Energy Efficient Pre-Fetching - Models to Implementation

by

Adam C. Manzanares

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 14, 2010

Keywords: Storage Systems, Pre-fetching, Virtual File System

Approved by:

Xiao Qin, Chair, Assistant Professor of Computer Science and Software Engineering
David Umphress, Associate Professor of Computer Science and Software Engineering
Wei-Shinn Ku, Assistant Professor of Computer Science and Software Engineering
Yan Gu,Assistant Professor of Computer Science and Software Engineering

Abstract

With the rapid growth of the production and storage of large scale data sets it is important to investigate methods to drive the cost of storage systems down. We are currently in the midst of an information explosion and large scale storage centers are increasingly used to help store generated data. There are several methods to bring the cost of large scale storage centers down and we investigate a technique that focuses on transitioning storage disks into lower power states. To achieve this goal this dissertation introduces a model of disk systems that leverages disk access patterns to prefetch popular sets of data to produce energy saving opportunities.

Using our model, we have developed a simulator that allows us to quickly change various parameters to investigate the relationship that file access patterns, disk energy parameters, and simulation parameters have on the overall energy efficiency of disk systems. To help improve the validity of our simulation results we leveraged the validated disk simulator, DiskSim, and added disk power models to DiskSim. This allowed us to test our energy efficient strategies with a validated storage system simulator.

The last part of this dissertation focuses on implementing a large scale storage system virtual file system. We introduce the Energy Efficient Virtual File System, or EEVFS, to manage the data placement and disk states in a cluster storage system. Our modeling and simulation results indicated that large data sizes and knowledge about the disk access pattern are valuable for storage system energy savings techniques. Storage servers that support applications that stream media is one key area that would benefit from our strategies. The last chapter of the dissertation introduces

the concept of parallel striping groups, which attempt to improve the performance of EEVFS while maintaining energy savings.

Acknowledgments

I would like to thank and extend my gratitude to Dr. Xiao Qin who served as my advisor during the duration of my graduate studies at Auburn University. I met Dr. Qin as an undergrad at New Mexico Tech and he guided me in a research project as an undergrad. I went off to the University of Colorado and when Dr. Qin was hired at Auburn he recruited me into his research group at Auburn. Dr. Qin has spent many hours guiding and mentoring me and I am sincerely grateful for the experience he has provided me as my advisor. His hard work and dedication to the academic field will serve as an important example of what a successful academic can achieve.

I would also like to thank Dr. David Umphress because he has served as my secondary advisor and I have enjoyed the feedback and encouragement that he provides. I believe that his experiences and suggestions have proved to be strong influences in my early academic career. I would also like to thank Dr. Yan Gu and Dr. Wei-Shinn Ku for serving on my dissertation committee and providing insightful feedback when needed. I want to thank the faculty and staff of the CSSE department as they have guided me through this process. Jo-Ann Lauranitis has helped me greatly with the administrative process and I am thankful for her quick responses to all of my questions. I also would like to extend my thanks to Dr. Shiwen Mao for serving as the outside reader on short notice for this dissertation.

Our research group has been supportive and helpful all of the years I have been at Auburn University. I would like to thank Xiaojun Ruan because we have worked together on several projects and he has helped me meet several deadlines. I would also like to thank Shu Yin for providing me feedback and support during the dissertation process. Ziliang Zong and Kiranmai Bellam are two research group members who

have helped me greatly during the first years of my stay at Auburn. I would also like to thank Zhiyang Ding, Yun Tian, Jiang Xie, and James Majors for the help during the last stages of my dissertation work.

I would like to acknowledge my parents Jose and Margaret Manzanares because they have served as the greatest inspiration in my life. Their climb through the journey of life is an amazing feat and I am thankful for their endless support through any situation that I have faced. I also would like to thank my sister Anna because she has served as inspiration for me to pursue a fulfilling career.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to current trends in computing we are facing the so called data explosion. As the use of computers to help day-to-day tasks has increased, we also face a side effect of generating large amounts of data. This data must be stored on some sort of medium and currently hard disk drives have become the most common storage medium. Large scale storage systems are being developed and installed routinely and there is a significant amount of energy that must be consumed to operate these storage systems. There are many different methods to conserve energy and we have identified that hard disk drives consume a significant amount of the energy in a large scale storage system. To help alleviate the energy burden of hard disk drives we propose a strategy to manage to states of the hard disk drives.

This chapter continues by developing the problem statement clearly in Section 1.1. Section 1.2 presents the scope of the research Section 1.3 summarizes the main contributions of the dissertation. Finally Section 1.4 outlines the organization of the dissertation.

1.1 Problem Statement

The number of large-scale parallel I/O systems is increasing in today's high-performance data-intensive computing systems due to the storage space required to contain the massive amount of data. Typical examples of data-intensive applications requiring large-scale parallel I/O systems include; long running simulations [9], remote sensing applications [33] and biological sequence analysis [12]. As the size of a parallel I/O system grows, the energy consumed by the I/O system often becomes a large part

of the total cost of ownership [26][35][37]. Reducing the energy costs of operating these large-scale disk I/O systems often becomes one of the most important design issues. It is known that disk systems can account for nearly 27% of the total energy consumption in a data center [15]. Even worse, the push for disk I/O systems to have larger capacities and speedier response times have driven energy consumption rates upward.

Reducing energy consumption of computing platforms has become an increasingly hot research field. Green computing has recently been targeted by government agencies; efficiency requirements have been outlined in [8]. Large-scale parallel disks inevitably lead to high energy requirements of data-intensive computing systems due to scaling issues. Data centers typically consume anywhere between 75 W/ft2 to 200 W/ft2 and this may increase to 200-300 W/ft2 in the near future [1][43] These large-scale computing systems not only have a large economical impact on companies and research institutes, but also produce a negative environmental impact. Data from the US Environmental Protection Agency indicates that generating 1 kWh of electricity in the United States results in an average of 1.55 pounds (lb) of carbon dioxide (CO2) emissions. With large-scale clusters requiring up to 40TWh of energy per year at a cost of over $ 4B it is easy to conclude that energy-efficient clusters can have huge economical and environmental impacts [4].

Figure 1.1 presents the future energy use predictions of server and data centers. It presents several trend lines each corresponding to a scenario. According to historical trends the energy usage is going to increase at a greater than linear rate. There are also several trend lines that can improve the energy efficiency of server and data centers. Our goal is to help drive the server and data center usage down using new techniques, which will bring us closer to the state of the art scenario trend line.

