THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

PARAID: THE GEAR-SHIFTING POWER-AWARE RAID

By

CHARLES O. WEDDLE III

A Thesis submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Summer Semester, 2005

The members of the Committee approve the thesis of Charles O. Weddle III defended on June 22, 2005.

_____
An-I Andy Wang
Professor Directing Thesis

_____
Theodore P. Baker
Committee Member

_____
Kartik Gopalan
Committee Member

The Office of Graduate Studies has verified and approved the above named committee members.

To my Aunt Marilyn Schultz

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Andy Wang, for his guidance and support. Without his insight and support, this work would not have been possible. I would also like to thank my committee members Dr. Ted Baker and Dr. Kartik Gopalan for their help in this effort. I must also acknowledge my research partner Mathew Oldham for his contributions to this work. Mat's work on the measurement framework and trace playback program made the results obtainable. Finally, I would to thank my Aunt, Dr. Marilyn Schultz, for her guidance and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The rising cost of energy is becoming a concern beyond mobile computing platforms. Server-class computers cannot simply consume more power, since increased energy consumption translates into more heat dissipations, more cooling requirements, reduced computational density, and higher operating costs.

For a typical data center, storage alone accounts for 27% of the energy consumption, making storage an important target for energy reduction. Unfortunately, conventional server-class RAIDs are not designed for saving power, because loads are balanced in such a fashion that they require the use of all disks in the array for even light system loads.

This work introduces the Gear-Shifting Power-Aware RAID (PARAID), which reduces energy in server-class computing while retaining performance and reliability. The design of PARAID uses a skewed striping pattern to adapt to the system load by varying the number of powered disks. By powering off disks during periods of light load, PARAID can reduce the power consumed by a comparable conventional RAID device by 23%. By matching the number of powered disks to the system load, PARAID can also demonstrate request completion time and latency comparable to conventional RAID.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Energy consumption is an issue for many types of technology. The increasing cost of oil is making the consumption of gasoline by automobiles a concern. The energy consumed by computer technology has become just as much of a concern as the amount of gas burned in automobiles because the cost per MIPS is declining according to Moore's law, but the cost of electricity is increasing. Therefore, as computing becomes more affordable and ubiquitous, energy cost is poised to become a dominating fraction of owning and operating a computer.

The disk remains a significant source of power usage. In web servers, the disk typically accounts for 24% of the power usage; in proxy servers, 77% [2, 12]. Storage devices can account for as much as 27% of the operating cost in a typical data center [27]. The energy spent to operate servers in a data center has a cascading effect on other operating costs. Greater energy consumption leads to more heat dissipations, which in turn leads to greater cooling requirements [16]. The combined effects of energy consumption and heat also limit the density of computer racks. The lower density of computers leads to more space requirements, thus higher operating costs. The total cost of ownership in server-class computing can be significantly decreased by improving energy efficiency in server-class storage devices.

Approaches to reducing the energy consumption in disks have been explored, but most are achieved at the expense of degrading performance. Popular approaches involve trading off performance directly, such as reducing the rotational speed of the disk, causing read/write

requests to be slower [1, 2, 12, 19]. Not until recently have new approaches started to emerge to achieve both goals [4, 17].

Introducing power-saving techniques on server-class disks is challenging, because the performance and reliability introduced by conventional RAIDs must be maintained in order for a solution to be practical for commercial use. Conventional RAIDs balance the load across all disks in the array for maximized disk parallelism and performance [19]. To reduce power, a RAID device cannot simply power off disks and use caching. The load balancing in RAIDs keeps all disks spinning even when the server load is light. To be able to reduce power consumption, the individual disks in the array must be able to be powered on and off.

Frequent power cycles reduce the life expectancy of a disk due to their wear and tear on mechanical disks. When the life expectancy of the disk is reduced, the RAID device becomes significantly less reliable.

This thesis introduces the Gear-Shifting Power-Aware RAID (PARAID), which overcomes power, performance, and reliability challenges in server-class RAID devices. PARAID introduces a skewed striping pattern, which allows RAID devices to use just enough disks to meet the system load. PARAID can vary the number of powered disks by gear shifting sets of disks, giving PARAID the opportunity to reduce power consumption. PARAID has shown that power can be saved with limited degradations in performance and reliability. In respect to a comparable conventional RAID, PARAID can reduce the amount of power consumption on average by 23%, while maintaining comparable performance numbers.

1.2 Observations

Three fundamental observations drive the design of the PARAID:

Over-provisioned resources: When a system is not under peak load, disk load balancing over-provisions resources. A RAID device consists of an array of disks. To reduce the power consumption in a RAID effectively, all or some of its disks must be powered off. Conventional RAIDs use a uniform striping pattern that balances load on the array of disks. The balanced load allows a RAID device to maximize disk parallelism and performance. This uniformity makes data management simple and allows all disks to be accessed in the same way. Its built-in load balancing also ensures that no disk becomes a bottleneck.

However, load balancing created by a uniform striping pattern provides significantly fewer opportunities to power off disks because it requires all disks in the array to be powered on to serve a file. This means that when the server is under light load, all disks have to remain powered, even though a smaller array of disks could adequately handle the load. Load balancing leads to an over-provisioning of resources when the system is not under peak load.

Unused storage space: Increasingly, storage capacity is outgrowing its demands, and not all the storage space on a RAID device is used. Due to the 100% increase per year in aerial density of storage and the exponential drop in the storage pricing, researchers are increasingly looking for creative ways to consume the unused storage. For example, a work at Princeton explores trading off capacity for performance [26]. The Elephant file system also explores the possibility of storing every version of file updates [24].

Additionally, administrators tend to purchase more space in advance to avoid frequent upgrades [10]. Because of this overcompensation, storage space is left unused, which could be used opportunistically. This space could be used for data block replication or for some other storage to help reduce power consumption.

Cyclic fluctuating load: Many system loads display daily cyclic fluctuations. A 24-hour period of academic web traffic load displays activity as a bell curve with the afternoon being at the curve crest, reflecting students' schedules. Depending on the types of traffic, different systems may exhibit different cyclic patterns, with varying ranges of light to high loads over the course of a day [13].

The design of conventional RAIDs does not take advantage of this fluctuating load. In fact, conventional RAIDs provide the same level of performance regardless of system demands. Under periods of light load, a conventional RAID will have all of the disks powered even though a fraction of those disks could handle the load. This is certainly an opportunity lost.

1.3 Thesis

This thesis hypothesizes that by using a novel data distribution technique, it is possible to achieve energy savings while preserving performance. To do this, the PARAID design needs to provision resources according to fluctuating workloads. Since performance degradation is not an option for servers, PARAID, at the minimum, needs to strive to preserve peak performance.

PARAID should also take advantage of any unused space to conserve energy. Strategic data redundancy techniques can be used to create opportunities to power off disks. The overhead involved in maintaining redundancy information should not overpower the energy saving benefits. Finally, PARAID cannot be too aggressive using power switches to achieve energy saving, since server-class disks are not designed to be powered on and off frequently. PARAID needs to explore the cyclic workload behavior to power switch disks in a sparing and effective way.

The remaining thesis first reviews the basics of the RAID design (Chapter 2). Then, Chapter 3 presents PARAID, along with explaining how unused storage can be traded for both performance and energy savings. Chapters 4, 5, and 6 show a detailed PARAID design, which is prototyped and empirically evaluated. Finally, Chapter 7 relates PARAID to existing work; Chapter 8 discusses the some future directions of PARAID; and Chapter 9 presents a summary and conclusions.

CHAPTER 2
CONVENTIONAL SERVER CLASS RAIDS

In 1988, Berkeley introduced a concept in mass storage called Redundant Arrays of Inexpensive Disks or RAID [19]. RAID combines multiple disks into an array of disks that yields performance exceeding that of a single large expensive disk. The goals of conventional RAID are to increase disk performance by maximizing disk parallelism and to make storage more reliable through redundancy. It is necessary to understand how conventional RAID works to be able to understand the goals, design, and evaluation of the new Gear-Shifting Power-Aware RAID.

2.1 Physical and Logical Disk

The description of a RAID device directly ties to the physical and logical characteristics of a disk device. A physical disk consists of the mechanical hardware. A ***physical disk*** needs to be represented in computer software in terms of a ***logical disk***.

Figure 2.1 shows the mechanical parts of a disk. A disk contains one or more magnetic ***platters***, which store the data on the surface. Each platter is attached to a spindle, which is attached to a spindle motor. When the motor is powered, the platters rotate. For both surfaces of each platter, a ***disk head*** is responsible for reading and writing data. Each head is attached to a ***disk arm***, which is attached to the arm assembly that moves the arms together over the surface of the platters.

Figure 2.2 shows the logical representation of the disk. The logical disk represents the mechanical disk in software. Each surface on each platter is viewed logically as having concentric circles radiating out from the center of the platter surface to the edge of the platter

surface. Each circle represents a ***track*** of data on the platter. A track consists of ***sectors*** (Figure 2.2), and a sector is the smallest data access unit that can be read from or written to the disk. The same track on every surface of every platter forms a ***cylinder*** (Figure 2.1).

Disk access involves three timing components: seek time, rotational latency, and data transfer time. Seek time is the amount of time needed for the arm to move to the correct radial position on the platter surface. Rotational latency is the amount of time taken to rotate the desired sector under the disk head. The transfer time is the duration for data to be read or written to or from the platter's surface. Data transfer time depends on the rotation speed, the density of the magnetic media, and the number of sectors that can be stored in a track, which is a function of radial distance of the head from the center of the platter.

When discussing RAID, it is important to know what a sector is because the RAID software issues reads and writes at this access granularity. The algorithm used by the RAID software to distribute the content of a file into various sector locations on different disks is known as a ***striping pattern***, which in terms defines the ***personality*** (or the ***RAID level***) [19] of a RAID device. It is also important to know about the disk latency, a factor of seek time and rotational latency, and data transfer time because these metrics define the performance for a disk and a RAID device.

Finally, not all disks are the same. Server-class storage uses high performance disks to be able to meet peak demand on the server. A typical server-class disk will have a platter rotation between 10,000 and 15,000 rotations per minute and have an average latency of 2.99 ms [5, 6]. Lower performing disks, for example those used in laptop computers, rotate at 5400 to 7200 rotations per minute and have an average latency of 7.14 ms [7, 8].

One advantage to lower performing disks is that they use less power, typically 2 W when active and less then a Watt when inactive [7, 8]. This is much less power compared to a high performance server-class disk that uses around 13 Watts when active and 10 Watts when inactive [5, 6]. A possible alternative approach to save energy than that proposed by PARAID is to use lower performing disks in server class computers. This approach suffers from one critical disadvantage. These lower performing disks cannot meet the demand of the peak load for a server without installing additional lower performing disks. In addition to degraded performance, lower performing disks are less reliable [7, 8]. This alternative approach is not satisfactory because performance and reliability are degraded.

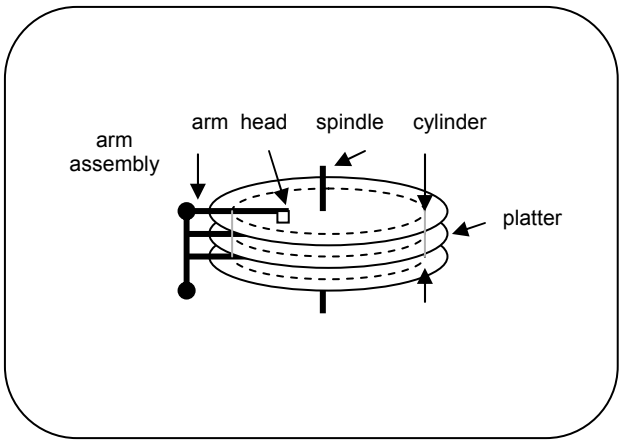

Figure 2.1: Physical disk components

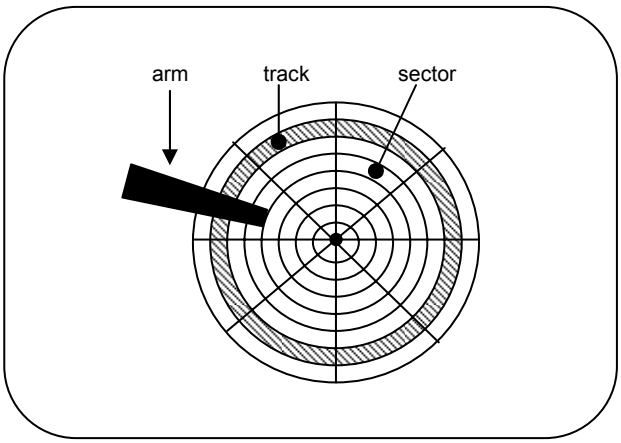

Figure 2.2: Logical disk components

2.2 Conventional RAID

RAID was introduced as an alternative to a single large disk because the performance of the CPU and memory continued to outpace that of a disk. RAID narrowed this performance gap by allowing multiple disks to access data in parallel. The success of RAID is evident today due to its continued wide-spread use in commercial, academic, and even personal computing.

A RAID makes several disks appear as one storage device. The logical view of a RAID device is a single storage device, which is also the view perceived from file systems above. Figure 2.3 shows the difference between the logical view of the RAID device and the physical disks. Each logical sector in a RAID is translated through a mapping function to one of the disks based on a given RAID striping pattern. Figure 2.3 shows how logical sectors are evenly distributed across the disks in the array creating a uniform striping pattern.

The uniform striping pattern is used by RAIDs to maximize the parallelism to access multiple disks. Since all disks are involved in every data request, the ***throughput*** (the bytes transferred within the duration of transfer) outperforms that of a single disk. However, the ***disk access latency*** (seek time and rotational delay) suffers, since each request has to wait for the slowest disk among all to access the data. To be able to write the sectors of data in a uniform striping pattern, RAID uses a mapping function to map the logical sector of data to a sector on a disk in the array.

Logically, RAID manages data in ***chunks***. Physically, a chunk of data consists of many sectors. Typical chunk sizes are 4, 16, 32, 64, or 128 Kbytes. For the common setting of 512-byte sectors, a 4-Kbyte logical chunk has eight sectors.

Chunks that reside in the corresponding locations of every disk in the array form a stripe in the array. Figure 2.3 shows how four chunks from the logical RAID make up a stripe on four disks. Logical chunk A is mapped to the first disk in the array; B to the second; and so on. When the chunks of data are read from the disks, reads are performed in parallel, effectively improving the read performance over a single disk.

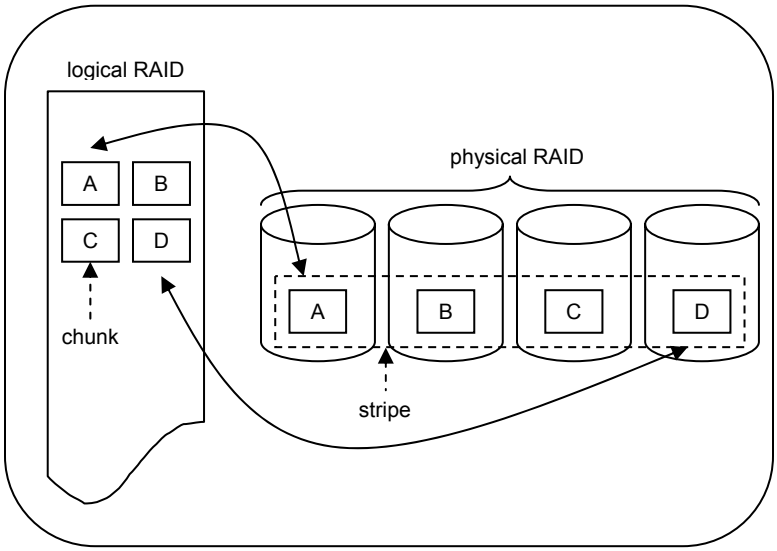

Figure 2.3: Logical RAID to physical disks

To provide reliability, RAID introduces data redundancy, so data loss from failed disk(s) can be recovered through the redundant data. The reliability of a disk is defined by its mean time to failure, or MTTF. The MTTF for an individual disk is rated by the disk manufacturer. The MTTF for a group of disks is the MTTF of a single disk in the array divided by the number of disks in the array [19]. For example, for 100 disks with a rated MTTF of 30,000 hours each, the MTTF for the entire group is 300 hours. An administrator would have to replace a disk every two weeks! RAID improved upon this by introducing a check disk into the array of disks that

contained redundant information. When a disk fails and is replaced, the check disk can be used to reconstruct the data on the disk. By introducing the check disk, the MTTF of an array of disks exceeds the useful lifetime of any single disk.

2.3 RAID Levels

A RAID personality (or RAID level) is defined by how logical sectors are mapped to the disks and the strategy to use and distribute redundant data. The original RAID paper introduced 5 different RAID levels, ranged from simple mirroring of data to complex distribution of redundant data. In the context of PARAID, it is worthwhile reviewing RAID levels 0, 1 and 5. RAID level 0 was not introduced in the seminal work on RAID but was formally introduced by the same group in 1994 [3].

