium on Microarchitecture*, pages 184–193, 1997.
- [70] U. Kremer, J. Hicks, and J. Rehg. Compiler-Directed Remote Task Execution for Power Management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP00)*, October 2000.

- [71] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla S. Ellis. Power Aware Page Allocation. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS00)*, pages 105–116, 2000.
- [72] E. K. Lee and C. A. Thekkath. Petal: Distributed Virtual Disks. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS96)*, 1996.
- [73] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *Proceedings of the USENIX Technical Conference (USENIX94)*, pages 279–291, 1994.
- [74] Y. Lin, S. Brandt, D. Long, and E. Miller. Power Conservation Strategies for MEMS-based Storage Devices. In *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS02)*, Fort Worth, TX, October 2002.
- [75] Michael J. Litzkow and Marvin Solomon. Supporting Checkpoint and Process Migration Outside the UNIX Kernel. In *Proceedings of the USENIX Technical Conference (USENIX92)*, pages 283–290, San Francisco, CA, January 1992.
- [76] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Operating-System Directed Power Reduction. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED00)*, 2000.
- [77] Sun Microsystems. Site planning guide for entry-level servers. <http://docs.sun.com/source/816-1613-13/Chapter5.html#34676>.
- [78] J. D. Mitchell-Jackson. Energy needs in an internet economy: A closer look at data centers, 2001.
- [79] T. Mudge. Power: A first class design constraint. *IEEE Computer*, 34(4):52–57, April 2001.
- [80] Find my Web Hosting. <http://www.findmyhosting.com/>.
- [81] K. Okada, N. Kojima, and K. Yamashita. A Novel Drive Architecture of HDD: "Multimode Hard Disk Drive". In *Proceedings of the International Conference on Consumer Electronics*, June 2000.

- [82] A. Papathanasiou and M. Scott. Power-efficient server-class performance from arrays of laptop disks. Technical Report 837, University of Rochester, May 2004.
- [83] Athanasios E. Papathanasiou and Michael L. Scott. Energy Efficient Prefetching and Caching. In *Proceedings of the USENIX Technical Conference (USENIX04)*, Boston, MA, June 2004.
- [84] L. Peh. Power modeling of interconnection networks. In *Power Summit*, November 2002.
- [85] E. Pinheiro and R. Bianchini. Energy Conservation Techniques for Disk Array-Based Servers. In *Proceedings of the 18th International Conference on Supercomputing (ICS04)*, June 2004.
- [86] E. Pinheiro and R. Bianchini. Nomad: A Scalable Operating System for Clusters of Uni and Multiprocessors. In *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, December 1999.
- [87] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP01)*, September 2001.
- [88] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. Technical Report DCS-TR-440, Department of Computer Science, Rutgers University, May 2001.
- [89] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of the International Symposium on Operating Systems Principles (SOSP01)*, 2001.
- [90] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the International Conference on Distributed Systems Platforms (Middleware)*, November 2001.
- [91] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving Portable Computer Battery Power through Remote Process

- Execution. *Mobile Computing and Communications Review*, 2(1):19–26, 1998.
- [92] L. Shang, L. Peh, and N. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA03)*, February 2003.
- [93] L. Shang, L. Peh, and N. K. Jha. Powerherd: Dynamically satisfying peak power constraints in interconnection networks. In *Proceedings of the 17th International Conference on Supercomputing (ICS03)*, June 2003.
- [94] D. Siewiorek and R. Swarz. *Reliable Computer Systems Design and Evaluation*. A K Peters, third edition, 1998.
- [95] Bob Vila The Ultimate Home Site. [http://www.bobvila.com/ArticleLibrary/Subject/Energy\\_Efficiency](http://www.bobvila.com/ArticleLibrary/Subject/Energy_Efficiency)
- [96] K. Skadron, M. Stan, and T. Abdelzaher. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA02)*, February 2002.
- [97] V. Soteriou and L. Peh. Dynamic power management for power optimization of interconnection networks using on/off links. In *Proceedings of the 11th Symposium on High Performance Interconnects (Hot Interconnects)*, August 2003.
- [98] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8):1125–31, 1997.
- [99] Superb. <http://services.superb.net/colo/colo.php#SCS>.
- [100] A. Vahdat, A. Lebeck, and C. Ellis. Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency. In *Proceedings of the SIGOPS European Workshop*, September 2000.
- [101] Robbert Van Renesse, Ken Birman, Mark Hayden, Alexey Vaysburd, and David Karr. Building Adaptive Systems Using Ensemble. *Software Practice and Experience*, 28(9):963–979, 1998.

- [102] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-Driven Integrated Hardware-Software Optimizations Using Simple-Power. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA00)*, pages 95–106, 2000.
- [103] H. Weatherspoon and J. Kubiatowicz. Erasure Coding vs. Replication: A Quantitative Comparison. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, March 2002.
- [104] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of the 1st Symposium on Operating System Design and Implementation (OSDI94)*, November 1994.
- [105] H. Yan, R. Krishnan, S. Watterson, and D. Lowenthal. Client-centered energy savings for concurrent http connections. In *Proceedings of the 14th ACM Workshop on Networks and Operating System Support for Digital Audio and Video (NOSSDAV)*, June 2004.
- [106] H. Yan, R. Krishnan, S. Watterson, D. Lowenthal, K. Li, and L. Peterson. Client-centered energy and delay analysis for tcp downloads. In *Proceedings of the 12th IEEE International Workshop on Quality of Service (IWQoS)*, June 2004.
- [107] Do It Yourself. <http://doityourself.com/aircond/centralairconditionerefficiency.htm>.
- [108] Y. Zhou, P.M. Chen, , and K. Li. The Multi-Queue Replacement Algorithm for Second-Level Buffer Caches. In *Proceedings of the USENIX Technical Conference (USENIX01)*, June 2001.
- [109] Q. Zhu, F. M. David, Y. Zhou, C. F. Devaraj, P. Cao, and Z. Li. Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture (HPCA04)*, February 2004.
- [110] Q. Zhu, A. Shankar, and Y. Zhou. PB-LRU: A Self-Tuning Power Aware Storage Cache Replacement Algorithm for Conserving Disk Energy. In *Proceedings of the 18th International Conference on Supercomputing (ICS04)*, June 2004.