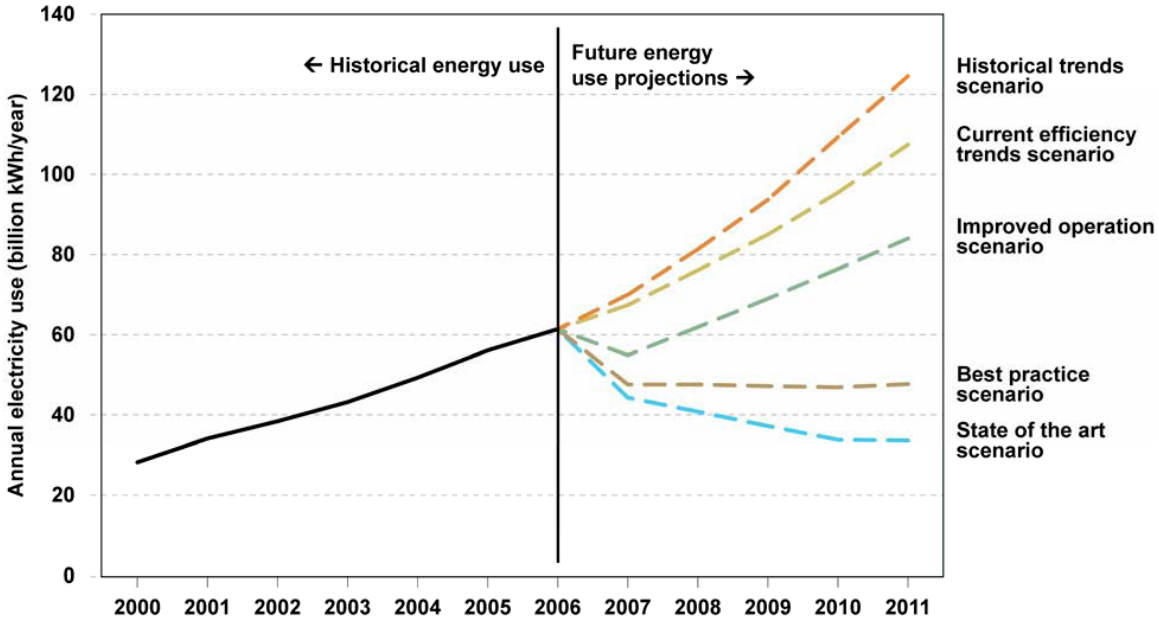


Figure 1.1: EPA Report to Congress on Server and Data Center Efficiency, 2007

1.2 Research Scope

Our research focuses on methods to lower the disk energy consumption in storage systems. The main goal of our research is to try and place many disks into the standby state to conserve energy. To place disks into the standby state they must have large periods of inactivity. To help produce periods of inactivity for a disk we propose prefetching popular data and move this data into a buffer disk. We are attempting to skew the workload to a subset of disks, which will allow us to place lightly loaded disks into the standby state.

This work is accomplished through the use of models and simulations. We present two models to help us model the energy efficiency of disk systems. We model the disk serving requests and also the state transition changes and their impact on the performance of the disk system. Using these models we developed our own simulator which we used to test many parameters of the model quickly. Our simulator was augmented by making changes to the DiskSim simulation environment and also developing a file

system simulator. Finally we develop a prototype implementation of a virtual file system that supports our energy efficiency strategies and also develop a data layout technique that improves performance and also energy efficiency.

1.3 Contributions

The major contributions of the research presented in this dissertation follows:

1. A motivational example is presented along with the development of a disk model that is used to gather preliminary simulation results.

2. An advanced disk system model is developed along with two algorithms that aid in conserving energy while maintaining acceptable response times.

3. The DiskSim simulation environment is extended to support disk energy models and a simulated file system is developed to interact with DiskSim.

4. A virtual file system prototype, EAVFS, is developed to put our modeling and simulation results into practice.

5. An energy efficient data layout, striping groups, is presented and implemented.

1.4 Dissertation Organization

This dissertation is organized in the following manner:

Chapter 2 introduces related work that is briefly reviewed and contrasted against the contributions of this dissertation.

Chapter 3 provides the motivational work for the rest of the dissertation. An example scenario is presented where the use of prefetching and buffer disks is highlighted. Also a simple mathematical model is presented and simulation results based on the model are presented.

Chapter 4 introduces advanced models for the modeling of disk requests and energy states of disks. We also introduce advanced algorithms that take into account the response time to prevent major delays. Thorough simulation results are also presented in this chapter.

Chapter 5 details how improvements were made to our simulation framework to improve simulation results. Changes were made to the DiskSim simulator to support disk energy states and a simple file system simulator is used to collect simulation results.

Chapter 6 introduces the Energy Aware Virtual File System (EAVFS), which is a prototype virtual file system that I developed to implement some of the ideas that were developed using models and simulation techniques.

Chapter 7 presents parallel striping groups that are implemented in EAVFS. These groups are introduced to help maintain performance while still providing energy savings.

Chapter 8 summarizes the main contributions of this dissertation and presents a couple of future research directions based on the ideas contained in the dissertation.

Chapter 2

Literature Review

2.1 Energy Efficient Disk Systems Related Work

2.1.1 Strengths and Limitations of Related Work

Almost all energy efficient strategies rely on DPM techniques [2]. These techniques assume a disk will have several power states. Lower power states have lower performance, so the goal is to place a disk in a lower power state if there are large idle times. There are several different approaches to generate larger idle times for individual disks. There are also several approaches to prefetch data, although many techniques have focused on low power disks.

1. Memory cache techniques - Energy efficient prefetching was explored by Papathanasiou and Scott [24]. Their techniques relied on changing prefetching and caching strategies within the Linux kernel. PB-LRU is another energy efficient cache management strategy [41]. This strategy focused on providing more opportunities for underlying disk power strategies to save energy. Flash drives have also been proposed for use as buffers for disk systems [5]. Energy efficient caching and prefetching in the context of mobile distributed systems has been studied [30] [42]. These three research papers focus on mobile disk systems, whereas we focus on large scale parallel disk systems. All the previously mentioned techniques are limited in the fact that caches, memory, and flash disk capacities are typically smaller than disk capacities. We propose strategies that use a disk as a cache to prefetch data into. The break-even times of disk drives

are usually very high and prefetch data accuracy and size become a critical factor in energy conservation.

2. Multi-speed/low power disks - Many researchers have recognized the fact that large break-even times limit the effectiveness of energy efficient power management strategies. One approach to overcome large break-even times is to use multi-speed disks [31] [39]. Energy efficient techniques have also relied on replacing high performance disks with low energy disks [4]. Mobile computing systems have also been recognized as platforms where disk energy should be conserved [5][17]. The mobile computing platforms use low power disks with smaller break-even times. The weakness of using multi-speed disks is that there are no commercial multi-speed disks currently available. Low power disk systems are an ideal candidate for energy savings, but they may not always be a feasible alternative. Our strategies will work with existing disk arrays and do not require any changes in the hardware.

3. Disk as cache - MAID was the original paper to propose using a subset of disk drives as cache for a larger disk system [6]. MAID designed mass storage systems with the performance goal of matching tape-drive systems. PDC was proposed to migrate sets of data to different disk locations [26]. The goal is to load the first disk with the most popular data, the second disk with the second most popular data, and continue this process for the remaining disks. The main difference between our work and MAID is that our caching policies are significantly different. MAID caches blocks that are stored in a LRU order. Our strategy attempts to analyze the request look-ahead window and prefetch any blocks that will be capable of reducing the total energy consumption of the disk system. PDC is a migratory strategy and can cause large energy overheads when a large amount of data must be moved within the disk system. PDC also

requires the overhead of managing metadata for all of the blocks in the disk system, whereas our strategy only needs metadata for the blocks in the buffer disk.

2.1.2 Observations

With the previously mentioned limitations of energy efficient research we propose a novel prefetching strategy. Our research differs from the previous research on the following key points.

1. We develop a mathematical model to analyze the energy efficiency of our prefetching strategy. This mathematical model allows us to produce simulations that offer insights into the key disk parameters that effect energy-efficiency.