One metric to characterize different RAID levels is by their overhead cost in storage capacity. The overhead cost is the capacity used to store redundant data divided by capacity usable to store the actual data. For example, mirrored data from one disk to another disk has a 100% overhead cost. For every one data disk there is one check disk for the mirrored data. Usable storage capacity is the total capacity of data disks and check disks that can be used to store data. For example, mirrored data has 50% usable storage capacity because for every two disks, only half of that, one disk, is used for data.

RAID level 0, also known as striped or non-redundant mode, maximizes disk parallelism to achieve maximum performance but offers no data redundancy. RAID 0 offers the minimal reliability and has the same reliability as a group of disks without any check disks. This level appends the capacity of the disks in the array so that the logical device has a total size equal to that of the disks added together. The striping pattern used by this level balances disk load in a uniform fashion for maximum disk parallelism. The overhead cost of this device is 0% and the storage capacity is 100%.

RAID level 1, also known as disk mirroring. This level mirrors all disks, which provides for data redundancy but the overhead cost is 100% and the storage capacity is 50%. Mirroring can actually improve latency for the RAID storage device over that of a single disk. Since the data is mirrored (or replicated), two mirrored disks can race to serve I/O requests and possibly reduce the access latency. It is common to use RAID 1 when only two disks are available and

reliability is desired. For many disks, RAID 1 can statistically lose 1/3 of disks without data loss.

RAID level 5, also known as block-interleaved distributed-parity, provides redundancy and performance to a large number of disks. This level is appealing to large data centers because of these features. This level requires three or more disks in the array to accommodate at least two data disks and always just one check disk. This level employs the use of bit parity information to reconstruct lost data, which can reliably recover from single-disk failures. For every stripe in the RAID device, one parity block is created and must be stored, effectively using one check disk for the device. This parity information is distributed uniformly across the disk array so that no one disk is constantly being accessed for parity information. If a disk fails then the sectors on the other disks are used to calculate the missing data. If the sector happens to be a parity sector, the parity is simply recalculated. For 5 disks, RAID 5 has an overhead cost of 25% and a useable storage capacity of 80%.

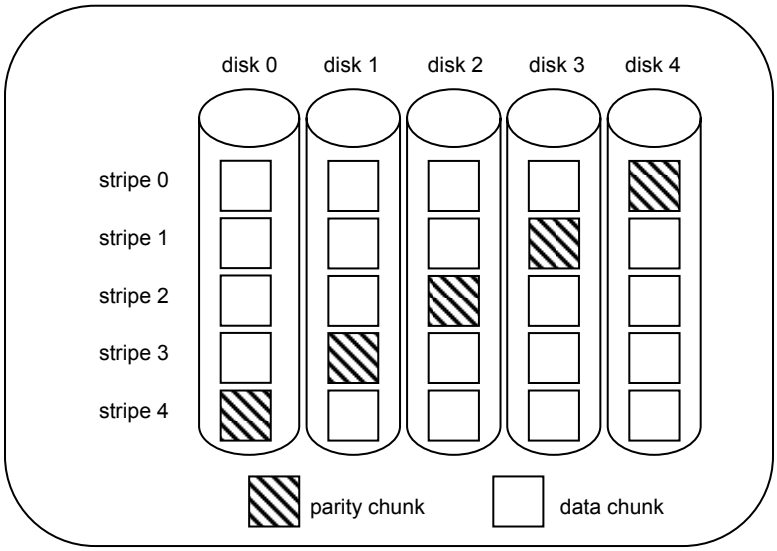

Figure 2.4: RAID level 5 striping pattern

Figure 2.4 shows the data layout of RAID 5 using a left-symmetric parity distribution. The parity chunk is calculated from the other chunks in a stripe. Because of this, a write request

in RAID 5 requires the data chunk and the parity chunk to be written. If the parity chunks were to be stored on one disk, then this disk would quickly become a hot spot in the array because every write request for any chunk would require a write on the parity check disk. RAID 5 solves this problem by having the parity chunks uniformly striped across the array.

2.4 Summary of RAIDs and Their Limitations

Conventional RAIDs improve the throughput of mass storage devices by combining disks into an array and accessing them in parallel. How those disks are used within the RAID device is defined by the RAID level that is used. Each RAID level has a different striping pattern that provides different benefits. RAIDs 0, 1, and 5 balance the load uniformly when possible, so that no single disk becomes the bottleneck.

Server-class RAIDs are not designed to save power for three major reasons. (1) Since RAID balances the load across all disks and requires all disks to be powered on to serve any request, all disks need to be power-switched as a single unit. Since servers tend to receive requests 24x7, the chance of turning off disks due to an idle server is significantly reduced [25]. (2) During the periods where the servers are lightly loaded, all drives are required to be powered to serve the requests, while perhaps a single powered drive is sufficient to meet the server demand. In other words, RAIDs over-provision resources during non-peak loads. (3) For reliability, server-class drives are not designed for frequent on-and-off power switches. Conventional power-saving approaches would be too aggressive on power switching and harm the hardware reliability of conventional RAIDs.

CHAPTER 3

POWER-AWARE RAID

3.1 Skewed Striping for Energy Savings

PARAID exploits over-provisioned and unused storage to improve energy efficiency and reliability. In particular, the use of skewed striping patterns is the key to achieving power savings without degrading performance. By assigning data blocks in a skewed pattern, the number of powered disks can vary according to the system demands.

3.1.1 The Power-Aware RAID and Skewed Striping

PARAID uses a skewed striping pattern to place the data blocks on the array of disks, so that the number of powered disks can vary according to the system demands. The disks in the array are first organized into hierarchically overlapping sets of disks, analogous to ***gears*** in automobiles. Figure 3.1 shows an example of data blocks stored on a four-disk array with two sets of disks, a two-disk set (gear 1) and a four-disk set (gear 2) containing the two-disk set. This organization enables the skewed striping pattern to create and shift gears to reduce disk power consumption. These gears allow for two operation modes. If PARAID operates with only gear 1, then disks 3 and 4 are not needed and can be powered off to conserve power.

Each gear in PARAID is capable of serving all I/O requests with different levels of performance, due to different levels of parallelism in disk accesses. Figure 3.2 shows an example of a four-block file being read from a PARAID device from either gear 1 (consists of two disks) or gear 2 (consists of four disks). Notice that some of the blocks are stored more than once. Disk 1 has a copy of data blocks 3 and 7, while disk 3 also has a copy of data blocks 3 and

7. Some blocks are replicated, so that a file can be accessed either through the two-disk set or the four-disk set.

The replicated blocks can be used to enhance reliability. However, not all blocks are replicated in PARAID. Blocks 5 and 6 are stored on disks 1 and 2, but not on disks 3 and 4. Non-replicated blocks will use other mechanisms for reliability, and will be discussed in Section 4.4.

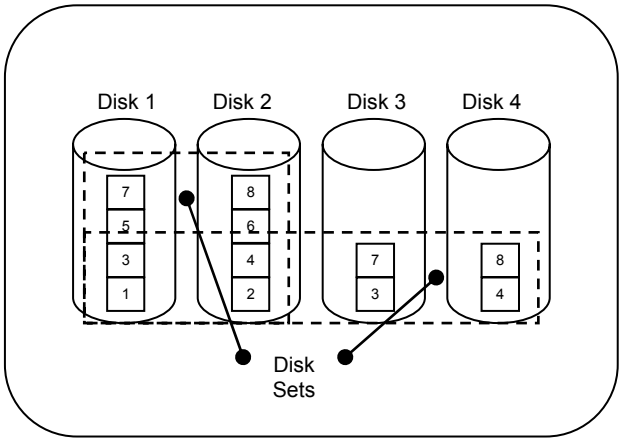

Figure 3.1: Skewed striping disk sets

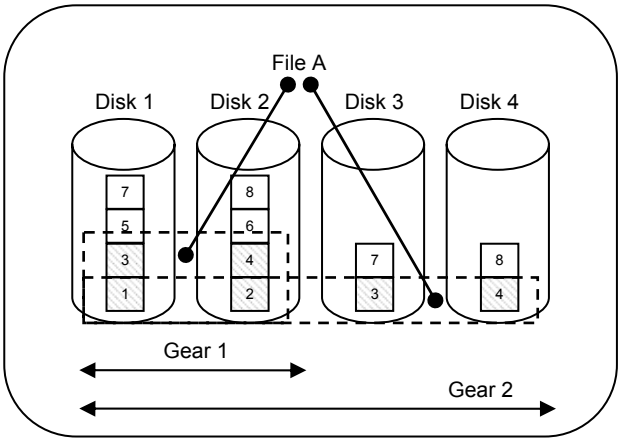

Figure 3.2: PARAID gears

3.1.2 Saving Power with Skewed Striping

To save power, PARAID needs to operate in a gear that will meet system demands, while powering off as many disks as possible. This can be thought of as using just enough disk parallelism for performance while minimizing the number of powered disks. This relationship between disk parallelism and the number of powered disks illustrates the inherent tension between performance and saving power. Maximizing disk parallelism will achieve maximum throughput while saving the least amount of power, whereas minimizing the number of powered disks will maximize power savings, but achieve the worst throughput.

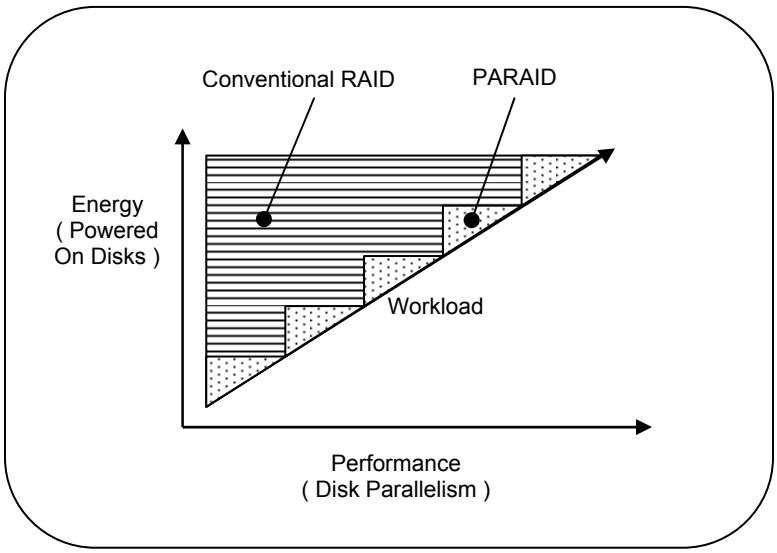

Figure 3.3: Workload approximation: conventional RAID versus PARAID

Figure 3.3 shows the relationship between performance and energy. As workload increases, energy and performance also increase. Due to gears, the PARAID device can power just enough disks to match the workload. Conventional RAIDs do not offer this power management granularity, and with RAIDs the number of powered disks provides only the maximal throughout. In an ideal implementation with zero overhead, PARAID can approximate the workload line arbitrarily close, as the number of gears increases. The closer to the workload line, the more power can be saved.

PARAID works because under a light system load, the performance perceived by end-users is not as sensitive to whether requests are served by two disks or four disks. These periods of light load are opportunities to power off disks and save power. Figure 3.4 shows how the number of disks can vary according to the system load. As the system experiences more load, the disk parallelism increases. Figure 3.5 also shows how a conventional RAID device cannot vary the number of power-on disks due its use of a uniform striping pattern.

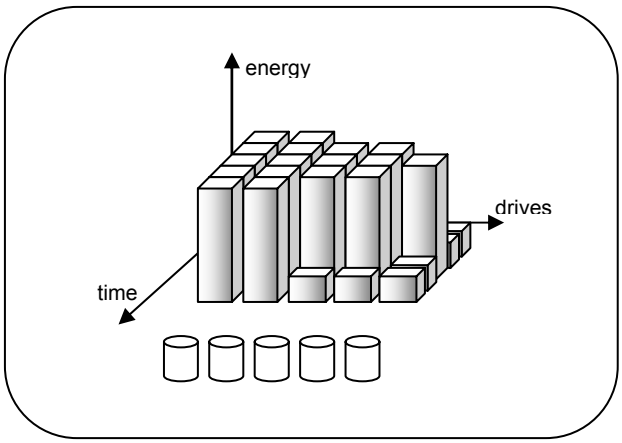

Figure 3.4: PARAID load balancing

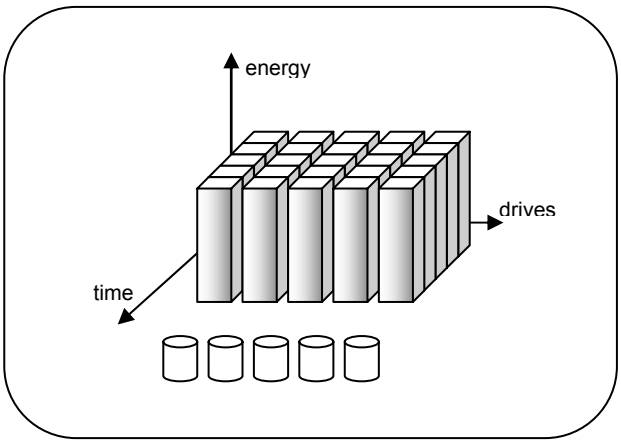

Figure 3.5: Conventional RAID load balancing

Gear shifting a PARAID device means switching from one gear to another. This can also be thought of as changing the view into the array of disks from one set of disks to another set of disks. Figure 3.6 shows how PARAID will gear shift to meet the system demand. PARAID knows when to gear-shift based on of the disk utilization for the disks within the gear. If the disks are utilized below a low-watermark threshold, then PARAID will shift into a lower gear. If the disks are utilized above a high-watermark threshold, then PARAID will shift into a higher gear.

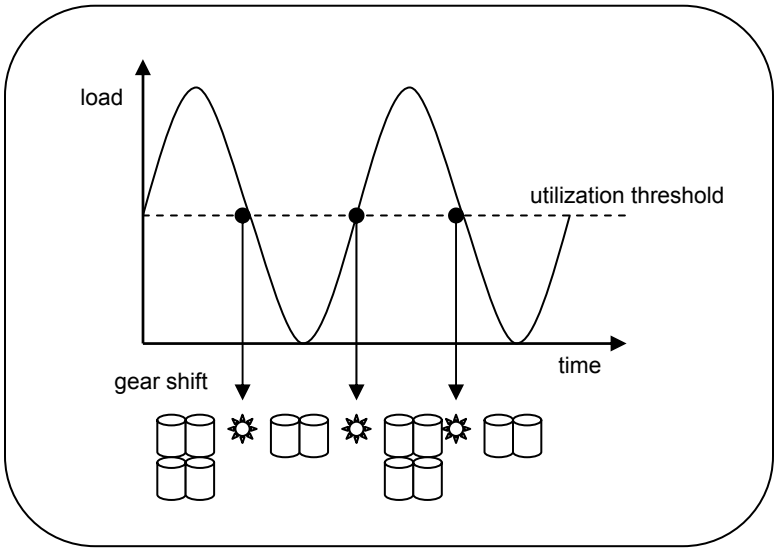

Figure 3.6: Gear shifting to adjust to load

The characteristics of the workload impact PARAIDs ability to save energy. Under workload that is consistently high, requiring peak performance or near peak performance all of the time, PARAID will not have sufficient opportunities to gear shift into lower gears and save energy. Under workload that is consistently low, trivial power-saving techniques can be used to save energy. Realistically, workload tends to fluctuate, driven by human behavior, between high and low periods over a period of time, for example over the course of a day or a week. People are more active during the day then at night. This cyclical fluctuation in workload is where PARAID excels by providing peak performance under high load and energy savings under low load.

3.2 Preserving Peak Performance

It is essential that performance be preserved. RAID is configured to meet the demands placed on the server under peak load. PARAID must be able to match that performance. This is accomplished in PARAID by operating in the highest gear when the load on the system demands it. Within each gear, load is balanced across the disks. This allows for maximum disk parallelism (maximum throughput performance) within the gears. When a PARAID device operates in the highest gear, all disks are being used in parallel to read and write data, providing a level of performance comparable to a same size RAID device.

A PARAID device has the potential to perform better in reading small files when operating in a low gear. This is due to the latency associated with reading from a disk. Disk latency is a factor of seek time and rotational latency. The average latency for a RAID device is bound by the latency of the slowest disk. When the number of disks is reduced in the array, it is more probable that the average latency is statistically reduced. Therefore, the average latency associated with reading small files is less on a PARAID device operating in a low gear, using fewer disks, compared to that of a conventional RAID device.

A PARAID device also has the potential to degrade throughput for large files when operating in a low gear due to the throughput achieved when reading files from multiple disks in parallel. When the number of disks is reduced, the throughput is reduced. Therefore, the throughput associated with reading large files is degraded on a PARAID device operating in a

low gear compared to that of a conventional RAID device. Reading large files is likely to trigger the PARAID device to perform a gear shift into a higher gear and increase disk parallelism.