2. We develop a prefetching strategy that tries to move popular data into a set of buffer disks without affecting the data layout of any of the data disks. We also perform simulations with parallel I/O intensive applications, which previous researchers have avoided.

Our strategies also have the added benefit of not requiring any changes to be made to the overall architecture of an existing disk system. Previous work has focused on redesigning a disk system or replacing existing disks to produce energy savings. Our strategy will either add extra disks or use the current disk system to produce energy savings under certain conditions.

2.2 DiskSim Related Work

The DiskSim simulator is a powerful tool for the modeling and simulation of disk systems and is used frequently for storage systems research [3]. Recent research projects based on the DiskSim simulation environment include reducing disk I/O performance sensitivity and conserving energy in disk systems [23][34]. Although

DiskSim is a powerful simulation tool research, projects have recognized the lack of power models as a limitation of DiskSim.

The Sensitivity-Based Optimization of Disk Architecture introduced accurate power models into the DiskSim environment, but there work was based on DiskSim 2.0 [29]. The other major paper to publish research related to DiskSim and power models is the Dempsey paper, which we were unable to obtain a copy of the source code [38]. Due to these limitations of previous research projects implementing power models it was decided to develop our own power models for the DiskSim 4.0 simulation environment.

2.3 Virtual File Systems and Striping Related Work

Cluster file systems are fast becoming a necessity due to the growth of cluster computing for high performance computing and web applications. Lustre is a popular clustering file system that has been designed for high performance [13]. The Parallel Virtual File System, PVFS, is an alternative high performance cluster file system that resides entirely in user space [20]. The two preceding file systems are designed for high performance applications and were designed with no energy efficient considerations. BlueFS is a distributed file system that was designed with energy efficiency considerations, but it focuses on mobile computing systems [22].

Due to the limitations of previous research we intend to develop an energy efficient cluster file system, EEVFS which is presented in Chapter 6. EEVFS manages disk states and data placement to reduce the energy consumption of disk system. EEVFS also uses striping groups to help to improve the performance of the storage system and the striping concept is outlined in [25]. Striping groups for EEVFS are explained in detail in Chapter 7.

Chapter 3

Preliminary Prefetching Scheme for Energy Conservation in Parallel Disk Systems

The objective of this study is to use the novel parallel disk architecture with buffer disks (see [43] and Figure 3.1) to aid in the reduction of the power consumption of large-scale parallel disk systems. To fully utilize buffer disks while aggressively placing data disks into low-power modes, we investigate an energy-aware prefetching mechanism (PRE-BUD for short) to dynamically fetch the most popular data into buffer disks. PRE-BUD attempts to prefetch data into the buffer disk with the desired consequence of reducing the total energy consumption of the parallel disk system. This work aims at designing two prefetching strategies using the PRE-BUD mechanism. Specifically, the first approach using PRE-BUD adds an extra disk, which performs as a buffer disk. The second approach uses an existing disk in the system as a buffer disk. The design of these two strategies relies on the fact that in a wide variety of data-intensive applications (e.g., web applications) a small percentage of the data is frequently accessed [18]. The goal of this research is to move this small amount of frequently accessed data from data disks into buffer disks, thereby allowing data disks to switch into low-power modes. Apart from energy efficiency, this work is focused on improving the reliability of parallel disk systems using traditional dynamic power management policies. We conduct experiments to confirm that the reliability of parallel disks can be enhanced through the buffer disk mechanism by reducing the number of state transitions. This research offers the following contributions. First, we create a mathematical model for large-scale parallel disk systems with buffer disks. Second, we develop two energy-efficient prefetching strategies in the context of the buffer disk architecture. Third, we quantitatively compare both of our prefetching

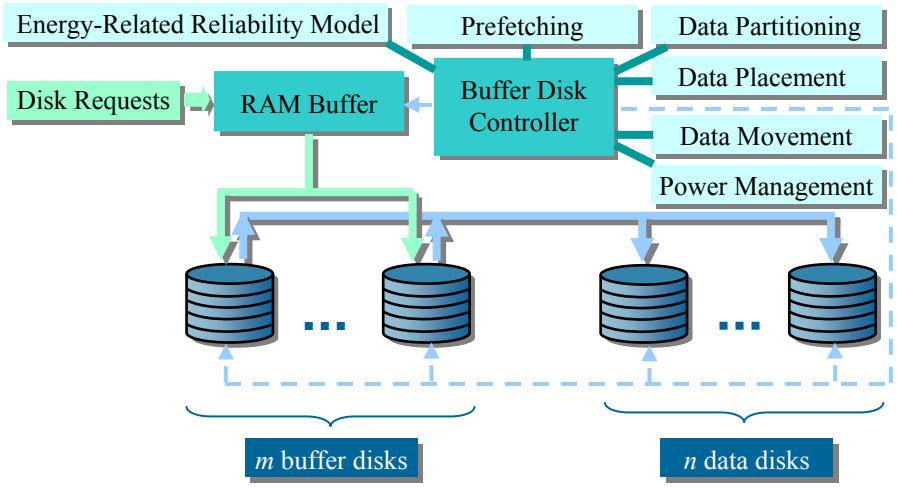


Figure 3.1: The buffer-disk architecture for parallel disk systems

approaches against two existing schemes including a dynamic power management technique and a non-energy-aware strategy. Fourth, we implement a simulator based on the mathematical model. Fifth, we use the simulator to quantitatively show that adding an extra buffer disk can substantially improve the reliability and energy savings of parallel disk systems without compromising storage capacity. Finally, we observed from experimental results that using an existing disk as a buffer disk compromises the storage space, but there is not the extra energy consumption and cost penalty associated with adding an extra disk i.e., the buffer disk. The rest of the chapter is organized as follows. Section 3.1 presents a motivational example and describes the mathematical model used for the purposes of this research. Section 3.2 describes our pre-fetching algorithm PRE-BUD. Section 3.3 presents simulation results and provides a discussion of the results. Section 3.4 is the conclusion of the chapter and future research directions are discussed.

3.1 Motivational Example

Our motivational example is based on the synthetic disk trace presented in Table 3.1. The requests all have the size of 275MB. This means each request will take approximately 5s to complete. This length was chosen, so seek and rotational delays would be negligible. There are N=4 disks used in this example, where each disk is given a unique letter. Each disk has two different data sections requested multiple times throughout the example. Each disk is modeled after the IBM 36Z15 using the parameters shown in Table 3.2 [4]. The example demonstrates only large sequential reads, which is a case that emphasises the benefits of our energy efficient strategies. This was also chosen to simplify our example to allow us to demonstrate the potential benefits of our approach. We also assume that all the data can be buffered which causes a small percentage of data to be accessed 100% of the time. This is only used for our motivational example and our simulation results vary this parameter to model real-world conditions. We also assume these strategies can be handled off-line meaning we have prior knowledge of the complete disk request pattern.