3.3 Maintaining Reliability

PARAID must be able to tolerate disk failures to retain the reliability semantics offered by conventional RAID. Because PARAID relies on power cycling disks to save energy, PARAID must also address a new concern over reliability. Power cycling has an adverse affect on the MTTF of a disk, which is designed for an expected number of power cycles during its lifetime. For example, the disks used in this thesis have a 20,000 power cycle rating [5]. Every time a disk is power cycled, it comes closer to its eventual failure. PARAID must manage the power switches with care to mask this undesirable effect of power cycling.

3.3.1 Resiliency Approaches without Data Redundancy

PARAID manages the power cycling of the disks by inducing a bimodal distribution of busy and idle disks. The busier disks stay powered on and the more idle disks often stay off, leaving a set of middle-range disks that are power-cycled more frequently. PARAID can manage the power cycles for each disk by switching the gear-membership role of the disks according to their current number of power cycles.

For example, take a PARAID device that has six disks in its array and three gears. The first gear has disks 1 and 2, the second gear has disks 1 to 4, and the last gear has all six disks. Figure 3.7 shows that given this gear layout, disks 1 and 2 are the busier disks, and disks 5 and 6 the most idle, leaving disks 3 and 4 to be power-cycled more frequently. By keeping a count of the number of power cycles that each disk has performed, PARAID can decide which roles to exchange amongst the disks. After disks 3 and 4 have reached a certain power cycle threshold relative to other disks, they can be role exchanged with the disks not in gear 1, so that each disk is rate-limited in terms of increasing the number of power cycles, helping to prolong the MTTF of the PARAID device as a whole. Also, due to the lack of power switching, gear-1 disks can maintain the same level of reliability as before, since they are always powered.

In addition to inducing the bimodal distribution of busy and idle disks, PARAID can rate-limit the disks for the frequency of power cycles. By rationing power cycles, PARAID can

operate with an eye to targeted life expectancy. For example, if the disks have a five-year life expectancy (due to the system upgrade policy), and the disks are expected to tolerate 20,000 cycles, then each disk in the array cannot be power-cycled more than 4,000 times a year (333 times a month or 76 times a week). Once any of the disks have reached the rationed amount of power cycles for a determined period, a PARAID device can operate in a mode where no power cycles will take place. Once the period expires and enters a new rationing period, the disks can be power-cycled again to conserve energy, according to system load.

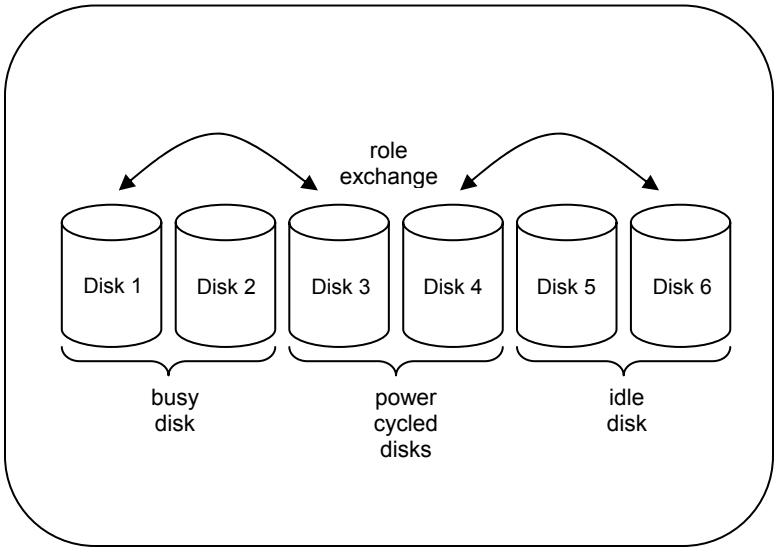

Figure 3.7: Disk role exchange

By creating a bimodal distribution of busy-to-idle disks, exchanging the roles of the disks within the array and rationing the number of power cycles for each disk, the life expectancy of disks due to frequent power cycling can be controlled effectively.

Furthermore, as PARAID is filled up over time, and the unused storage available for skewed striping diminishes, the skew of the striping pattern will be adjusted to approach a uniform distribution. Since the uniform distribution of blocks decreases the opportunities to power off disks, the frequency of power cycles will be automatically dampened automatically as PARAID ages.

3.3.2 Resiliency Approaches with Data Redundancy

The current PARAID design based on RAID 0 can tolerate a single-disk failure outside of the first gear, because of the way the data blocks are replicated across the disk array in a skewed manner. Also, each gear contains the full content of the PARAID content. If a disk fails, the PARAID device can choose a gear that does not use the failed disk to continue operating. Once the failed disk has been replaced, the new disk can be synchronized with other disks. In fact, PARAID can handle multiple disk failures as long as the gear-1 disks in PARAID are still operational. The PARAID device cannot guarantee a full recovery from a single-disk failure, which is when all sets of disks contain a common disk that fails. In this case, PARAID might be able to recover a subset of data blocks once the failed disk has been replaced, but the PARAID device would certainly need to stop operating until that happens.

The MTTF for a RAID-0 device is the MTTF of a single disk divided by the number of disks in the array [3]. The MTTF for PARAID is the MTTF of a single disk divided by the number of disks in the first gear. RAID 0 is the most comparable level to PARAID because PARAID—like RAID 0—balances load within each gear uniformly. Given a four disk array, each with a rated MTTF of 1,200,000 hours [16], a RAID-0 device using this array has an MTTF of 300,000 hours or 34 years. In comparison, a PARAID device using the same array of disks with 2 disks in the first gear would have an MTTF of 600,000 hours or 68 years.

3.4 Summary

By using a skewed striping pattern that creates sets of disks within the array of disks, the number of powered disks can vary according to system load. These sets of disks can be thought of as gears that can be shifted, so that just enough disks can be powered to match the current workload on the system. Opportunities to save power arise during light or moderate loads when low gears can be used, allowing unused disks to be powered off. This cannot be readily achieved by power switching conventional RAIDs.

Balancing the load within each gear allows for maximum disk parallelism for that gear. When the load on the system demands peak performance, high gears are used to match the performance of conventional RAID. Through the use of gear-membership role exchanges among disks and rationing power cycles, PARAID can be configured to adhere to specified life

expectancy. Overall, PARAID reduces the amount of energy consumed in mass storage while retaining similar performance and reliability.

CHAPTER 4

DETAILED PARAID DESIGN

The design of PARAID has five major components: the PARAID Personality, the PARAID Monitor, the PARAID Gear-Shifting Logic, the PARAID Reliability Manager, and the PARAID User Administration Tool.

Figure 4.1 shows various PARAID components, their logical associations, and their locations in a system. The PARAID Personality, Monitor, Gear-Shifting Logic, and Reliability Manager all reside within the kernel space of the operating system while the User Administration Tool resides in the user space. Most of the components reside within the kernel space, so they can communicate with one another without crossing the kernel/user space boundary and associate overhead. Through these five components, PARAID builds power-aware RAID devices, reduces power consumption, maintains performance, and manages reliability.

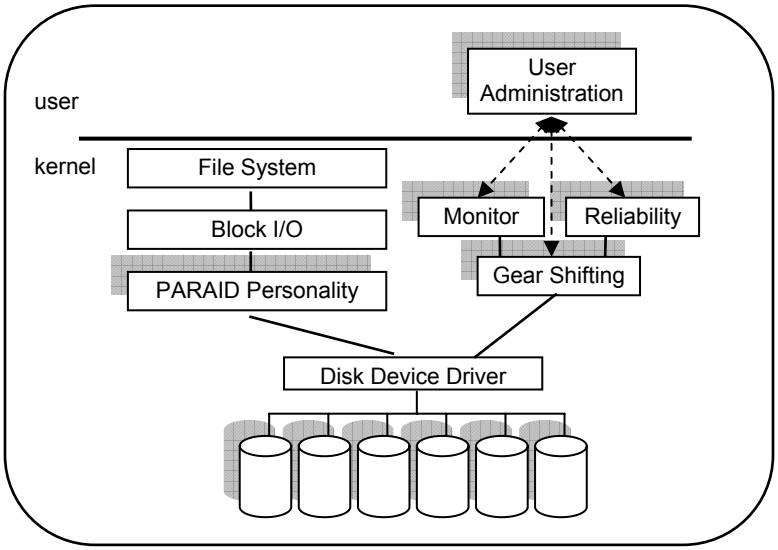

Figure 4.1: PARAID logical design

4.1 Building Power-Aware RAID Devices

The PARAID Personality is responsible for device creation, state management, and handling I/O requests.

4.1.1 Creating a PARAID Device

When a PARAID device is first created, the PARAID Personality must determine the size of the PARAID device, create the data structures for tracking the current states of the device, and create the PARAID super block. To do this, the system administrator must specify the disks to be included in the PARAID device and the desired number of gears and pass this information into the PARAID Personality upon creation. This information can be specified via a configuration file or command line arguments.

The storage capacity needed by a PARAID device can be computed in two ways: bottom-up or top-down. The bottom-up approach starts by assuming that everything can be stored within the first gear (the minimum number of disks that can hold the data) then incrementally builds additional gears to the remaining disks. This approach also allows incremental addition of disks to PARAID. Since gear-2 disks might not be big enough to hold the information for higher gears, this approach may require some backtracking to readjust the maximum number of the first gear. The top-down approach starts with uniform striping of the actual data across all disks (the highest gear), and incrementally creates lower PARAID gears via replicating some of the data as unused space permits. Currently, the PARAID prototype uses the bottom-up approach. The top-down approach is left as future work.

Based on this information, PARAID calculates the size of the PARAID device. The PARAID device capacity can be calculated with the algorithm listed in Table 4.1. The algorithm assumes that the disks in the array have the same size.

The number of disks in the first gear and the size of each disk will determine the size of a PARAID device. Once the capacity of the PARAID device has been determined, the space required for each gear (boundary) is allocated for each disk. These static boundaries on each disk for each gear are used to ease computing an offset when accessing data blocks. These boundaries can also change dynamically to provide more flexibility.

Table 4.1: PARAID capacity algorithm

PARAID Capacity Algorithm

n = number of gears
$nDisk_{gi}$ = number of disks in gear i
C_{g1} = storage capacity of gear 1, in disk blocks
C_{disk} = storage capacity of a disk

while ($\sum_{i=2}^{n} \frac{C_{g_1}}{nDisk_{g_i}} > C_{disk}$) {
  C_{gi} -= 1 disk block
}

Figure 4.2 also depicts how the capacity algorithm in Table 4.1 determines the total capacity for the device, considering the disks and gears. The total capacity for the PARAID device is the storage space on disks 1 and 2 that makes up the first gear. Notice that there is some unused space on disks 1 and 2. This is due to the PARAID Personality reducing the total capacity of the PARAID device so that all of the replicated data blocks for each gear can fit on all disks.

Once the size of the PARAID device is determined, the necessary data structures used to maintain the current state of the PARAID device are created and initialized. Each of the five PARAID components must maintain information about the PARAID device and the operation of the device. This information is listed with the discussion of each component.

Lastly, the PARAID device super block must be created and persisted. The super block for the PARAID device is 512 bytes or one sector and is written to the first sector of each disk. By persisting the super block, the PARAID device can be stopped and started without having to be created each time. Once the size of the PARAID device has been calculated, the data structures necessary to maintain the state of the device have been created, and the super block has been persisted, the PARAID device is ready to handle block I/O.

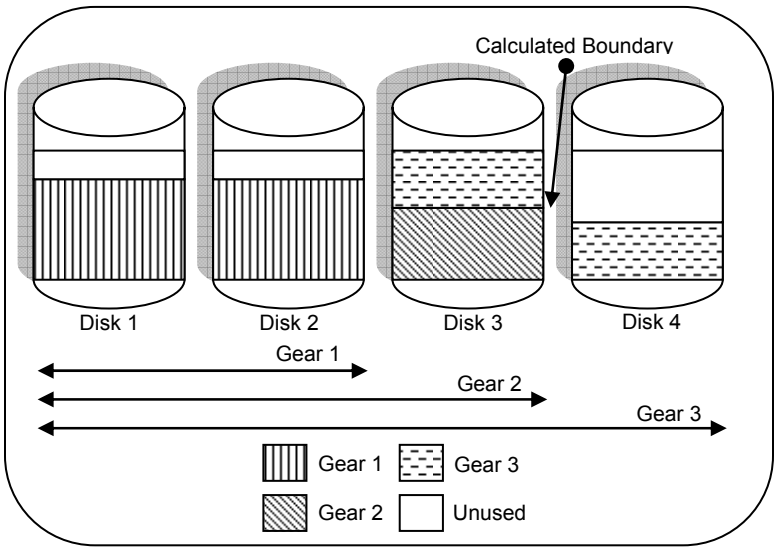

Figure 4.2: Placement of data blocks on disks by skewed striping

4.1.2 Handling Block I/O

It is the responsibility of the PARAID Personality to handle the requests sent to the PARAID device from the file system. The PARAID Personality must interpret an incoming request, map the block to the appropriate disk and sectors according to the striping pattern, and issue the block I/O to access the block.

The file system considers the PARAID device to be one contiguous block of storage to which it may issue read and write requests. The PARAID Personality is located between the file system and the disk. The PARAID Personality interprets the read/write requests from the file system so that it may place the blocks on the disks in the array according to the striping pattern. This is where the PARAID Personality writes the blocks to the disks in a skewed fashion so that the gears within the PARAID device can be created.

The formulas presented in Table 4.2 are used in the PARAID Personality to map the logical sector of the block I/O request sent from the file system to the disk. The mapping formulas map a logical sector to a disk and a sector on that disk device. Formula 1 in Table 4.2 is used to map a logical sector to a disk. Formula 2 is used to map a logical chunk to the chunk on a disk. Formula 3 is used to map a chunk on a disk to a sector on a disk.

Table 4.2: Skewed striping mapping formulas

1. PARAID Logical Sector to Disk Formula
Terms logical chunk number = logical sector number / sectors per chunk gi = number of disks in gear i Formula disk number = logical chunk number % gi
2. PARAID Logical Chunk to Disk Chunk Formula
Formula chunk number on disk i = logical chunk number / gi
3. PARAID Disk Chunk to Disk Sector Formula
Terms n = number of sectors per chunk f = sector offset within chunk = logical sector number % n r = gear boundary offset on disk Formula sector number = (chunk number on disk i * n) + f + r

The following example shows how the mapping function works in the PARAID device and is depicted in Figure 4.3. Take for example, a PARAID device with four disks and two gears. Gear 1 has two disks, and gear 2 has all four. The sector size is 512 bytes; the block size is 1 Kbyte; and the chunk size is 4 Kbytes. Currently, the second gear is active. Using this information, the logical sector 2008 can map the physical disk within the array using formula 1 in Table 4.2. In this formula the logical chunk number equals 251 (eight sectors in a 4-Kbyte chunk). Therefore, the logical disk number is 3 (251 % 4). The chunk on the disk, also the stripe in the gear, can be computed using formula 2. Since the logical chunk number is 251, the chunk number on disk 3 is 62 (251 / 4). Now the sector number on the disk can be computed using formula 3. Because logical sector 2008 is the first sector in the chunk, f is zero. The factor f represents the offset within the chunk for the sector; the first sector in the chunk has no offset. The factor r is zero because the storage space for gear 2 on disk 3 starts at the first sector for data storage. Therefore, the sector is 496 ((62 * 8) + 0 + 0).

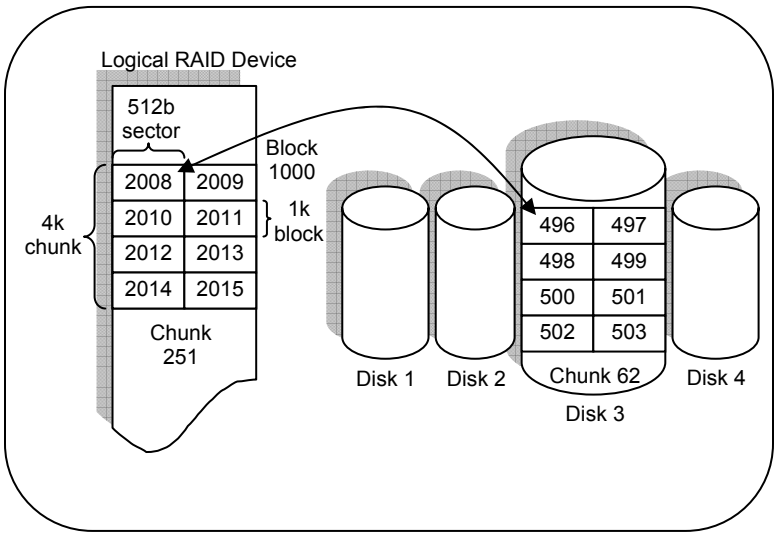

Figure 4.3: Logical to physical disk mapping

For data to be retrievable from any gear, some blocks need to be replicated, which means that some blocks need to be written more than once. The case where an update needs to apply to a disk that is not currently powered will be discussed in Section 4.2.2. The PARAID Personality determines whether a block needs to be updated to multiple disks by cycling through each gear for a block's potential storage locations. However, since gears have hierarchical overlapping sets of disks, an update to gear 2 disks can exclude gear 1 disks, which are already updated.