Time	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
Block	A1	A2	B1	B2	D1	D2	C1	C2	B2	B1	A1	A2	C1	C2	D1	D2	B2
Time	85	90	95														
Block	B1	A1	A2														

Table 3.1: Synthetic Trace

X = Transfer Rate = 55 MB\s	P_{Act} = Power Active = 13.5 W
P_{Idle} = Power Idle = 10.2 W	P_{Stdby} = Power Standby = 2.5 W
E_{AS} = Energy Active to Sleep = 13.0 J	E_{SA} = Energy Sleep to Active = 135 J
T_{AS} = Time Active to Sleep = 1.5 s	T_{SA} = Time Sleep to Active = 10.9 s

Table 3.2: Disk Parameters (IBM36Z15)

For a base line comparison we use a non-energy aware approach. This approach puts a disk in the active state if a request for this particular disk is received. If the

disk is not serving a request it remains in the idle state. We also compare our two approaches against an energy aware approach. The energy aware approach puts a disk into the standby state if the energy savings achievable is greater than the energy required to complete the state transition. The two different approaches we present are adding a buffer disk to the disk system or the use of an existing disk as the buffer disk. Following is an explanation of the mathematical model used to produce the results for our motivational example.

$$T_{E(Y)} = (T_{Act(Y)} * P_{Act}) + (T_{Idle(Y)} * P_{Idle}) + (T_{Sleep(Y)} * P_{Stdby}) + E_{Trans(Y)} \quad (3.1)$$

$T_{E(Y)}$ is the total energy Disk Y consumes serving the trace.

$$T_{Act(Y)} = \sum_{i=j}^{k} (len(\mathrm{Req}[i]))/X) \quad (3.2)$$

where $T_{Act(Y)}$ is the total time that $DiskY$ is active. It is a summation over all the requests involving $DiskY$. It starts with the first request for $DiskY(j)$ and ends with the last request for $DiskY(k)$. For the motivational example all requests are of the same size, so the length of each request is 275MB. The transfer rate X is fixed at 55MB/s, so we know all requests take approximately 5s to process. When using the buffer disk a request for a buffered block only causes the buffer disk to be active.

$$T_{Idle(Y)} = \sum_{i=j}^{k} (len(\mathrm{Req}[i]))/X) \quad (3.3)$$

where $T_{Idle(Y)}$ is the total amount of time $DiskY$ is idle. The non-energy aware strategy places disks in the idle state if they are not serving a request. This ends up being a summation over all requests that are not for $DiskY$. This will start at the first request that is not for $DiskY(j)$ and end at the last request for $DiskY(k)$.

The following equation determines if a disk will be idle between requests in all other approaches.

$$BE_{Idle} = (E_{AS} + E_{SA})/P_{Idle} = 14.5s \tag{3.4}$$

where BE_{Idle} is the energy break even period. If there is an idle time that is larger than BE_{Idle} energy conservation can be achieved by putting the disk to sleep. If the idle time is less than BE_{Idle} than the energy penalty to transition between the idle state and standby is greater than the energy savings possible by putting the disk to sleep. $T_{Sleep(Y)}$ = The time that $DiskY$ is sleeping (in the standby state). In the non-energy aware strategy a disk is never put into the standby state. The energy efficient strategy we compare against forces a disk to go to sleep whenever $T_{Idle(Y)} > BE_{Idle}$. The disk will stay sleeping until the next request is for a block on that disk. It will wake up the disk with enough time to make T_{SA} in time for the request, which is 10.9 s. The approaches using a buffer disk also use this strategy. The only exception to this rule is when a request for $DiskY$ is contained in the buffer $DiskY$ is allowed to sleep longer times.

$$E_{Trans(Y)} = \sum_{i=1}^{m} E_{AS}[i] + \sum_{i=1}^{n} E_{SA}[i] \tag{3.5}$$

where $E_{Trans(Y)}$ is the total energy consumed for all state transitions for $DiskY$. In the non-energy aware approach this will be zero for all of the disks. In the energy aggressive approaches it is the summation over all active/sleep transitions added to the summation over all sleep/active state transitions for $DiskY$.

$$BD_{EPre} = \sum_{i=j}^{k} (len(\mathrm{Req}[i]))/X) * P_{ACT} \tag{3.6}$$

where BD_{EPre} is the total amount of energy consumed by the buffer disk pre-fetching data. It is a summation over all of the requests that are put into the buffer disk. This value is zero whenever a buffer disk is not used.

$$BD_{ETot} = \sum_{i=j}^{k} ((len(\mathrm{Req}[i]))/X) * P_{ACT}) + BD_{E\,\mathrm{Pr}\,e} \tag{3.7}$$

where BD_{ETot} is the total amount of energy the buffer disk consumes for the entire trace. It is the summation over all requests that are in the buffer disk and the addition of the energy consumed in the pre-fetch phase. This is also zero when a buffer disk is not used.

$$T_{ES} = \sum_{i=1}^{N} T_{E[i]} + BD_{ETot} \tag{3.8}$$

where T_{ES} is the total energy consumed by all disks used to serve the synthetic trace. This also includes buffer disk energy if a buffer disk is used.

$T_{Idle(A)}$	70s	$T_{Idle(B)}$	70s
$T_{Idle(C)}$	80s	$T_{Idle(D)}$	80s
$T_{Act(A)}$	30s	$T_{Act(B)}$	30s
$T_{Act(C)}$	20s	$T_{Act(D)}$	20s
$E_{Trans(A)}$	0J	$E_{Trans(B)}$	0J
$E_{Trans(C)}$	0J	$E_{Trans(D)}$	0J
$T_{E(A)}$	1119J	$T_{E(B)}$	1119J
$T_{E(C)}$	1086J	$T_{E(D)}$	1086J
T_{ES}	4410J		

Table 3.3: Non-Energy Aware Results

The results presented in Tables (3.3-3.6) gave us some promising initial results. The two approaches using a buffer disk provided significant energy savings over the non-energy aware parallel disk storage system. Table 3.5 presents the results using our approach that adds an extra disk to the system. This has the benefit of not impacting the capacity of the large-scale parallel disk system. The other main benefit

$T_{Idle(A)}$	0s	$T_{Idle(B)}$	10s
$T_{Idle(C)}$	0s	$T_{Idle(D)}$	0s
$T_{Act(A)}$	30s	$T_{Act(B)}$	30s
$T_{Act(C)}$	20s	$T_{Act(D)}$	20s
$T_{Sleep(A)}$	45.2s	$T_{Sleep(B)}$	33.7s
$T_{Sleep(C)}$	53.7s	$T_{Sleep(D)}$	53.7s
$E_{Trans(A)}$	296J	$E_{Trans(B)}$	309J
$E_{Trans(C)}$	309J	$E_{Trans(D)}$	309J
$T_{E(A)}$	814J	$T_{E(B)}$	900.25J
$T_{E(C)}$	713.25J	$T_{E(D)}$	713.25J
T_{ES}	3140.75J		

Table 3.4: Energy Aware Results

$T_{Idle(A)}$	0s	$T_{Idle(B)}$	0s
$T_{Idle(C)}$	0s	$T_{Idle(D)}$	0s
$T_{Act(A)}$	10s	$T_{Act(B)}$	10s
$T_{Act(C)}$	10s	$T_{Act(D)}$	10s
$T_{Sleep(A)}$	100s	$T_{Sleep(B)}$	100s
$T_{Sleep(C)}$	100s	$T_{Sleep(D)}$	100s
$E_{Trans(A)}$	13J	$E_{Trans(B)}$	13J
$E_{Trans(C)}$	13J	$E_{Trans(D)}$	13J
$T_{E(A)}$	398J	$T_{E(B)}$	398J
$T_{E(C)}$	398J	$T_{E(D)}$	398J
BD_{EPre}	540J	BD_{ETot}	1350J
T_{ES}	3140.75J		

Table 3.5: PRE-BUD Approach 1

of our first approach is the fact that state transitions are lowered as compared to the energy aware baseline.

The problem is that this approach consumes more energy than the energy aware disk management scheme. This is due to the extra energy associated with adding a disk. This extra disk must first pre-fetch all of the data into the buffer, and then serve all the buffered requests. In our strategy we are able to buffer all of the data causing the buffer disk to be active the entire trace. The buffer disk is also active pre-fetching data before the trace begins. Adding an extra disk also costs money and could introduce pricing concerns when this strategy is scaled to accommodate larger

$T_{Idle(A)}$	0s	$T_{Idle(B)}$	0s
$T_{Idle(C)}$	0s	$T_{Idle(D)}$	0s
$T_{Act(A)}$	140s	$T_{Act(B)}$	10s
$T_{Act(C)}$	10s	$T_{Act(D)}$	10s
$T_{Sleep(A)}$	0s	$T_{Sleep(B)}$	100s
$T_{Sleep(C)}$	100s	$T_{Sleep(D)}$	100s
$E_{Trans(A)}$	0J	$E_{Trans(B)}$	13J
$E_{Trans(C)}$	13J	$E_{Trans(D)}$	13J
$T_{E(A)}$	1890J	$T_{E(B)}$	398J
$T_{E(C)}$	398J	$T_{E(D)}$	398J
BD_{EPre}	540J	BD_{ETot}	1350J
T_{ES}	3084J		

Table 3.6: PRE-BUD Approach 2

Repeats	1	2	3	4	5
Non-energy Aware	4410	8820	13230	17640	22050
Energy Aggresive Approach	3141	6294	9447	12600	15753
PRE-BUD 1	3482	5832	8182	10532	12882
PRE-BUD 2	3084	5184	7284	9384	11484
Repeats	6	7	8	9	10
Non-energy Aware	26460	30870	35280	39690	44100
Energy Aggresive Approach	18906	22059	25212	28365	31518
PRE-BUD 1	15232	17582	19932	22282	24632
PRE-BUD 2	13584	15684	17784	19884	21984

Table 3.7: Trace Repeat Results (J)

systems. This is what led us to try a second approach, which uses an existing disk as a buffer disk. For this experiment it was assumed $DiskA$ could be the buffer disk. This had the desired effect of bringing the energy consumption total below the energy aware strategy. The major negative of using an existing disk as the buffer disk is that the capacity of your disk storage system is decreased.

The potential energy savings gains are realized once the sample trace used for our motivational example is repeated. This results in huge energy savings for both of our approaches over the non-energy aware and energy aware approaches. When the trace is repeated 10 times adding a buffer disk to your parallel disk system could potentially

save 44% of the energy cost of a non-energy aware approach. Similarly using an existing disk for a buffer disk saves 50% of the energy cost. When comparing against the energy aware approach our first approach is able to save 22% and the second approach is able to save 30% of the energy. The energy savings of the energy aware approach is already considerable when compared to the non-energy aware strategy. Our approaches are able to produce a significant amount of extra energy savings over the energy aware approaches. This is due to the fact that using a buffer disk allows each disk the ability to be in the standby state for longer periods of time. The standby times a disk experiences between requests becomes cumulative if the requests are contained in the buffer disk. This is due to the fact that the request can be served from the buffer disk, so the requested disk does not have to transition to the active state.

It is important to note that this is the ideal case for our buffer disk framework. This represents a situation where we are able to buffer all of the data requested and all requests are large reads. These assumptions were only used to come up with some upper bound estimates for the potential energy savings. In our simulation results we tried to choose parameters that would more closely model real-world applications and settings.

For reliability concerns, we also decided to document the number of state transitions each disk experiences using a disk management scheme. Figure 3.2 represents the state transitions for each disk using the energy aware disk management scheme. The non-energy aware scheme has zero state transitions, since it never tries to put a disk in the standby state. The buffered disk approaches only needs one state transition per disk. This state transition is experienced after the disk is put into the standby state after being active delivering data to the buffer. After this the data transfer is complete and the disk is put into the standby state.

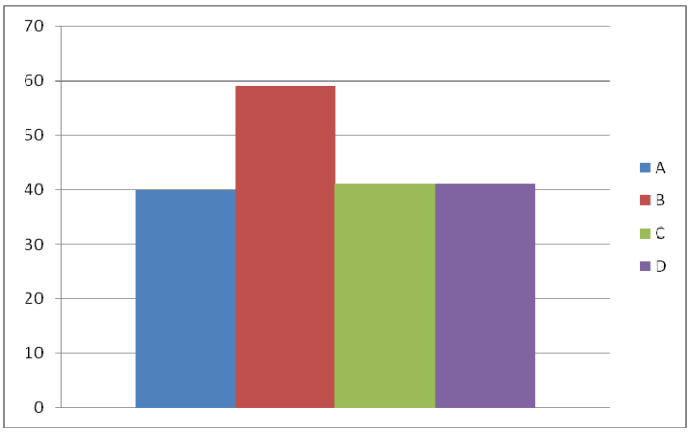


Figure 3.2: Disk State Transitions Energy Aware

Reliability is assumed to be a factor of state transitions. As the number of state transitions increases the reliability of the disk system goes down. This is due to the demands that changing states places on a hard drive. Figure 3.2 shows the state transitions for each disk using the energy aware approach. This approach does not consider the reliability of the disk as a factor and schedules a state transition whenever the disk has idle times greater than a threshold. The buffered disk approaches improves reliability by moving frequently accessed data into a single point. This can save a sleeping disk from having to be placed in the active state causing a state transition. In the case that all the data needed for an application can be placed in the buffer disk, all the other disks can sleep the entire life of the application. They only need to be active to move the requested data into the buffer before the application is executed.

3.2 Energy-Efficient Prefetching Strategy

Figure 3.3 outlines the algorithm used to pre-fetch blocks and how the pre-fetched blocks are used using the PRE-BUD strategy. The pre-fetch algorithm uses the frequency that a block is requested as a heuristic. The first step of the algorithm

```
1.  request[] ;          /*  request[]  holds all of the requests in the trace */
2.  i=0;
3.  while(not all requests have been processed)  /* Iterate over all requests */
4.     if(request[i].block  has been seen before)  /* Requested block has been seen */
5.        list.inc_ref(request[i].block)       /* Add one to the count for this block */
6.        i++;
7.     elseif(request has not been seen before)   /*Requested block has not been seen before*/
8.        list.add(request[i].block) /*Add the request to a list and increment its reference */
9.        i++;
10.
11. buffer[];       /* Buffer to hold requested blocks
12. Buffer Size=N;
13. i=0;
14. list.sort()       /* Sort the requests by their reference counts */
15. while(i<N)
16.    buffer[i]=list(i)  /* Move the frequently requested data into the buffer */
17.
18. i=0;
19. distance=0;   /* Distance between two requests on the same disk */
20. while(not all requests have been processed)
21.     if(request[i].block is in Buffer)
22.        buffer_Disk(request[i]);
23.       distance+=request[i].time + calc_distance(request[i]);
24.    elseif(request[i].block is not in Buffer)
25.       distance+=calc_distance(request[i]);
26.        process(request[i]);
27.        can_sleep(distance);
28.        distance=0;
29.   can_sleep(distance)
```