For example, consider a PARAID device that has four disks and two gears. The first gear consists of disks 0 and 1, and the second gear consists of all four. When a write request arrives, the PARAID Personality will use the logical sector of the request to find the disk location within gear 1, and then do the same for gear 2. For gear 1, the update will always result in a disk write to either disk 0 or disk 1. However, since gear 2 consists of all four disks, should the update location reside in disk 0 or disk 1, the update is not necessary, since the same update has been applied during the gear 1 iteration. Should the update location reside on disks 2 or 3, the update will be written in both places.

The PARAID Personality must maximize disk parallelism when reading blocks from the disks. Maximized disk parallelism is achieved by using the mapping formulas in Table 4.2 that calculate the disk and sector to read the block from. Disk parallelism is maximized on reads because the load is balanced across all disks in the gear when the blocks are written. This is

most important when PARAID is operating in the highest gear so that peak performance can be provided under peak workload.

Finally, the PARAID personality must track the number of disks and gears, and also the number of disks within each gear. This information is used to help map the blocks of data from the logical view to the physical disks. The logical chunk is obtained with each I/O request made to the device while the disk, chunk, and sector are computed from this information with each request.

4.2 Gear Shifting

The PARAID Gear-Shifting Logic is responsible for gear shifting PARAID, which includes power cycling the disks, and disk synchronization.

4.2.1 Power Cycling

The PARAID Gear-Shifting Logic performs gear shifts between gears. The PARAID Monitor helps the gear-shifting logic by advising when the gears should be shifted.

To gear shift between gears, the PARAID Gear-Shifting Logic must know the current gear and the target adjacent gear. The PARAID Gear-Shifting Logic powers off appropriate disks when shifting to a lower gear, and powers on necessary disks when shifting to a higher gear.

When powering on disks, stale disk content has to be resynchronized. Once the synchronization is complete, the current gear can be changed to the new gear. Powering off disks is immediate, since no disk synchronization is necessary.

4.2.2 Disk Synchronization

When disks are powered off, no requests are sent to those disks. As soon as a powered off disk misses a write request, the disk no longer contains the most up-to-date data for all data blocks. The disk needs to synchronize the stale data at the time when it is powered on or right before the stale information is accessed. Full synchronization requires that all stale data be updated with current data. Depending on the total size of the stale data, this process could take a long time. The on-demand approach only updates the stale data when it is accessed. The on-

demand approach allows the gear shift to take place much more quickly, but the full synchronization approach provides better data consistency.

To be able to synchronize a disk, outstanding write requests to powered off disks are captured by the PARAID Gear-Shifting Logic. In the case of full synchronization, when an powered off disk is switched to a powered on state, the PARAID Gear-Shifting Logic reissues a list of outstanding write requests to the disk that is to be synchronized. Sometimes this process involves rereading the data from a committed copy before reissuing the write data.

In the case of on-demand synchronization, the PARAID Personality uses a dirty-block list. If the block being accessed is dirty and not cached, PARAID will retrieve the block from the first gear and return it to the requestor. PARAID will then write that block to the disk(s) that has stale data, effectively piggybacking the synchronization step at the access time, and skipping the rereading step at times.

To capture the write I/O requests, the PARAID Gear-Shifting Logic needs to make sure that the first-gear disks are always up-to-date. The Gear-Shifting Logic also needs to track the stale disk locations for synchronizations. This list of dirty blocks is not only kept in memory for fast access during on-demand synchronization but is also persisted to secondary storage in case of system failure.

A failed disk can stop the gear-shifting process. Disks can also fail in the middle of the synchronizations. However, the list of outstanding writes is maintained throughout the disk failure and recovery process. Once the failed disk recovers, the synchronization can continue from where it left off.

Whether to use on-demand or full synchronization is configurable. On-demand synchronization will allow PARAID to adjust to frequent changing workload. This also means tracking additional writes while the disks are not synchronized. The full-synchronization approach may be preferable if there are few gear shifts and the workload is dominated by reads, effectively keeping the number of blocks to be synchronized small. The full synchronization method is also available for manual maintenance of the PARAID device. For example, an administrator would need to have a consistent system state before pulling out a hard disk.

4.3 Saving Energy

The PARAID Monitor monitors the workload placed on the PARAID device, determines when the gears should be shifted based on the workload, and makes requests to the PARAID Gear-Shifting Logic to switch gears.

4.3.1 Shifting to a Higher Gear

To decide whether the PARAID device should gear shift into a higher gear, the PARAID Monitor needs to know the load placed on the current gear. The gear utilization threshold is the metric used to determine when to perform a gear shift. Since the data and requests are uniformly spread within a gear, the percent utilization of the gear can be extracted by the percent utilization of individual disks within a gear, in number of requests per time interval. A system administrator can set the gear utilization threshold to specify how utilized a gear should be before switching to a higher gear (e.g. 80%).

To know the demand trend on the system, the PARAID Monitor keeps a moving average of utilization over the past 60 seconds for each disk. For every second, each disk is checked to see if any access has occurred. If an access has occurred, the disk is marked as active for that second. Table 4.3 lists the formula used to calculate the moving average.

Table 4.3: Moving average formula

PARAID Monitor Disk Moving Average Formula
Terms t = time interval for moving average in seconds u = number of seconds disk utilized over time interval t Formula Moving average = u / t

If the moving average is above the threshold for all disks in the active gear then the PARAID Monitor will make a request to the PARAID Gear-Shifting Logic to switch to the next higher gear. Figure 4.4 shows a gear shift from a low gear to a high gear because the moving utilization average rose above the threshold.

Switching to a higher gear is aggressive, so that the PARAID device can respond quickly to a sharp increase in workload. As soon as the gear utilization rises above the threshold, an upward gear shift will be performed. Figure 4.5 shows how PARAID responds to fluctuating workload with sharp increases by performing multiple upward gear shifts.

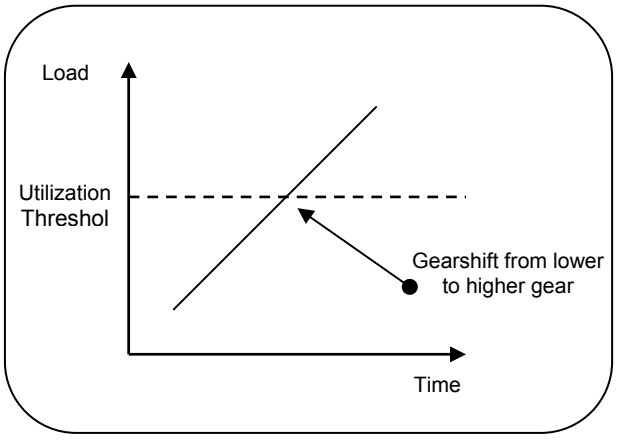

Figure 4.4: Upward workload trend

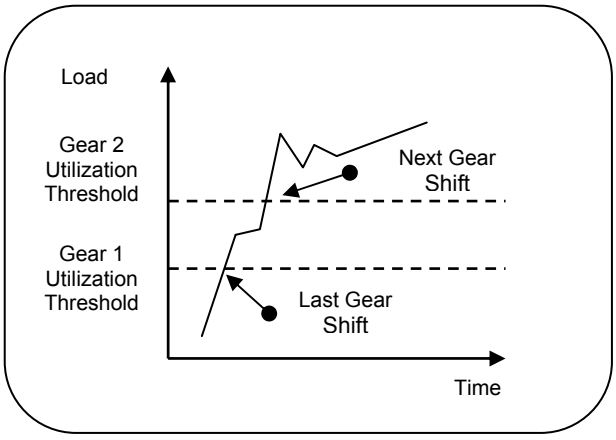

Figure 4.5: Steep upward workload trend

4.3.2 Shifting to a Lower Gear

To decide whether the PARAID device should downshift, the PARAID Monitor not only needs to know the current gear utilization, but also the utilization trends. Downshifting gears needs to be done conservatively, so that wild swings in system activities will not (1) mislead the PARAID device into a gear that cannot handle the requests (Figure 4.6) and (2) cause rapid oscillating of gear switching.

This is prevented by keeping moving averages of the gear utilizations at three different time intervals: 10, 60, and 300 seconds. The moving averages for each time interval are computed with the formula in Table 4.3. Monotonically decreasing moving averages at 300

seconds, 60 seconds, and 10 seconds identifies a steady decreasing utilization trend. Having identified this trend, the PARAID Monitor then checks if the next lower gear with fewer disks can handle the current workload based on a threshold (computed as a function of the number of disks available at the lower gear). If both conditions are met, the PARAID Gear-Shifting Logic switches to the next lower gear.

Figure 4.6 shows that the PARAID Monitor performs a downshift in gears when a significant downward trend in the gear utilization is detected and when the current utilization is below a certain threshold. Notice this approach is more stable and resilient to wild swings in workload.

To be able to identify the utilization trends and make the decision to gear shift, the PARAID Monitor must also maintain the threshold for each gear. In addition, the moving averages are continuously computed by the Monitor and checked against the thresholds for downshifting opportunities.

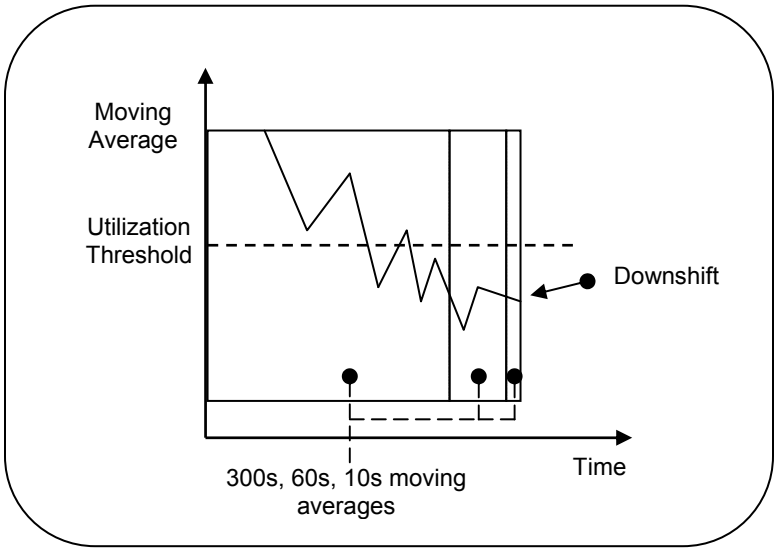

Figure 4.6: Downward workload trend

4.4 Managing Reliability

The PARAID Reliability Manager is responsible for managing the life expectancy of each disk and recovering a PARAID device upon disk failure. The goal of the PARAID Reliability Manager is to match a level of reliability provided by conventional RAIDs.

4.4.1 Managing Life Expectancy

Power cycles reduce the life expectancy of disks. The disks used in the evaluation of PARAID have a rating of 20,000 as the expected number of power cycles during the entire disk life. Thus, it is crucial to use power cycles sparingly and effectively, so that they are unlikely to become the cause of disk failures.

To achieve this, the PARAID Reliability Manager rations power cycles over the disks. The system administrator needs to tell PARAID Reliability Manager the power-cycle ratings of disks, the time interval to enforce the rationing, and the desired life expectancy of the disks.

Table 4.4: Power cycle rationing formula

Power Cycle Rationing Formula
Terms p = number of power cycles for the disk i = number of rationing interval(s) per year l = life expectancy in year(s) Formula number of power cycles per rationing interval = p / (i * l)

The formula in Table 4.4 trivially determines the number of disk power cycles that can be rationed within a rationing time frame, given the rationing intervals, desired life expectancy, and the power cycle rating. For example, a disk with a 20,000 power-cycle rating using a weekly rationing time interval and a five-year life expectancy can be power-cycled 76 times a week (20,000 / (52 * 5)).

Each gear-shifting decision is checked with the PARAID Reliability Manager to make sure that the power cycle is allowed according to the rationing scheme. Of course, an administrator can override and force gear shifting to occur for maintenance purposes.

Recall that due to the use of bimodal distributions, busier disks and more idle disks are power cycled less frequently, leaving the middle-range disks to be power cycled more often. When those disks perform significantly more power cycles compared to other peers (based on a threshold), the Reliability Manager will role exchange this disk with a disk that has performed fewer power cycles. By exchanging the roles of the disks, the number of power cycles can be evenly distributed across the disks, and the overall life expectancy of the PARAID device can be significantly prolonged.

4.4.2 Recovery from Disk Failure

One of the benefits of using a skewed striping pattern is the expanded ability of the PARAID device to recover from disk failures. Take RAID 0 as a basis for comparison. RAID 0 will suffer data loss if any of the disks fails. However, PARAID based on the RAID 0 implementation (or PARAID 0) can survive disk failures due to replicated data blocks, as long as the failed disk is not within the first gear.

When a disk fails, the PARAID-0 Reliability Manager notifies the system administrator through log files, real-time status outputs, or alert email messages. Once the administrator has replaced the failed disk, the PARAID-0 Reliability Manager instructs the PARAID 0 Gear-Shifting Logic to synchronize the new disk with the first-gear disks. No specialized reliability mechanisms are required.

If failure involves one of the first-gear disks, the PARAID-0 device will fail. Then, PARAID 0 needs to rely on conventional recovery techniques such as backups. However, a PARAID based on RAID 5 will be able to use the additional parity information to survive such failures.

The time it takes for a disk to recover depends on the amount of data needed to be transferred to the new disk. If a failed disk participates in only the highest gear, the data required for recovery will be much less than that of a failed disk that participates in many different gears. This is an inherent property of the skewed striping pattern, as shown in Figure 4.2.

For example, given a PARAID device with four disks of 30 Gbytes each and two gears of two and four disks respectively, the amount of time for disk recovery can be conservatively determined by the time to read data from gear 1 and write it to the new disk. The calculated capacity for this device using the PARAID capacity algorithm would be 60 Gbytes. Disk 3 and 4 require 15 Gbytes of storage each. This means that 15 Gbytes of data need to be synchronized if either disk 3 or 4 fails.

To be able to manage the reliability, the PARAID Reliability Manager needs to know the life expectancy and the rationing time interval. Based on this information the Reliability Manager can calculate the allowable power cycles per rationing interval for each disk and determine the best disk to role exchange when needed.

4.5 User Administration

The PARAID User Administration Tool, pdadm, short for PARAID administration, provides the knobs and buttons that allow the administrator to configure the PARAID device for desired system characteristics.

4.5.1 Controls

The knobs and buttons provided to the PARAID administrator can be categorized by each component. Table 4.5 shows the entire set of controls, categorized by component, that are available to the system administrator.

For the PARAID Gear-Shifting Logic, pdadm allows the PARAID administrator to request to jump into any preconfigured gear. Of course, changing into the wrong gear at the wrong time could negatively impact the performance of the storage device. If the PARAID device were to be switched from a high gear to a low gear while the system was experiencing peak load, the device would not be able to serve the data fast enough, ultimately overloading the system. The User Administration Tool will notify the administrator if the requested gear does not have the throughput to handle the current load, so that this can be prevented.

The User Administration Tool also allows the request to synchronize a disk, so an administrator can bring all disks into a consistent state, perhaps because they have been powered off for a very long time.

For the PARAID Monitor, pdadm allows changing the minimum time interval between gear shifts, so PARAID can react more quickly to downward utilization trends. This tool also allows the administrator to set the upper and lower utilization thresholds for each gear. By having a higher upper threshold, more power can be saved because PARAID will spend more time in lower gears, with fewer powered disks. By having a lower upper threshold, the PARAID device might perform better, having more disks powered on longer. By having a higher down shifting threshold, the PARAID device will react more quickly to downward sloping trends and shift into lower gears more quickly. While having a lower downshifting threshold, the PARAID device will wait longer to shift into lower gears, taking a very significant downward utilization trend to switch into a lower gear. Lastly, this tool allows the administrator to turn the Monitor on and off. It might be the case that the administrator does not want the PARAID Monitor to run at all and the administrator will shift the gears manually.

Table 4.5: PARAID User Administration controls

PARAID User Administration Controls	
PARAID Disk Manager	
Gear Shift Button	Manually shift gears
Synchronize Button	Manually synchronize the disks
PARAID Monitor	
Time Interval Knob	Adjust the mandatory time between gear shifts
Gear Threshold Knob	Adjust the upper and lower utilization thresholds for each gears.
On/Off Button	Turn the monitor on or off
PARAID Controller	
Rationing Interval Knob	Adjust the time interval that rationing is enforced
Disk Life Expectancy Knob	Adjust the minimum guaranteed life expectancy for the disks
Power Cycles/Disks Knob	Adjust the number of power cycles for a disk
Default Operating Gear Knob	Set the operating gear when the PARAID device has used its rationed amount of power cycles for an rationing interval

For the PARAID Reliability Manager, pdadm allows the PARAID administrator to set the rationing interval, the life expectancy for PARAID, and the power-cycle rating for a disk. The administrator has the ability to tweak the reliability versus power savings through these PARAID Reliability Manager knobs. If the load on the system fluctuates frequently, and the desired goal is to maximize power savings, then the life expectancy of the disks will decrease. If the goal is to maximize the life expectancy, then the number of times a disk can be power cycled per interval will be reduced, potentially missing out on opportunities to save power.