Figure 3.3: The energy-efficient prefetching strategy or PRE-BUD

iterates over all of the requests and counts the references for each unique block. Then it sorts the list of unique blocks by the number of references to each block. At this point the algorithm puts the highly requested blocks into the buffer until it is full. The last part of the algorithm also iterates over all requests trying to figure out how long each disk can sleep. If a block requested for a disk is in the buffer the disk can sleep longer. The buffer disk handles the request and the distance between requests on the same disk becomes cumulative. If a requested block is not in the buffer the disk must be woken up to serve the request, this is handled by the process function. The distance is then set to zero since the disk had to be woken up. Using the frequently accessed heuristic the PRE-BUD strategy should have a small performance impact on the system. Almost all steps of the algorithm run linearly with respect to the number of requests. The only step that is not linear is the phase that sorts the list of requests according to their frequency. Sorting is a common procedure and is known to have a best-case run-time of nlogn. The PRE-BUD strategy is able to have a run time of $n + nlogn$ using an efficient sorting algorithm. The PRE-BUD strategy is not assumed to be optimal, since the requested blocks are sorted using their frequency. The frequency is used as a heuristic to select blocks to be placed in the buffer. An optimal strategies goal would be to select the requests to be placed in the buffer that produce the largest impact on the standby time of disks. The largest increases in standby times for disks result in the largest energy savings gains.

3.3 Simulation Results

For our simulation results it was decided to increase the synthetic trance length to 200 requests. This would increase the total time of that the simulation was ran and give our buffered disk strategies room to increase idle times for disks. The first parameter tested is the hit rate of the buffer disk. For our motivational example 100% of the data requested was placed into the buffer. This results in a hit rate of 100%.

To more accurately model applications that access 20% of the data available 80% of the time we lowered the buffer disk hit rate. This was implemented by increasing the number of requests not in the buffer by 5 until the hit rate was lowered past 80%. We are assuming that the buffer disk will be able to hold all of the frequently accessed data. All energy results are in Joules.

Number of Disks	Value
Number of Disks	4,5,6,7,8,9,10
Block Size	275,225,175,125,75,25,10,5 MB
Hit Rate	100,97,5,92.5,90,87.5,85,82.5,80,77.5,75%
Transfer Rate	55 MB/s
Number of Requests	200

Table 3.8: Simulation Parameters

The results displayed Figure 3.4 held the number of disks to 4 and also kept the data size of each request at 275MB. We have omitted the results of the non-energy aware approach, since they are constant and higher than the energy aware strategy. As expected the performance of both of our strategies were lowered when the hit rate was decreased. This is expected since our motivational example demonstrated a best case scenario. Disk sleep times are lowered once a miss is encountered. This is due to the fact that a disk has to wake up to serve the request. This will increase the energy consumption of disks that have to serve the missed requests. This leads to an increase in the total energy consumption of the entire system. Our buffered large-scale parallel disk system is still able to consume less energy than the energy aware approach. The energy aware and non-energy aware disk systems are not affected by buffer disk miss rates.

The first buffer disk approach begins to approach the same level of performance as the energy aware strategy. It is only able to save 10% energy over the energy aware strategy when the hit rate is 75%. This is because adding the extra disk puts extra energy requirements on the system, and lowering the hit rate further impacts the

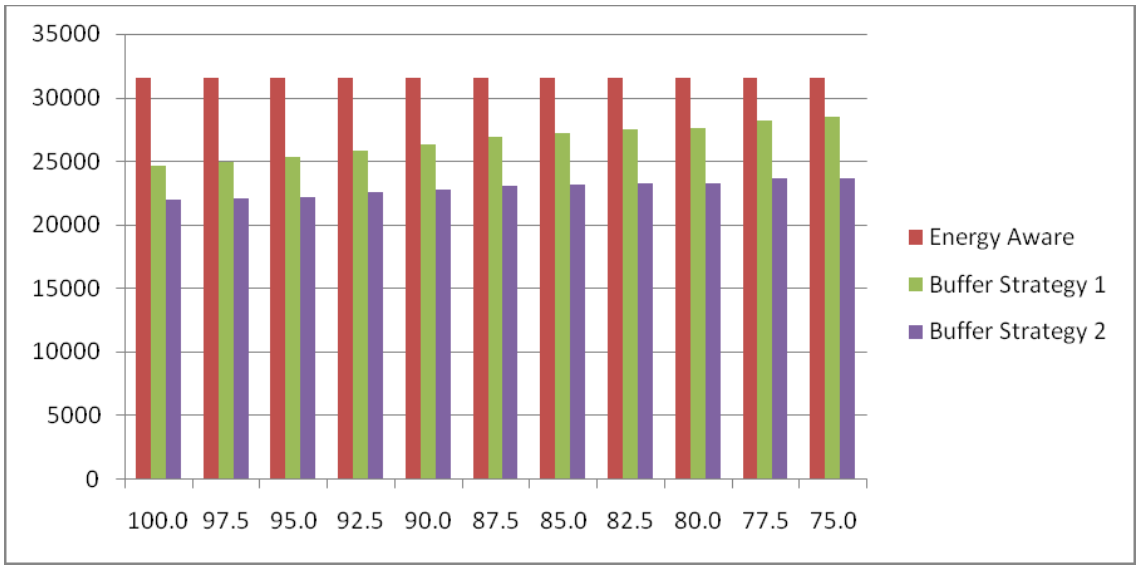


Figure 3.4: Impact of buffer disk hit rate

energy benefits of the first strategy. The second buffered disk approach is still able perform 25% better than the energy aware approach. This is because there is not the extra energy penalty of adding an extra disk. The capacity of your disk system will be lowered using this approach.

The hit rate becomes a very important factor in the performance of our approaches. If the buffer disk is constantly missing requests then both strategies will eventually downgrade to the energy aware approach. Fortunately applications have been documented to request 20% of the data available 80% of the time. Our heuristic based approach would work considerably well in this case. This is modeled by the 80% hit rate. The buffered disk approach one and two are able to save 12% and 26% energy over the energy aware strategy when the hit rate is 80%. Similarly, they are able to save 37% and 47% of the total energy compared to the non-energy aware approach.

The first buffer disk approach downgrades more quickly than the second approach as the hit rate is decreased as compared to the energy aware approach. This is not