The administrator can set the rationing of power cycles to be weighted seasonally. For example, if more workload on average is experienced in the winter season, the administrator can set this season to be weighted more heavily than the other seasons. This would increase the number of rationed power cycles for the rationing intervals over that season and lower the number of rationed power cycles for the other seasons.

If the rationed number of power cycles is not used within a rationing interval for a particular disk, the remaining power cycles can be accumulated into the next rationing interval. Conceivably, by the end of the life expectancy, the disks could have many power cycles remaining. As time goes by over the lifetime of the disks, the remaining power cycles will have been continually accumulated into the next rationing interval. Using this information, the administrator can decide to keep the current power saving policy configured for the device and enjoy the extended PARAID lifespan of the disks or set a more aggressive power saving policy and enjoy more power savings

CHAPTER 5
PARAID IMPLEMENTATION

The PARAID prototype was built on Linux using the 2.6.5 kernel, which was the latest release at the time of implementation. Linux was chosen due to its open source and it contains a software RAID module.

The PARAID Personality, PARAID Monitor, PARAID Reliability Manager, and the PARAID Gear-Shifting Logic are built in the kernel space. The PARAID User Administration Tool is built in the user space. The PARAID Personality is built as a part of the Linux software RAID module. The PARAID Gear-Shifting Logic, Monitor, and Reliability Manager are also implemented as Linux kernel modules, but are not part of the Linux software RAID. These kernel modules are loaded when the PARAID Personality is loaded. The module functions within the Gear-Shifting Manager, Monitor, and Reliability modules are called from the PARAID Personality logic.

5.1 PARAID Personality Implementation

The PARAID Personality is implemented as a part of the Linux software RAID, which is a kernel module. The Linux software RAID is implemented as the md (multiple device) device driver module, which builds RAIDs from multiple disks. An md device is categorized as a block device within the Linux operating system; the resulting software RAIDs are, therefore, block devices.

Each of the conventional RAID personalities are implemented as modules that are loaded and used by the md device driver as needed. The md device driver knows different types of multiple devices that it can create, along with which associated modules to load. When a

creation request is made to an md device driver, the md device driver associates an array of disks, loads the appropriate module, and associates the two. The md device driver creates a data structure of virtual function pointers for each new md device. When a RAID module is loaded, the RAID module will initialize the corresponding virtual function pointers to its own functions, so that the md device driver can call specific functions defined by the RAID module. This concept is depicted in Figure 5.1.

Most important is the RAID Personality's implementation of the block request I/O function, which is assigned to and used by the block queue for a new RAID device. This means that all I/O activities will use a RAID Personality's specific implementation, which can enforce its own striping pattern by mapping the logical block location to the physical block location. Figure 5.2 shows how the request function of each RAID Personality takes in the block I/O requests and maps them to the physical disks.

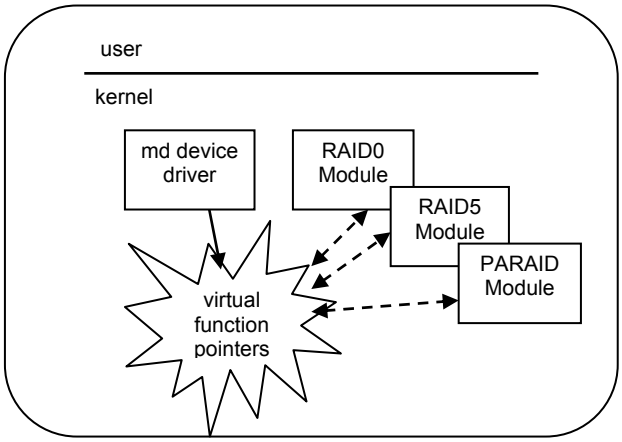

Figure 5.1: MD Device Driver

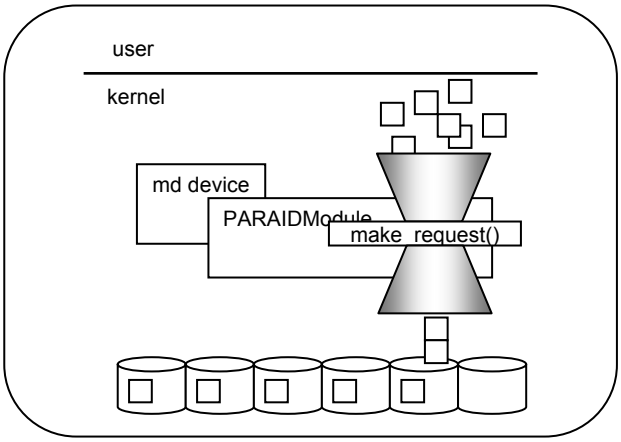

Figure 5.2: Make Request Function

For the PARAID Personality implementation, changes were made to the md device driver to become aware of PARAID and its modules for its creation. The PARAID Personality module contains the I/O request function, which maps the logical block locations on the md device to the physical block locations on the disks in a skewed fashion.

During PARAID initialization, memory is allocated for several data structures. Most important is the data structure that tracks the current state of the PARAID device. This data structure tracks the gears in the device, the active gear, the size of the array, the participating disks, and the size of each disk, among other things. This state data structure also allows different PARAID modules to communicate, in addition to using external module functions. For example, when the PARAID Gear-Shifting Logic switches gears, it does so by changing the active gear data element on the state data structure to the new gear. The I/O request function picks up the gear change by examining the active gear data element before sending off another block I/O.

Also during initialization, the PARAID Personality module starts a PARAID Linux kernel daemon. This daemon provides the heartbeat to the PARAID device, but most important, the daemon calls the PARAID Monitor at regular intervals, so that the Monitor can decide when to gear shift.

5.2 PARAID Gear-Shifting Logic Implementation

The major responsibilities of the PARAID Gear-Shifting Logic include power cycling the disks in a gear-shifting fashion and synchronizing disk content. The Gear-Shifting Manager exposes an external module function that allows the Monitor to tell the Gear-Shifting Manager when a gear shift is needed and what gear to shift into. The PARAID Gear-Shifting Logic is implemented as a Linux kernel module that is loaded whenever the md device driver loads the PARAID Personality.

The Gear-Shifting Manager controls the power status of disks through the disk device I/O control interface. In particular, PARAID currently uses SCSI devices, for which the device driver provides built-in I/O control commands for starting and stopping SCSI disks.
To synchronize the content of a powered-off disk before bringing it back into operation, the Gear-Shifting Manager keeps a list of the outstanding blocks that need to be updated for each disk. Maintaining these lists is not CPU-intensive, and they are updated whenever a write request is made to a powered off disk. For the current implementation, the Gear-Shifting Manager performs a full synchronization after bringing back a powered off disk, by iterating through the list for that disk and reissuing all outstanding writes. On-demand synchronization is

currently not implemented. For each block that needs to be synchronized, the Gear-Shifting Manager will first read in the data block from a first-gear disk, and then write the block to the disk being brought back online. Once the synchronization is complete, the Gear-Shifting Manager switches to the new gear by changing the active gear on the PARAID state data structure.

5.3 PARAID Monitor Implementation

The major responsibility of the PARAID Monitor is to decide when a gear shift should take place. To make this decision, the Monitor tracks disk activity, so that the moving averages of disk utilization can be calculated. The PARAID daemon, started by the PARAID Personality module, calls an external module function on the PARAID Monitor module every second. This allows the Monitor to track the busyness of a disk for the past 10-, 60-, and 300-second periods. Basically, if a disk has had any read or write activity since the last second, then the disk is considered busy for that second, which is recorded in a data structure that keeps the disk utilization for the 10-, 60-, and-300 second time intervals. To be specific, the Monitor keeps a counter to track read and write requests made to each disk. If the counts have increased since the last monitoring interval, the disk is considered to have been utilized. The PARAID Monitor is implemented as a Linux kernel module that is loaded whenever the md device driver loads the PARAID Personality.

As well as keeping track of the current activity on the disks, the Monitor calculates the moving averages every second and checks to see if there is an opportunity to shift gears. This is done as specified in the design of the Monitor by checking the moving averages to the upper and lower thresholds of the device. Currently, the Monitor checks only one threshold for each gear. When a gear shift is needed, the Monitor contacts the PARAID Gear-Shifting Manager to perform the gear shift.

5.4 PARAID Reliability Manager Implementation

The PARAID Reliability Manager is responsible for managing the life expectancy of the PARAID device and also recovering from a disk failure. This component has not been

implemented in the prototype, but would be implemented as a Linux kernel module in the same fashion as the PARAID Gear-Shifting Logic and PARAID Monitor.

5.5 PARAID User Administration and Raidtools

The major responsibility of the PARAID user administration component—or pdadm for short—is to provide a way for the PARAID administrator to communicate with the PARAID device. This communication includes setting device configuration parameters, so that the PARAID device can be tweaked for optimal performance. The pdadm User Administration Tool uses the device I/O control for communication.

To send a command, pdadm obtains a handle to the PARAID device and issues an I/O control command to it. The command is received by the md device driver, which then reacts accordingly. Common commands to PARAID are hardwired into pdadm and the md device driver, and specific actions taken by the md device driver vary by the command.
Table 5.1 lists the commands that are currently implemented in the pdadm user administration program.

Table 5.1: pdadm controls

PARAID User Administration Controls	
pdadm -c <value>	Change the PARAID device active gear. '-c 1' to change to gear 1
pdadm –d <value>	Turn the PARAID debug information on or off. 0 - off, 1 – on
pdadm –e	Print md device debug data to /var/log/messages.
pdadm -i <value>	Change the time interval for the PARAID Monitor.
pdadm -m <value>	Turn the PARAID Monitor on or off. 0 - off, 1 - on.
pdadm -p <value>	Print paraid information. 0 – PARAID read/write statistics, 1 – PARAID current state, 2 – PARAID synchronization data from PARAID Disk Manager
pdadm -t <value>	Change the utilization threshold for the PARAID Monitor
pdadm –v	Print the version of pdadm

The pdadm program was implemented as a user space C program, which is one part of a larger set of RAID tools called Raidtools. Raidtools provide utilities to administer RAID devices and have been included in popular Linux distributions for years. Perhaps the most important of these is the mkraid tool, which creates RAID devices. The mkraid tool also examines a RAID configuration file called raidtab, which exists in the standard Linux /etc directory and issues device I/O control commands to the md device driver. The raidtab file defines the RAID device that is to be created by the mkraid tool: Table 5.2 lists a raidtab file that defines a RAID-0 device with four disks in the array.

Table 5.2: RAID Level 0 raidtab

RAID Level 0 /etc/raidtab

```
raiddev /dev/md0
raid-level              0
nr-raid-disks           4
persistent-superblock   0
chunk-size              4
device                  /dev/sdb1
raid-disk               0
device                  /dev/sdc1
raid-disk               1
device                  /dev/sdd1
raid-disk               2
device                  /dev/sde1
raid-disk               3
```

The raiddev parameter identifies the md device that is to be created. The raid-level parameter identifies the level of RAID to create—in this case RAID level 0–while the nr-raid-disks parameter defines the number of disks to be included in the array. The persistent-superblock parameter defines whether the super block should be persisted to the disk. (Persisting the superblock allows the RAID device to be shut down without having to recreate it to start the device again.) The chunk-size parameter identifies the size of the chunk, in Kbytes, that is to be used in the RAID device. The device parameter identifies disks to be included in the array while the raid-disk parameter identifies the role the disk above it will play in the array.

The mkraid tool had to be changed, so that it could handle making PARAID devices. Additional raidtab parameters had to be defined and the mkraid tool had to be updated to be able to recognize these new parameters. Table 5.3 lists a raidtab file for a PARAID device with four disks in the array and two gears. The first gear has disks 0 and 1, and the second gear has all four disks.

Table 5.3: PARAID raidtab

PARAID /etc/raidtab

```
raiddev /dev/md0
raid-level              9
nr-raid-disks           4
nr-gears                2
default-tier            1
persistent-superblock   1
chunk-size              4
debug                   0
device                  /dev/sdb1
raid-disk               0
device                  /dev/sdc1
raid-disk               1
device                  /dev/sdd1
raid-disk               2
device                  /dev/sde1
raid-disk               3
gear                    0
gear-width              2
gear                    1
gear-width              4
```

The new parameters added for PARAID are the nr-gears, gear, and gear-width. The nr-gears parameter identifies the number of gears in this PARAID device. The gear parameter identifies a gear and the gear-width parameter identifies the number of disks included in that gear. In addition, the md device driver had to be updated, so that it knows how to handle the new PARAID commands.

5.6 Source Listing

Table 5.4 lists the source files created or changed for the implementation of the PARAID prototype. The table includes the file name and the path where the file can be found, and it also includes a short description of the PARAID modifications. Additionally, a line count is included for each file. The total source consists of 3,435 lines of PARAID code.

Table 5.4: PARAID source listing

File	Description	Lines
/linux/drivers/md/paraid.c	Implements the PARAID RAID personality.	1070
/linux/include/linux/raid/paraid.h	Implements the PARAID RAID personality.	134
/linux/drivers/md/paraid-dm.c	Implements the PARAID Disk Manager	662
/linux/include/linux/raid/paraid-dm.h	Implements the PARAID Disk Manager.	52
/linux/drivers/md/paraid-mon.c	Implements the PARAID Monitor.	478
/linux/include/linux/raid/paraid-mon.h	Implements the PARAID Monitor.	44
/linux/drivers/md/paraid-pm.c	Implements the PARAID Print Manager Linux Kernel Module. This module contains all of the system output functions for the PARAID device.	182
/linux/include/linux/raid/paraid-pm.h	Implements the PARAID Print Manager.	18
/linux/drivers/md/md.c	Implements the Multiple Device, md.	212 (added)
/linux/include/linux/raid/md_u.h	Implements the Multiple Device, md, user space definitions.	24 (added)
/linux/include/linux/raid/md_p.h	Implements the Multiple Device, md, physical space definitions.	5 (added)
/linux/incude/linux/raid/md_k.h	Implements the Multiple Device, md, kernel space definitions.	16 (added)
/raidtools/pdadm.c	Implements the PARAID User Administration tool, pdadm.	480
/raidtools/mkraid.c	The make raid Raidtool implementation.	11 (added)
/raidtools/parser.c	The Raidtool command parser implementation.	47 (added)

CHAPTER 6

EVALUATING PARAID

Unlike most of the existing power reduction work, PARAID was evaluated empirically and under realistic workloads to verify the validity of its design. Empirical evaluation of PARAID involves prototyping PARAID, constructing a power measurement framework, and searching for appropriate and realistic workloads. The two types of workload used were a web server workload and the PostMark benchmark [14].

6.1 Measurement

A measurement framework was built to empirically evaluate the execution of the PARAID prototype (Figure 6.1). The measurement framework included both the hardware and software components.

6.1.1 Measurement Framework Hardware

The measurement system hardware consisted of two computers, with one acting as a client and the other a server. The client computer hosted the trace playback system that generated workload, and the server computer hosted a web server that used a RAID storage device (Table 6.1). Four disks were used to build the RAID devices on the server. The client and server computers were connected directly to each other by a CAT-6 crossover cable so that extraneous network traffic would not interfere with the experiments.

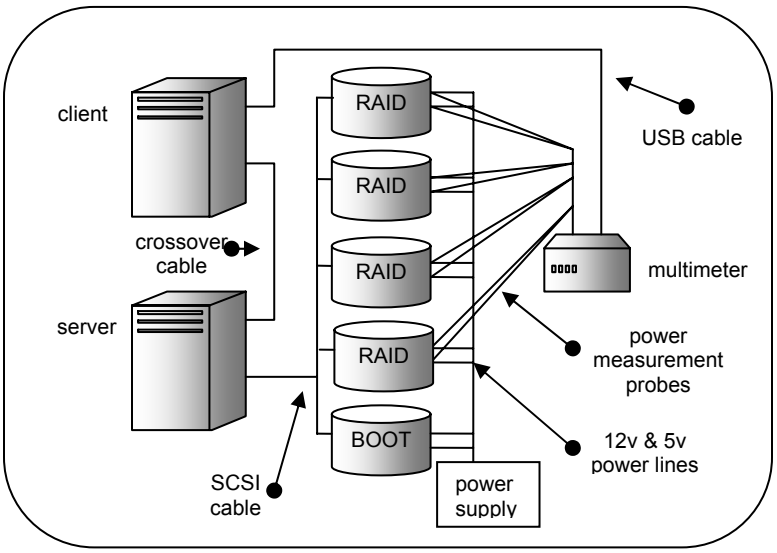

Figure 6.1: Measurement framework

Table 6.1: Measurement framework hardware

	Server	Client
Processor	Intel Xeon 2.8 Ghz	Intel Pentium 4.2.8 Ghz
Memory	2 Gigabytes	1 Gigabytes
Network	Gigabit Ethernet	Gigabit Ethernet
Disks	36.7 GB 15k RPM SCSI Ultra 320	160 GB 7200 RPM SATA

To measure the power of the disks, the power measurement framework included an Agilent 34970A digital multimeter. The multimeter sampled the power of each disk several times per second. Each disk was connected to the multimeter on a unique channel, and the multimeter was connected to the client computer via a universal serial bus. The sampled data gathered by the multimeter was sent to the client computer. The multimeter took several samples per channel per second but logged only the average to the client computer, approximately once per second. Figure 6.1 shows the client and server computers and the multimeter in the measurement system.

Measuring the power consumption of any electrical device is based on Ohm's Laws, which state that current can be calculated by dividing the voltage drop across a resistor by the

amount of resistance from the resistor. Ohm's Laws also state that the power used by a device can be calculated by multiplying the voltage drop across the device by the current.

Table 6.2: Ohms Laws

Ohms Law For Current
I (Current) = V_r (Voltage Drop of Resistor) / R (Resistor)
Ohms Law for Power
P(Power) = V_d (Voltage Drop of Disk Device) * I

To use Ohm's Laws to measure the power of a disk, a closed circuit was created and a resistor was inserted in series at the beginning of the circuit, as shown in Figure 6.2. A multimeter measured the voltage drop across the resistor, V_r, by measuring the voltage at the beginning of the resistor and the voltage at the end of the resistor, and calculating the difference between the two. The amount of current used by the resistor—which is also the current used by the disk—can thus be calculated using Ohm's Laws, as listed in Table 6.2. A multimeter also measures the voltage drop across the disk, V_d, and this is used to calculate the power of the disk.

To create this closed circuit in the measurement system, each disk was removed from the web server, and a resistor was introduced into each disk's 12V+ and 5V+ power lines. The 12V+ line supplied power to the spindle motor; the 5V+ line provided power to the disk electronic chipset. The SCSI cable is connected directly to the motherboard, which allows the SCSI cable to maintain the same performance as if the disks were connected to the SCSI hot swappable backplane in the server.

A 0.1 Ohm resistor was inserted in series between the power supply and the adapter for each 12V+ and 5V+ line. One set of multimeter probes was attached to both ends of the resistor, and another set was attached to each of the positive and negative lines of the corresponding disk, as shown in Figure 6.3. This allowed the multimeter to measure the voltage drop across the

resistor, and the voltage drop of the disk. The current of the disk was calculated from the voltage drop of the resistor. The voltage drop across the disk remained constant.

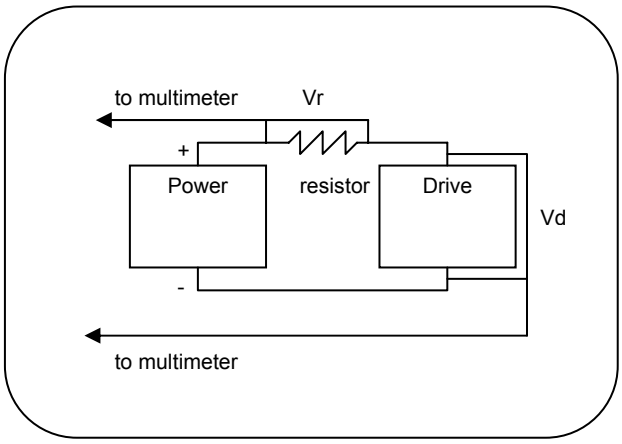

Figure 6.2: Introducing a resistor in series between the disk and the power supply.

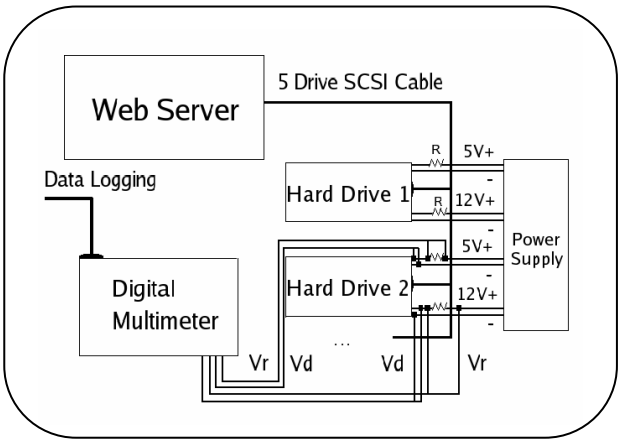

Figure 6.3: Measuring the power of the disks in the measurement system.

6.1.2 Measurement Framework Software

The software used in the measurement framework falls into two groups: the web server software and client workload software. During the experiments, the server received the workload generated from the client, so that PARAID running on the server could be evaluated. The PARAID prototype was built under Linux 2.6.5, which required the installation of the Fedora Core 2 operating system on the server. The server used the ext2 file system and also had the Apache web server 2.0.52 installed.

During the experiments, the client is responsible for generating the workload that is sent to the server and logging the power measurement data from the multimeter. The client has Microsoft Windows XP installed because the Agilent Multimeter software logs the data using Microsoft Excel.

The trace playback software that resides on the client is called WebClient, and it was developed by Mathew Oldham at Florida State University [18]. The WebClient program was implemented in Java version 1.5 and was designed to playback web log traces.

While replaying the web log activities, the WebClient program generates performance statistics. The WebClient program measures end-to-end performance metrics that encompass both server and per-request performance numbers. WebClient measures latency, which is the elapsed time to receive the first response from the server for a given request. WebClient also measures the elapsed time between sending the first byte of the request to receiving the last byte of the response for every request. Additionally, WebClient records the total number of connections completed per second and the total number of concurrent connections between the client and server. The WebClient measures throughput by accounting for the total number of bytes sent by the server during the corresponding transmitting interval. These end-to-end statistics, generated by WebClient, are used to evaluate the performance of the PARAID prototype.

6.2 Workload

An ideal workload stresses a system in representative ways, so that realistic system performance characteristics can be captured. Trace replays are commonly used to capture this level of realism. The downside of this choice is that traces may also reflect hardware characteristics that cannot be easily duplicated with the benchmarking framework. The workload chosen was a Web server workload, which is unique in that Web server activity exhibits daily fluctuations in amount of activity. More activity was observed during the day and less during the night. This fluctuating characteristic was essential for evaluating PARAID in its ability to save power.

Capturing the real workload is tricky. It is important to capture a snapshot of the file system of the traced server so that the effects of fragmentation are not overlooked in measurement. Most system administrators have security concerns for revealing such level of details of a system [21]. Web servers might contain sensitive user data that cannot be revealed, such as the answers for final exams. To work around this problem, the workload capture program disregarded the actual file content stored on the web server and captured only the name, size, type of file, file creation time stamp, modification time stamp, and last access time stamp. The capture program then encrypted the file names using the SHA-1 algorithm [22] and filled files with randomized characters.

The trace was captured from an academic web server within the Computer Science Department at Florida State University. This web server is used by all students and faculty of the department for classes, administration, and research. The activity was captured from August 2004 to November 2004. The file system contained approximately 47 Gbytes of data, 44,000 directories, and 518,000 files.

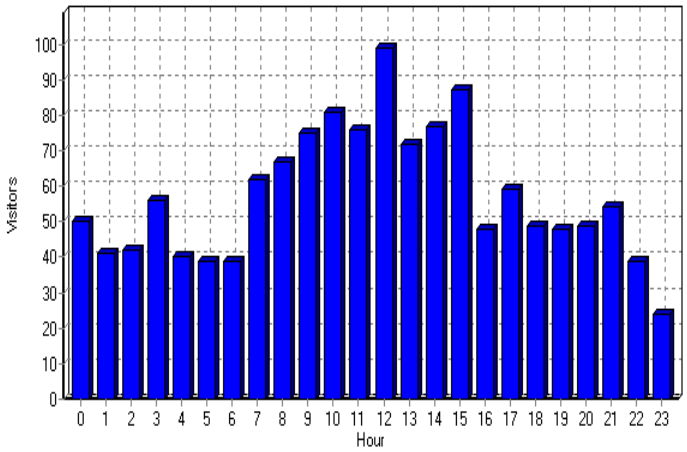

Figure 6.4: September 23, 2004, workload

After analyzing the traces, September 23, 2004 was chosen as one of the Wednesdays that had a higher volume of activity. The data from that day contains only the typical daily fluctuation in load, which can be exploited by PARAID for saving power. This particular 24-hour period of trace activity had an approximate total 450 Mbytes of data transmitted within 17,000 requests. The activity for this day is shown in Figure 6.4. Notice that the peak traffic is in the middle of the day, during which time, PARAID will gear-shift into a higher gear. For the remainder of the day, PARAID can find opportunities to save power.

To warm up the cache, six hours of trace activity from the day before was run before the data gathering began. The total duration of the trace used in the experiments was lengthened to 30 hours.

In addition to the web server workload, the PostMark benchmark was used to evaluate PARAID. PostMark is designed to simulate an Internet Service Provider (ISP) workload. ISP workload consists of a combination of electronic mail, netnews, and web-based transactional traffic [14]. The PostMark benchmark was chosen because it can stress the peak performance of PARAID.

6.3 Experimental Results

It was important to compare PARAID with a conventional RAID 0 device, since neither have reliability mechanisms such as mirroring or parity bits. Comparing PARAID to other RAID levels with reliability mechanisms will overstate PARAID's performance numbers.

The PARAID prototype was also compared with a modified RAID 0 device, RAID-0 LRU, which used a least recently used (LRU) policy to determine when to power off individual disks [2]. Since a disk experiences a significant spike in power as it is powered on, a disk that is too frequently power-cycled will use more power than a disk always powered. Therefore, a break-even threshold is used, which is an interval at which the power saved by turning off a disk during this interval is equal to the power used to bring the disk back online. Also, this interval needs to account for the time needed to spin a disk back up, which can take around 8 seconds for server-class drives.

For the RAID-0 LRU device, the LRU policy used an 18-second break-even threshold. Therefore, a disk is not powered off unless it has been inactive for at least 18 seconds. When using an LRU policy for each disk in RAID-0 LRU, some of the disks can stay powered, while others might be powered off. (In practice, this scenario was rare, because all disks are required to serve the I/O requests due to the uniform striping pattern.)

It is important to evaluate RAID-0 LRU, because it demonstrates that server-class storage has few opportunities to power all disks off. Therefore, it was expected that RAID-0 LRU would perform in a manner similar to RAID 0 most of the time. Only when there was no workload would RAID-0 LRU actually save power.

The PARAID device used in the experiments, PARAID-0 2g, had two gears: the first gear had two disks, and the second gear had four. This device switches into a high gear when the

workload on the system demands it. When the workload is light, this device switches into the lower gear, using only two disks, and thus saves power.

The RAID-0 device had four disks. All RAID 0, RAID-0 LRU, and PARAID-0 2g were compiled on the measurement server, and all three used a chunk size of 4 Kbytes with the ext2 file system installed.

To conduct each experiment, the trace of the web workload captured on September 23 was replayed on the measurement framework. The client generated activity to the web server, which ultimately forwarded I/O requests to the benchmarking devices. Each experiment was repeated four times. In preliminary experiments, replaying the trace at normal speed did not generate enough activity on the web server to cause the PARAID-0 2g device to switch into a higher gear, so the web user arrival rate was accelerated to eight times the normal speed. (Timing dependent on human interactions, such as the time between user mouse clicks on links, is not accelerated.)

In the evaluation of PARAID, two important questions need to be answered. First, does a PARAID device reduce the amount of power consumed, compared to a conventional RAID? Second, is performance preserved using a PARAID device compared to that of a conventional RAID?

6.3.1 Power Measurements

Figure 6.5 shows the amount of power consumed versus time for RAID-0, RAID-0 LRU, and PARAID-0 2g respectively. The lines strongly correlate to the level of workload over the 24-hour trace playback period: more power is consumed during the midday when the web traffic volume was heaviest as seen in Figure 6.4.

To produce these graphs, the voltage drops were sampled across the resistors and the disks several times per second and were averaged every one second. The one-second averages were used to compute the power used per second, and these numbers were averaged over a four-minute period to form a data point.

Figure 6.5 shows that RAID 0 ranged between 20 and 34 watts; RAID-0 LRU ranged between 12 and 33 watts; and PARAID-0 2g ranged between 16 and 32 watts.

On average, RAID-0 LRU used 5% less power usage than RAID 0; however, at a given time, RAID-0 LRU can save anywhere from 0% to 47% l. PARAID-0 2g saved 23% of power

on average, and it can save between 6% and 40% power at a given time. Note that powering off a disk only stops a disk from spinning its platter, and therefore, only the 12V line is powered off. Power is still needed for the 5v line that powers a disk's electronic chipset, so that the disk can pass commands along the daisy-chained disks via the SCSI interface cable. Therefore, the power never drops to zero for the PARAID cases, even when all of the disks are powered off. In fact, a disk with the spindle motor powered off still consumes about three watts of power for the electronic chipset.

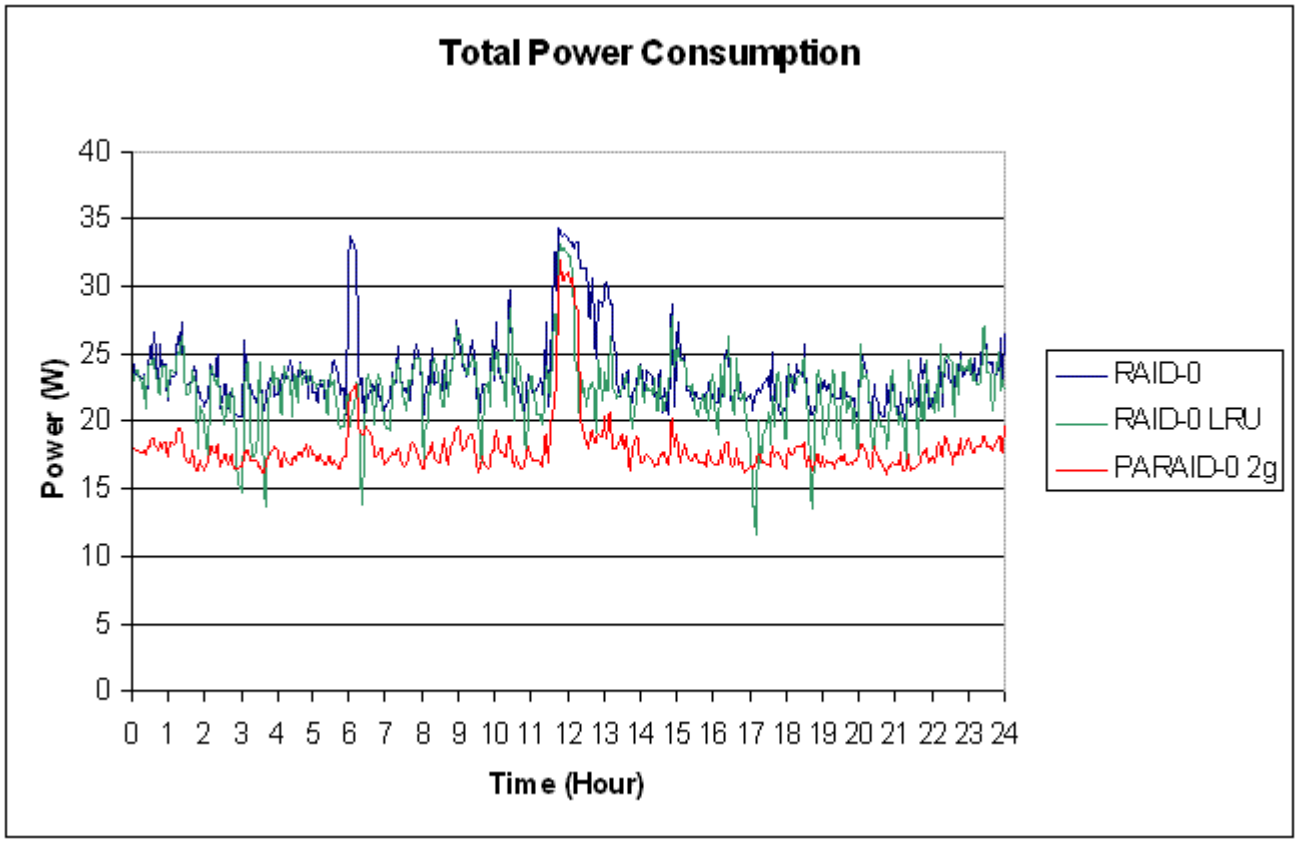

Figure 6.5: Power measurements

6.3.2 Performance Measurements

For performance, Figures 6.6 and 6.7 show CDFs of per-request latency and completion time respectively in milliseconds (ms). The per-request latency measures the time for the web server to return the first byte of data. For RAID 0, 65% of the files had latencies less than 1 ms, and almost 100% of the files had latencies less than 100 ms. For RAID-0 LRU, 75% of the files had latencies less than 1 ms. RAID-0 LRU drops some of the requests due to the latency of spinning disks back up, which is greater than the web server timeout. This was not a problem for PARAID-0 2g because PARAID dose not switch to the new gear until the disks have spun up. Because of this, Figure 6.6 shows that some of the requests were never served: less than 100% of the files served in 1000 ms. For PARAID-0 2g, 77% of the files have latencies of less than 1 ms, and almost 100% of the files had latencies of less than 90 ms.