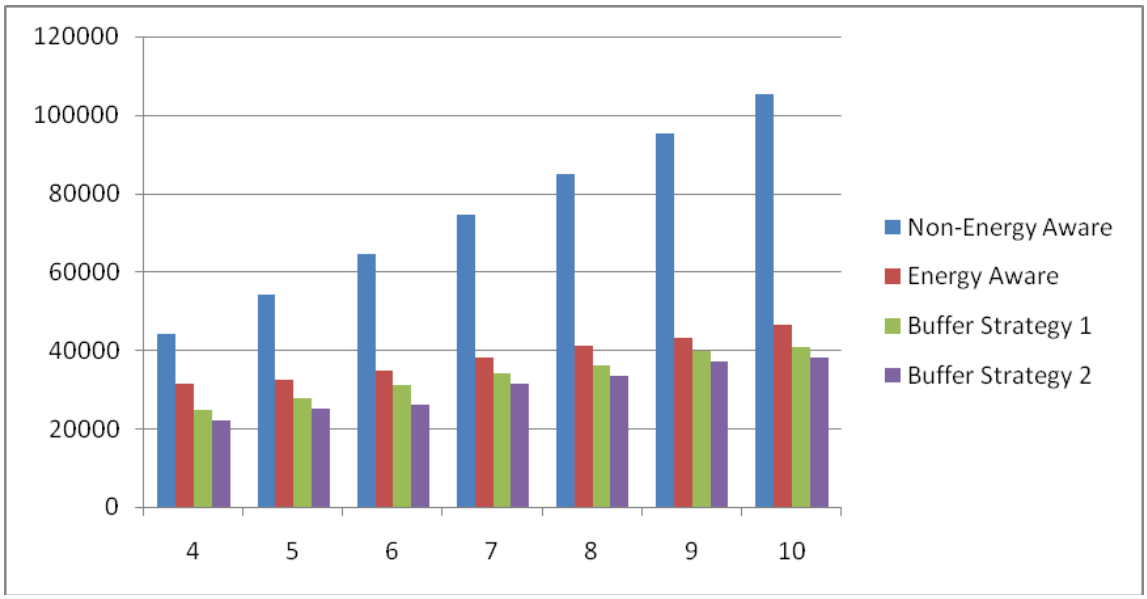


Figure 3.5: Impact of the Number of Disks

that great of a concern since the first strategy is still able to have a positive impact on the reliability of the disk system as compared to the energy aware approach. The first buffer disk approach is still able to produce significant energy savings over the non-energy aware approach without compromising the reliability of the system. The energy savings performance of the second approach does not diminish as quickly as the first approach, but there will be an impact on the capacity of the system. The second approach is also able to reduce the number of state transitions.

For the next set of experiments we increased the number of disks contained in the disk system. This was done to show the effect that scaling the amount of disks up would have on our system. We kept the size of the buffer constant. This had the effect of increasing the miss rate as disks were added to the system. If a disk is added to the system it demands more blocks to be placed into the buffer. Not all of these blocks can be placed into the buffer disk. As a consequence of this, the hit rate is lowered.

From Figure 3.5 we are able to see that the non-energy aware approach wastes a considerably larger amount of energy as compared to all of the energy aware approaches. This is expected since the non-energy aware approach is not able to place disks in the standby mode. Buffer strategy 1 was able to produce a 12% increase in energy savings over the energy aware strategy when 10 disks were simulated. Similarly, buffer strategy 2 performed even better with an 18% increase. This is expected again because of the energy overhead adding an extra disk buffer strategy 1 requires. Our approach produces promising results as the number of disks is increased. This is an important observation, since our target system is a large-scale parallel system. This leads us to believe our system will produce energy benefits regardless of the number of disks in a system. The number of buffer disks may be scaled with the size of the system. This work does not try to figure out a way to decide on the number of buffer disks needed for a large scale parallel disk system and leaves this for future research.

The last parameter varied to produce our simulation results was the data size of the requested blocks and is presented in Figure 3.6. One of the assumptions made in our motivational example included the disk system would only conduct large reads. This assumption was relaxed by lowering the data size of the requests. All data sizes are in MB and the energy outputs are in Joules. We kept the number of disks at four and the number of requests at 200. This meant that low data sizes would produce low amounts of energy because they would be processed quickly.

It was expected that the data size would have a greater impact on the performance of our buffer disk approach. This assumption was based on the fact that smaller data sizes decrease the time frames in which a disk is able to sleep. This is evident by looking at the results of the energy aware approach against the non-energy aware strategy. As the data size is lowered the energy aware strategy is no longer

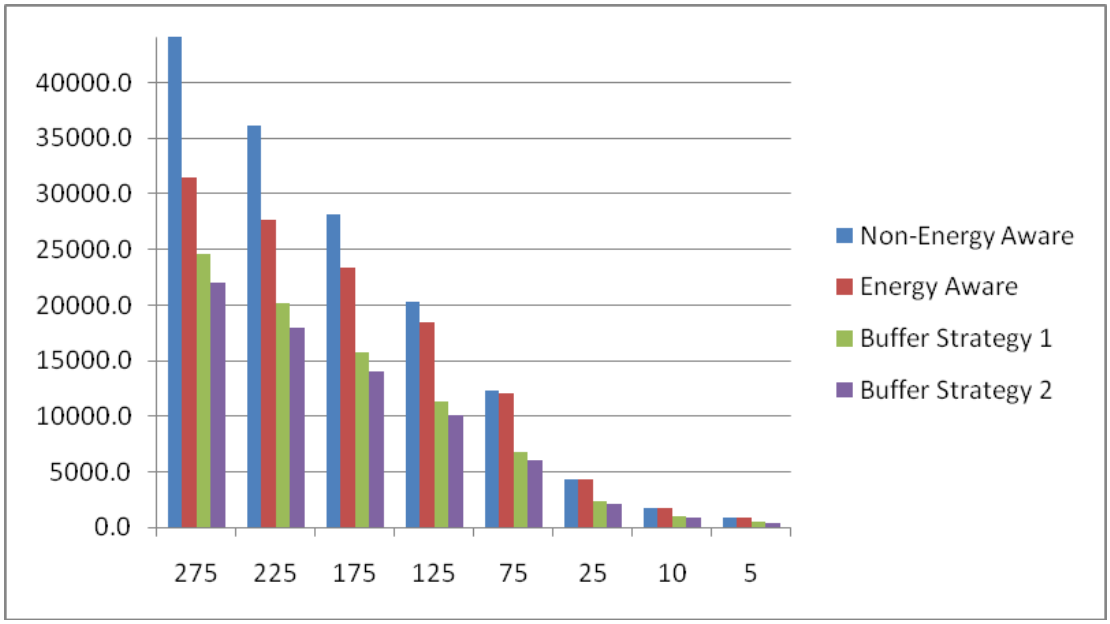


Figure 3.6: Impact of data size

able to save energy by putting a disk into the sleep state. It turns back into the non-energy aware strategy once the data size becomes 10MB or less. The buffered disk approaches were still able to perform better than the non-energy aware approach.

This is achieved by the fact that the buffer disk increased the time frames that a disk can sleep. By pre-fetching the frequently accessed data for a disk it increased the odds that the current request will be in the buffer disk allowing the disk corresponding to the request to increase its sleep time window. This window must be larger than BE_{Idle} or the energy penalty associated with the state transition will be greater than the energy that can be saved by sleeping the disk. These results become very promising since one of our original assumptions is that requests would be large reads. These results show that our approaches may also work for small reads.

3.4 Chapter Summary

The use of large-scale parallel disk systems continues to rise as the demand for information systems with large capacities grows. Large-scale parallel disk systems allow someone to combine smaller disks to achieve large capacities. The problem is that these large- scale disk systems can be extremely energy efficient. Disk systems can account for nearly 27% of the energy demands of a large-scale computing platform. The energy consumption rates are rising as disks become faster and disk systems are scaled up. Our goal was to increase the energy efficiency of a large-scale disk system using a buffer disk that would pre-fetch frequently accessed data. This had the effect of increasing idle periods for disks facilitating long sleep times. As sleep times are increased the disk is able to save energy over being in the idle state.

We proposed two different methods of using a buffer disk. The first strategy added an extra disk to the system and the second approach used an existing disk as the buffer disk. The first strategy consumes more energy because an extra disk is added, but does not compromise the capacity of the disk system. We compared our approaches against non-energy aware and energy aware approaches. The buffered disk approaches were both able to produce energy savings as compared to the non-energy aware and energy aware strategies. The energy savings was larger when compared against the non-energy aware strategy. Although our approaches did not save as much energy as the energy aware approach, they were both able to positively impact the reliability of the disk system. Both approaches were able to lower the number of state transitions as compared to the energy aware approach.