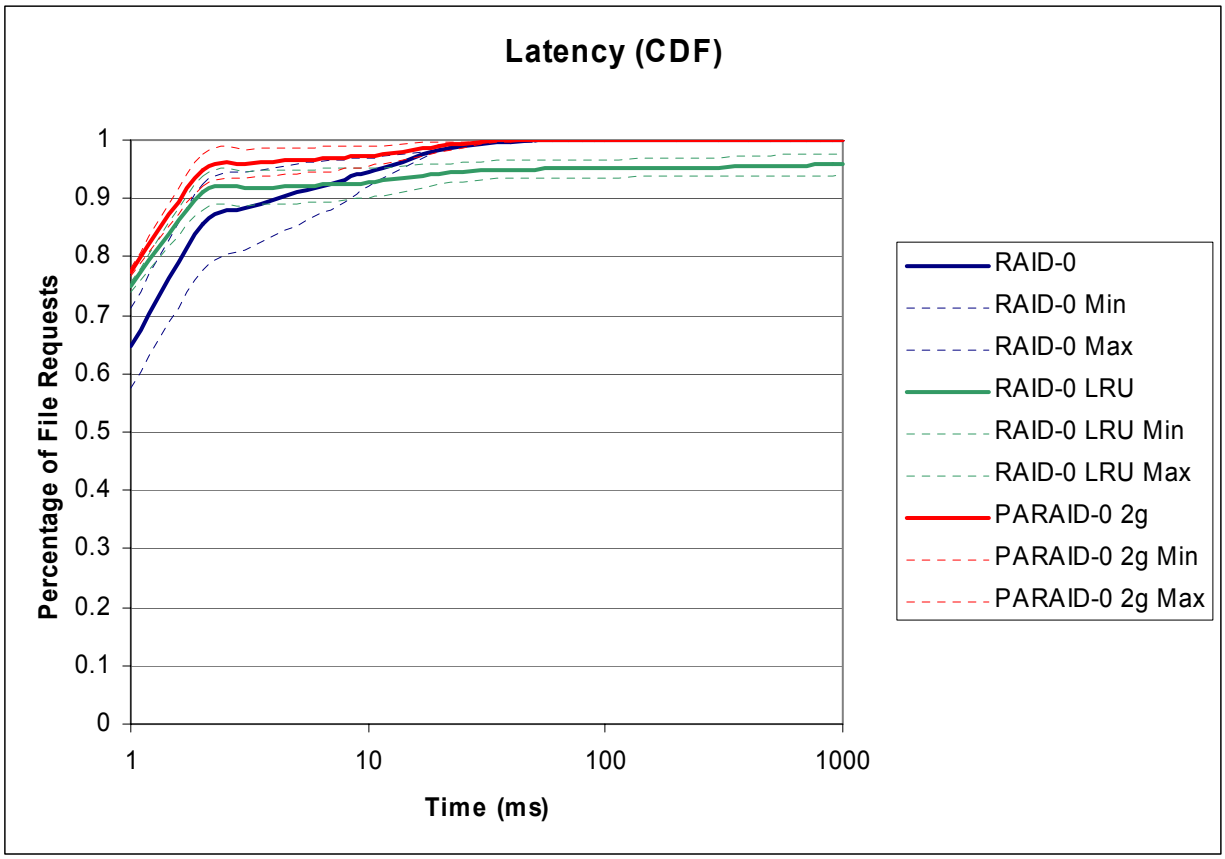

Figure 6.6: Latency measurements

Figure 6.7 shows the CDF of per-request total completion time, which is the time between sending the first byte of a request from the client to receiving the last byte from the server (at the client end). For RAID 0, 34% of the requests had a completion time of less than 1 ms, while almost 100% of the files had a total completion time of less than 400 ms. For RAID-0 LRU, 40% of the files had a completion time of less than 1 ms. Again, not all of the requests were successfully served by the web server using RAID-0 LRU. For PARAID-0 2g, 41% of the files were served in less than 1 ms, and almost 100% of the files were served in less than 600 ms.

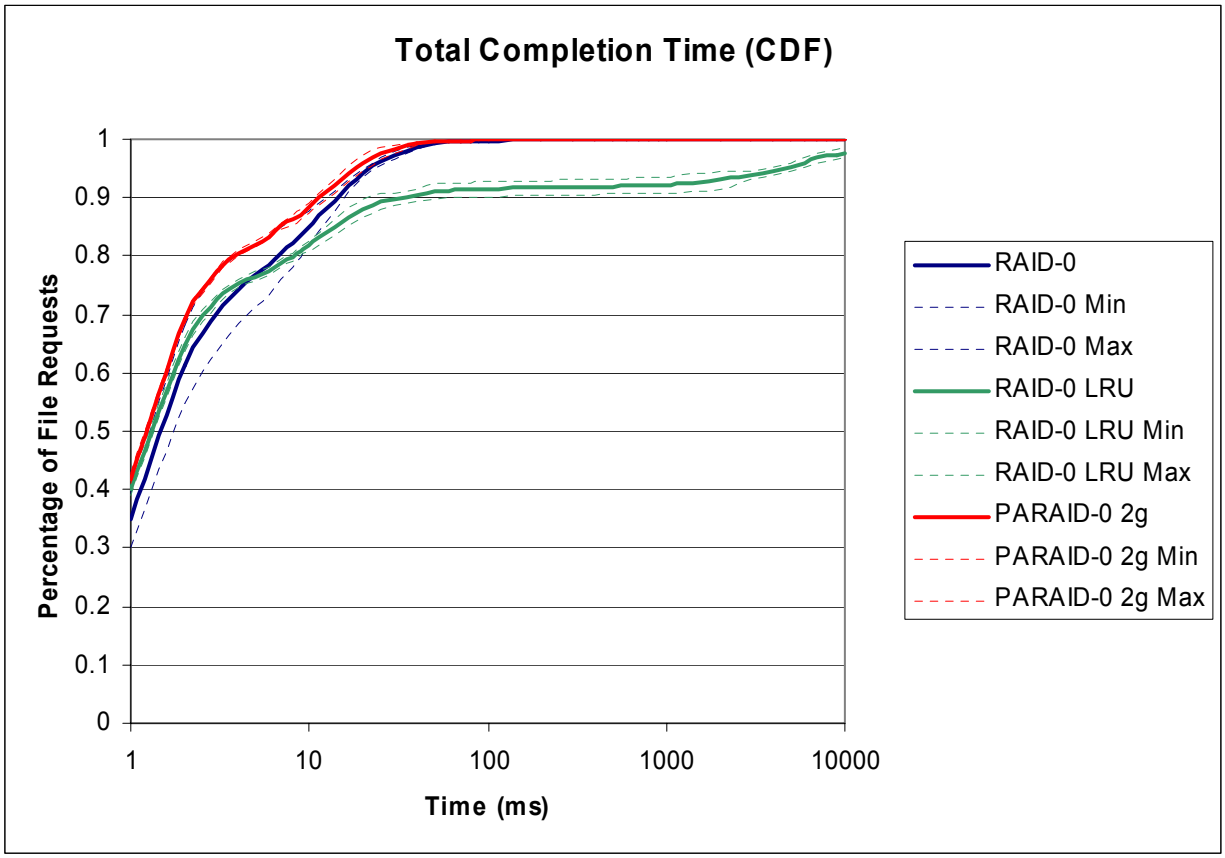

Figure 6.7: Total Completion Time measurements

6.3.3 Postmark Benchmark Measurements

The PostMark benchmark is a popular ISP synthetic benchmark, which was used to stress the peak performance of a storage device for its read- and write-intensive activity [14]. The PostMark Benchmark was run only on the server (without the client) (Figure 6.1). Figure 6.8 shows three sets of PostMark numbers for RAID 0 and PARAID-0 2g. Each benchmark was conducted four times directly on the server used for the web benchmark. With an average file size of 1 Kbyte, RAID-0 was able to complete 50,000 transactions in 5.6 seconds on average, while PARAID-0 2g did the same in 4.4 seconds. With an average file size increased to 20 Kbytes, RAID 0 was able to complete 50,000 transactions in 52 seconds on average, while PARAID-0 2g did the same in 54 seconds. With the same file size setting with 100,000 transactions, RAID 0 completed the benchmark in 110 seconds while PARAID-0 2g did the same in 116 seconds. Overall, the observed overhead is within 5%.

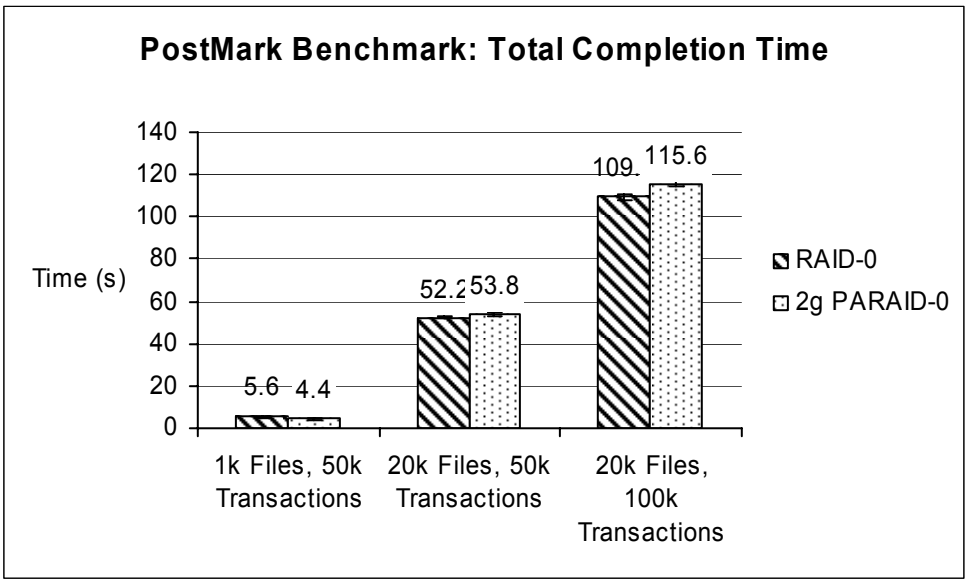

Figure 6.8: PostMark Benchmark measurements

6.3.4 Reliability Measurements

The reported power benefits of PARAID-0 2g involved only one power cycle over the course of the 24-hour trace playback period. This power cycle took place during the middle of the day when the PARAID-0 2g device gear shifted from gear 1 into gear 2 to accommodate the high activity, and then gear-shifted back to gear 1 once the activity had decreased. Also, only the two disks not in the first gear were power–cycled; therefore, it is conceivable that PARAID-0 2g can operate with 365 power cycles a year, knowing that the daily fluctuations of load are highly predictable. With a 20,000 power-cycle rating [5], the number of power cycles incurred by PARAID should have a very limited impact on the life expectancy and the reliability of these disks.

6.4 Discussion of Results

The empirical results need to answer two questions regarding power and performance. The experiments also raise more questions: what is the impact of having more gears on a PARAID device? Why did PARAID-0 2g outperform RAID 0 in latency and total completion time at times? What impact did the workload used for the experiments have on the results?

In terms of the power impact made by the number of gears, two gears saved 18% more power than using only one gear. For one gear, the only opportunity to save power is when there is no traffic on the server. Outside of that opportunity, RAID-0 LRU will use just as much power as RAID 0. With a busier workload, a one-gear PARAID device would save even less power. Having gears allows PARAID to more accurately vary the number of powered disks according to demand. Figure 6.9 shows the amount of power savings lost using a one-gear PARAID device. Figure 6.10 shows how having multiple gears allows one to save more power by closely matching the workload curve.

Having more gears can save more energy, but there is a capacity and write propagation overhead associated with each additional gear. PARAID trades capacity for saving energy, and the capacity overhead is a factor of the number gears in the device and the number of disks per gear. In addition, PARAID relies on block replication to be able to create the gears. As the number of gears increases, more blocks need to be replicated and maintained for consistency.

Because of the overhead, using fewer gears that save a significant amount of energy is preferable to using many gears that only offer marginally more energy savings.

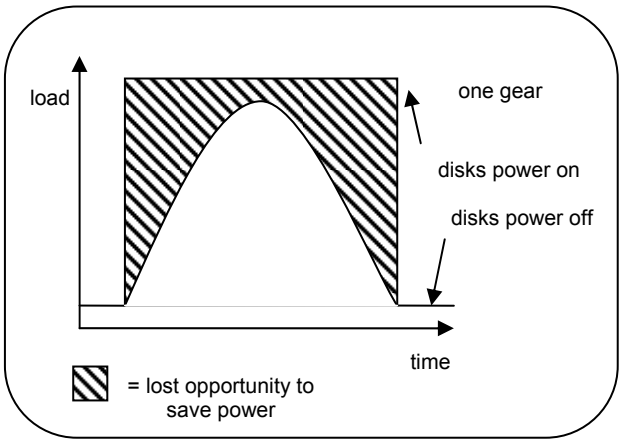

Figure 6.9: Opportunity lost to save power by RAID 0

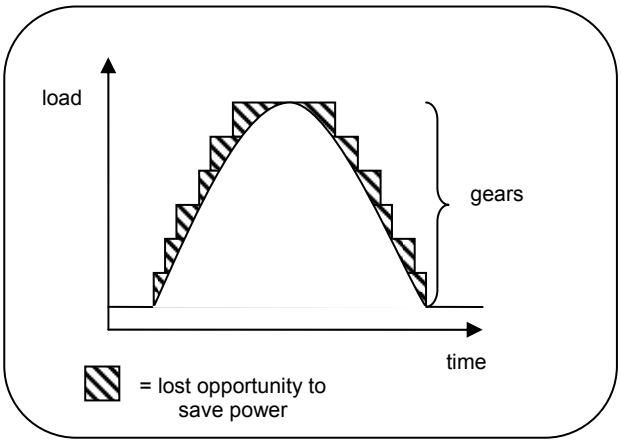

Figure 6.10: Opportunity lost to save power by PARAID.

In terms of performance, RAID-0 LRU performed worse than PARAID-0 2g. The slower latency can be directly attributed to the need to power on more disks from power-saving mode. Having multiple gears hides this delay better by having a threshold to time-cushion the disk spin-up time.

PARAID-0 2g on average used 23% less power than RAID 0, with only a four-disk array. Should PARAID use a larger array with more gears, over a longer period of time, the savings would be even more significant.

PARAID-0 2g performed well compared to RAID 0, in both latency and total completion time. With statistical significance, PARAID-0 2g was able to serve 10+% more requests with a latency of 1 ms. In terms of total completion time, PARAID-0 2g outperformed RAID 0 for the first 10 ms, having 4% to 7% more requests completed. RAID 0 did eventually catch up to PARAID-0 2g performance and ultimately surpass PARAID-0 2g as the latency and the total completion time increased. For a uniformly striped RAID, as the number of disks increase, the probability of one of the disks incurring a full seek and waiting for a full rotation approaches 1. On the other hand, skewed striping allows smaller files to be served faster as the number of

powered disks decreases. The RAID devices used in these experiments had a chunk size of 4 Kbytes. If the requested file was over 8 Kbytes, more than two disks would be required to serve the file in the RAID-0 device, while PARAID-0 2g can potentially use only two disks when it is operating in the first gear.

RAID 0 has the potential to serve large files faster than PARAID-0 2g, while operating with only the first gear. The throughput of RAID 0—a factor of the number of disks that can transfer bytes in parallel—will be better than that of PARAID-0 2g at lower gears. This might explain why eventually RAID 0 catches up to PARAID-0 2g as the total time to completion increases.

This analysis reveals that the type of workload has strong implications on the opportunities to save power. The web workload used had a defined fluctuation of loads. The power measurement curves strongly correlate with the workload curves, and PARAID with multiple gears was able to take advantage of that fact.

How would a change in the workload affect the results? If workload had been heavier over the course of a 24-hour period, there would have been less of an opportunity for PARAID to save power. In fact, a heavy enough workload would prevent any form of power-saving approaches from being effective. The reverse of that is also true: if the workload had been lighter, more power would have been saved. However, a trivial all-on-and-off RAID-0 LRU can serve the purpose of power-saving. What has been revealed through experimentation is that by matching the way disks are used with the fluctuating workload, significant power benefits can be gained. Also, it is possible to meet peak performance needs while degrading performance in a way not perceivable by end users during off-peak hours.

In summary, PARAID-0 2g did save power with limited performance degradation. It does make a significant difference to have multiple gears in the PARAID device. This allows PARAID to more closely follow the workload curve, ultimately saving more power. PARAID-0 2g outperforms RAID 0 for small files, but underperforms with large files, explaining the gaps between the two devices in latency and total completion time. Workload certainly plays a significant role in the amount of power that can be saved, and the opportunity for PARAID to save power is dependent on the characteristics of the system workload.

CHAPTER 7

RELATED WORK

Previous energy reduction work has mostly been done in the area of mobile computing. Only recently has reducing energy been a concern in server-class computing. Some approaches directly involve the storage hardware; some involve RAIDs; other approaches involve the entire file system or even the applications. This chapter focuses on RAID approaches. Note that most results obtained in the arena of energy reduction are based on simulation, whereas PARAID was prototyped and empirically measured.

Through simulation, massive arrays of idle disks (MAID) [4], have shown to use only 15% of the power that is used in comparable RAIDs that are always powered. The primary goal of MAID is to provide a disk-based archival system that can compete with tape libraries in terms of capacity and performance. MAID takes advantage of the observation that 50% of the data that is written is never accessed in a mass storage system in super computing environments [15]. Therefore, MAID is structured into two sets of disks—data disks and cache disks. The data disks hold all data blocks, while the cache disks hold a copy of the recently accessed data blocks. When an I/O request arrives, MAID first checks the cache disks. If the data is on the cache disks, the data is returned, otherwise the request is sent to the data disks. MAID reduces power by keeping the cache disks always powered and spinning down the data disks when not needed. MAID is effective in saving power for mass storage that handles tape archival workload because this workload is predictable. The cache disks are populated with the data blocks that are accessed frequently, allowing the data disks to stay powered off.

PARAID differs from MAID in both intention and design. The intention of MAID is to provide a competitive alternative to tape archival storage systems. Under tape archival workload, MAID can excel to save power. MAID uses a power management policy similar to

RAID-0 LRU; the disks are either on or off based on disk activity. If the workload contains light but short periodic accesses with consistent cache misses, the power management policy of MAID will keep all the disks powered. This provides MAID with limited opportunity to save power on workload that is fluctuating in nature. PARAID can hug the curve of the workload by gear shifting into the best gear, allowing PARAID to save power more aggressively. As far as capacity is concerned, MAID requires cache disks.

Popular data concentration (PDC) [20] claims to save more power than MAID. PDC centers on the popularity, or the frequency, of file access. PDC puts the most popular data on the first disk, the second most popular on the second disk, and so on. Disks are powered off in PDC based on activity; disks that are not active for a certain duration are spun down. PDC was evaluated via simulations.

PARAID differs from PDC in the way data is skewed. PARAID will outperform PDC in throughput because PARAID balances load within each gear through uniform striping, while PDC stores each file only on one disk. PDC does create more opportunities to power off disks, but PDC sacrifices performance both at the peak and light system loads.

The power-aware cache management policy (PA-LRU) [27] claims to use 16% less energy and achieve a 50% better average response time than that of LRU cache management policy. PA-LRU assumes no striping, and it saves power by caching more data blocks from the less active disks. Lengthening the access interval for these less active disks allows these disks to be powered off for longer durations. The authors overcame some of the shortcomings of PA-LRU by later introducing the partition-based cache management policy (PB-LRU) [28]. PB-LRU divides the cache into separate partitions for each disk. Each partition is managed separately by a replacement algorithm such as LRU. PB-LRU can adapt to different workloads more quickly (something PA-LRU had trouble doing) because the cache miss sequencing is easier to control. PB-LRU provides similar energy savings and I/O response times to that of PA-LRU. In both papers, the evaluation was based on simulation.

PARAID differs significantly from the approaches by PA-LRU [27, 28]; therefore, it is difficult to compare the two directly. The biggest difference is that the work done with PARAID considers the effects of striping in storage while PA-LRU does not. This has significant impacts on performance in storage. PA-LRU will simply not be able to deliver the same level of performance as PARAID while saving energy.

Carrera et al. [2] and Gurumurthi et al. [9] explored power reduction of server-class disks by varying the speed of the disks. Both papers showed through simulation that power consumption can be significantly reduced. Both of these works simulated a hypothetical multi-speed disk. The simulated disk varies its speed according to the load imposed on the disk, in an effort to save energy. On the other hand, PARAID uses off-the-shelf disks and is prototyped to demonstrate the actual energy savings.

CHAPTER 8
FUTURE WORK

The research associated with PARAID is on-going work. There are several areas of continuing research with PARAID: determining the reliability of a PARAID device, exploring different striping strategies, measuring PARAID under a wider range of workloads, and extending PARAID to be based on RAID 5. Also, the PARAID Reliability Manager was not implemented in the initial prototype and is currently under development.

8.1 Reliability Simulator

It is difficult to test disk failure from power cycling because of the time commitment required. The disks used in the PARAID evaluation have a 20,000 power-cycle rating. It is simply not practical to test this empirically by power cycling a disk 20,000 or more times. To overcome this obstacle, a simulator will be built to project the usefulness of the PARAID Reliability Manager rationing and role exchange algorithms. Validation will only be done with limited time constraints (e.g., measuring the aging of a disk according one of the popular S.M.A.R.T tools [23]).

The simulator will work with the actual PARAID components, so that input into the simulator will be the same input for the PARAID Reliability Manager. A trace will be used to drive the simulator. The simulator will allow for different algorithms to be tested quickly and also project the reliability of PARAID in terms of disk failures due to power cycling.

8.2 Striping Strategy

The current PARAID-prototype changes the disk layout of RAID. Therefore, adapting to different RAID levels requires the creation of different PARAID levels. Another approach is to store soft replicated states in the unused portion of a RAID to create the gears. The replicated states are considered soft because each state can be invalidated and reconstructed without affecting the data consistency and correctness of the original RAID operations. The original RAID level disk layout then can be preserved. Soft replicated states can be created in a skewed fashion. This more modular and flexible approach would allow PARAID to sit easily on top of current RAID levels. Also, PARAID could be augmented to existing RAIDs. Figure 8.1 shows an example of the soft-state PARAID design. The example PARAID device in the figure has four disks and three gears. The first gear consists of disks 1 and 2; the second gear, disks 1 to 3. Both gears 1 and 2 use soft state. The third gear consists of all four disks and does not use soft state. When operating in this gear, the original RAID code and disk layout is used. To create the gears, the data blocks from disk 4 are replicated to form the second gear, and then the blocks from disk 3 are replicated to form the first gear. Because the data blocks of disk 4 are replicated in gear 2, disk 4 can be powered off when PARAID is in gear 2. The same applies for disks 3 and 4 when gear shifting into gear 1.

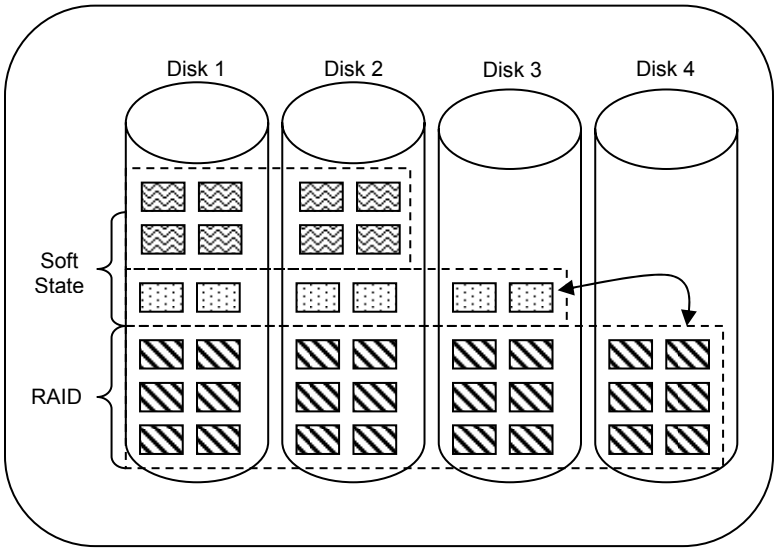

Figure 8.1: Soft state skewed striping design

8.3 Workload

Workload plays a large role in the success of PARAID. How much energy can be saved is dependent on the type of workload. The evaluated PARAID prototype has been tested against web server workload. Other workload types need to be tested for the applicability of PARAID to saving power. For example, database workload is heavily transaction-oriented and tends to have more write requests, stressing the server throughput and response time [11]. Database workload will evaluate how well PARAID handles update propagation when a drive is brought back online. Also, PARAID needs to be evaluated against heavier web server workload. The web server workload obtained in this evaluation had to be accelerated to be able to stress the measurement framework.

8.4 RAID Level 5

The initial PARAID prototype was built with a uniform striping pattern within each gear, which provided no redundancy, similar to RAID 0. Therefore, there is no way to recover from a disk failure within the first gear of PARAID.

To be as reliable as RAID 5, one of two strategies can be employed. The first strategy is to construct a parity block for every stripe as is done in RAID 5 for the first gear only. The parity block allows for a single disk failure within the first gear. Also, the additional properties of being able to handle any number of disk failures outside of the first gear would still hold. This approach is more reliable than RAID 5 due to other replicated blocks. The first gear consists of a minimum of three disks imposed by existing RAID 5 standard. Three disks are needed because a disk is used to store the parity. If only two disks are used, mirroring is a better alternative.

The second strategy to be as reliable as RAID 5 is to use soft state replication as mentioned above. Since the soft state sits on top of a RAID level personality, the underlying RAID 5 mechanisms can be completely reused. This approach is less intrusive and can be applied to other existing RAID levels. Also the skewed striping strategy could be easily changed. A skewed striping strategy based on file temperature could be interchanged very easily when using a soft state approach.

CHAPTER 9
CONCLUSION

This thesis hypothesizes that by using a novel data distribution technique, it is possible to achieve energy savings while preserving performance on server-class storage systems. Through the use of a novel skewed striping pattern, PARAID was able to achieve performance in both latency and total completion time comparable to that of conventional RAID, while consuming 23% less energy.

The prototyping and evaluation processes of PARAID have illustrated the following lessons:

- Servers are purchased to meet the peak load specifications, leaving the system under-utilized most of the time. PARAID exploited underutilizations in storage capacity and performance and translate them into power savings.
- Since performance has been the primary focus of servers, energy savings are not built-in to the system design. This afterthought can be seen in the existing RAID designs and the construction of the PARAID power measurement framework.
- Since servers constantly receive requests, conventional ways of using system-wide measure of idleness to govern power provide limited opportunities. By organizing storage resources in hierarchical overlapping sets of disks or gears, power can be saved by powering the appropriate set of components to match the fluctuations of system demands. The skewed replication strategy used by PARAID and smooth switching among gears also mask the disk spin-up latency.
- Cyclic fluctuations detected in user loads allow PARAID to use stable and limited gear shifting to achieve power savings and deliver matching performance. Since the reliability of server-class drives assumes limited number of power cycles, PARAID can also

leverage the knowledge of cyclic load fluctuations to power switch sparingly and effectively, thus controlling and limiting the adverse effects of power cycling on reliability.

PARAID has demonstrated that achieving energy savings does not necessarily involve significant performance loss. Another insight that emerges from the PARAID is that the characteristics of the workload ultimately dictate the system design. For the average performance of PARAID, this thesis has only explored an academic web server trace. Different workload types with different fractions of reads and writes, file systems, workload curves, peak-to-trough ratios of workload volumes, and access localities will lead to different strategies to form hierarchical overlapping set of disks. Different groupings of disks lead to different performance, energy consumption, reliability characteristics, and overhead characteristics. This insight also leads us to revisit the implications of workloads and to design more tailored systems.

REFRENCES

[1] Cao P, Felten EW, Li K, Implementation and Performance of Application-Controlled File Caching, *Proceedings of the 1st Operating Systems Design and Implementation Symposium*, 1994.

[2] Carrera E, Pinheiro E, Bianchini R, Conserving Disk Energy in Network Servers, *Proceedings of the 17th Annucal ACM International Conference on Super Computers*, 2003.

[3] Chen P, Lee E, Gibson G, Katz R, Patterson D, RAID: High-Performance, Reliable Secondary Storage, *ACM Computing Surveys*, 26(2), June 1994.

[4] Colarelli D, Grunwald D, Massive Arrays of Idle Disks For Storage Archives, *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, November 2002.

[5] Fujitsu, MAP Series Disk Drive, 2005. http://www2.fcpa.fujitsu.com/sp_support/ext/enterprise/datasheets/map10krpm-datasheet.pdf.

[6] Fujitsu, MAS Series Disk Drive, 2005. http://www2.fcpa.fujitsu.com/sp_support/ext/enterprise/datasheets/mas15krpm-datasheet.pdf.

[7] Fujitsu, MHV 4200 RPM Series Disk Drive, 2005. http://www.fcpa.fujitsu.com/download/download/hard-drives/mhv-at-datasheet.pdf.

[8] Fujitsu, MHV 5400 RPM Series Disk Drive, 2005 http://www.fcpa.fujitsu.com/download/download/hard-drives/mhv-ah-datasheet.pdf.

[9] Gurumurthi S, Sivasubramaniam A, Kandemir M, Franke H, DRPM: Dynamic Speed Control for Power Management in Server Class Disks, *Proceedings of the International Symposium on Computer Architecture*, pages 169-179, June 2003.

[10] Hamblen M, IS Managers caught in storage capacity chase, *Computerworld*, May 1997. http://www.computerworld.com/news/1997/story/0,11280,5204,00.html

[11] Hsu W, Smith A, Young H, Characteristics of production database workloads and the TPC benchmarks, *IBM System Journal*, 40(3), pages 781-802, 2001.

[12] Huang H, Pillai P, Shin KG, Design and Implementation of Power Aware Virtual Memory, *Proceedings of the 2003 USENIX Annual Technical Conference*, 2003.

[13] Iyengar A, Challenger J, Dias D, Dantzig P, High-performance Web Site Design Techniques, *IEEE Internet Computing*, 4(2):17–26, March 2000.

[14] Katcher J, PostMark: A New File System Benchmark, Technical Report TR3022, Network Appliance Inc., October 1997.

[15] Miller E, Katz R, An analysis of file migration in a Unix supercomputing environments, *Proceedings of the 1993 USENIX Winter Technical Conference*, pages 421-433, 1993.

[16] Moore J, Chase J, Ranganathan P, Sharma R, Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers, *Proceedings of the 2005 USENIX Annual Technical Conference*, 2005.

[17] Nightingale EB, Flinn J, Energy-Efficiency and Storage Flexibility in the Blue File System, *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, December 2005.

[18] Oldham M, A Power and Performance Measurement Framework for Server-Class Storage, Honors Thesis, Florida State University, April 2005.

[19] Patterson DA, Gibson G, Katz RH, A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD International Conference on Management of Data*, 1(3):109-116, June 1988.

[20] Pinheiro E, Bianchini R, Energy Conservation Techniques for Disk Array-Based Servers, *Proceedings of the 18th Annual ACM International Conference on Supercomputing (ICS'04)*, June 2004.

[21] Radcliff D, Guarding the data warehouse gate, *Computerworld*, October 2001. http://www.computerworld.com/industrytopics/financial/story/0,10801,64307,00.html

[22] RFC-3174 - US Secure Hash Algorithm 1, 2001. http://www.faqs.org/rfcs/rfc3174.html

[23] SANtools, Inc., 2005. http://www.santools.com/smartmon.html

[24] Santry DS, Feeley MJ, Hutchinson NC, Veitch AC, Carton RW, Ofir J, Deciding when to forget in the Elephant File System, *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, 1999.

[25] Xu R, Wang A, Kuenning G, Reiher P, Popek G, Conquest: Combining Battery-Backed RAM and Threshold-Based Storage Scheme to Conserve Power, Work in Progress Report, *19th Symposium on Operating Systems Principles (SOSP)*, October 2003.

[26] Yu X, Gum B, Chen Y, Wang R, Li K, Krishnamurthy A, Anderson T, Trading Capacity for Performance in a Disk Array, *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.

[27] Zhu Q, David FM, Devaraj C, Li Z, Zhou Y, Cao P, Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management, *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, February 2004.

[28] Zhu Q, Shanker A, Zhou Y, PB-LRU: A Self-Tuning Power Aware Storage Cache Replacement Algorithm for Conserving Disk Energy, *Proceedings of the 18th Annual ACM International Conference on Supercomputing (ICS'04)*, June 2004.

BIOGRAPHICAL SKETCH

Charles O. Weddle III

Charles O. Weddle III was born in Madison, Wisconsin, in 1971. He spent his childhood in Indiana. In spring of 1994, he completed his Bachelors of Science degree in Computer Science from Miami University. Charles spent eight years working in the industry as a software engineer before returning to Florida State University to obtain his Masters of Science degree in Computer Science in summer of 2005. Charles will begin work on his PhD in Computer Science in the fall of 2005. Charles is a member of the Association for Computer Machinery and the Institute of Electrical and Electronic Engineers. Charles is also a member of the Computer Science academic honorary Upsilon Pi Epsilon. Charles's other interests include running, golf, and history.