For the future research work we would like to work on adding more than one buffer disk to the system. The number of buffer disks will have to be increased as the scale of the disk system is increased. This will add the extra requirements parallel applications demand and our strategies will have to be modified to reflect these changes. Our work also used only synthetic traces for our simulation results.

We need to incorporate traces from real-world applications to improve the feasibility of our approaches. The reliability impacts of using the buffer disk can also be researched more heavily. We demonstrated the reduction in state transitions, but need to come up with a model that relates state transitions and reliability.

Chapter 4

Prefetching Models and Simulation

4.1 Motivational Example

For a simple motivational example that demonstrates the utility of the buffer disk architecture, we present a scenario that is depicted in Figure 4.1. Each horizontal bar represents the time a particular disk is busy or idle. Figure 4.1 presents requests for individual disks that are represented by the specific colors and patterns presented in the legend. Idle periods for all of the disks are represented with the orange color. If we are using the IBM 36Z15 disk for disks A, B, and C DPM techniques will not be able to save any energy. DPM requires a disk to have an idle period greater than the break-even time. For the IBM 36Z15 disk the break-even time is 14.5 seconds. The largest idle-period for any of the disks presented in Figure 4.1 is 8 seconds. This means that DPM is unable to save any energy is this example, even though there are idle periods of 8 seconds. The total energy consumed by all of the disks to serve all of the requests is approximately 949.2 Joules. Each disk must remain in the idle state, which consumes 10.2 W, when they are not serving a request.

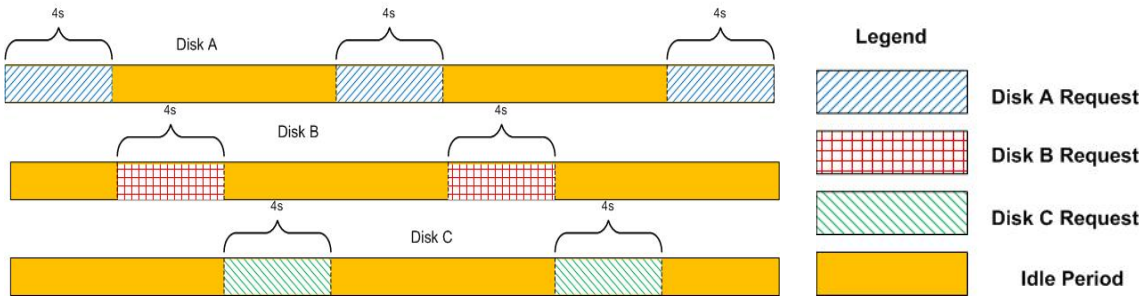


Figure 4.1: Sample Disk Trace

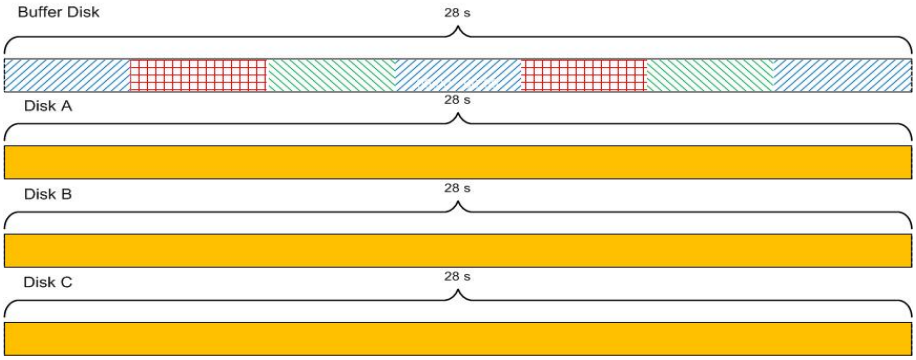


Figure 4.2: Buffer Disk Added to Architecture

If we were able to prefetch the requested data from all three disks into a single disk, which is represented by Figure 4.2, we could have one single disk do the work of the three disks. Disks A, B, and C will be put into the sleep state and remain in the sleep state for the entire length of the trace.

Using a buffer disk allows one to trade many lightly loaded disks, for a smaller number of heavily loaded disks. The key to energy savings using a buffer disk is to accurately place frequently requested data into the buffer disk. This allows non-buffer disks to have larger idle-window sizes as compared to not using a buffer disk. If a request can be served from a buffer disk, the corresponding data disk for this particular request treats the time for the buffer disk to serve the disk request as an extra idle window. The key to energy savings with the buffer disk strategy is to have consecutive hits from the perspective of a single disk, so the disk can see a long continuous idle window. Adding an extra buffer disk represents one of our approaches, PRE-BUD1, to conserving energy in parallel storage systems. This approach will consume 804 J, including the energy required to prefetch the data from all three disks. Similarly, if you used Disk A to prefetch requested data from Disk B and Disk C, Disk A would now become a buffer disk. Disk A would remain active for 28 s, while Disk B and Disk C would sleep for 28 s. This preceding approach, PRE-BUD2, will consume 680 J. PRE-BUD1 is able to save 15.3% and PRE-BUD2 is able to save 28.4% energy

over the DPM strategy. These numbers will go up if the trace presented in Figure 4.1 is repeated. This is because the requested blocks are already in the buffer disk and sleeping a disk is 4 times more energy efficient than leaving it in the idle state.

4.2 PRE-BUD Energy-Efficient Prefetching Strategy

In this section, we describe our energy-efficient prefetching strategy for parallel storage systems with buffer disks. Energy consumption in parallel disk systems can be reduced by placing idle disks into the standby state, which causes the idle disks to stop spinning completely. The fundamental goal of PRE-BUD is to improve energy efficiency of parallel disks through the following two energy saving principles. First, by reducing the number of power state transitions one can decrease the energy overhead of spinning down the disks. Second, increasing the number and lengths of standby intervals can foster new opportunities to aggressively turn disks into the standby state. PRE-BUD implements these two energy saving principles using the concept of buffer disks, which contain frequently accessed data blocks that are prefetched and buffered. There are two buffer disk architectures: (1) adding buffer disks to the disk system, PRE-BUD1, and (2) using existing disks as the buffer disk(s), PRE-BUD2. The energy-efficient prefetching strategy, PRE-BUD, described in this chapter can be successfully applied to deal with the two approaches to the architecture. In this study, let us first focus on parallel disk systems with a single buffer disk. Then, in Section 4.5 we briefly discuss how to extend PRE-BUD to conserve energy in parallel disk systems with multiple buffer disks.

The PRE-BUD strategy is a greedy algorithm in the sense that blocks fetched into a buffer disk in each prefetching phase (see Steps 8-11 in Figure 4.3) are the ones that have the highest energy savings, which in turn attempts to maximize the energy efficiency of the parallel disk system. PRE-BUD has two key components: the prefetching module and the energy-saving calculation module. Given a parallel

disk system with a buffer disk, the prefetching module determines which blocks to fetch from any of the parallel disks to improve the energy efficiency of the entire disk system. If the buffer disk is full while more blocks have to be fetched, the prefetching module is tasked with deciding which blocks need to be evicted. The prefetching module relies on the second module to calculate and update the energy savings of referenced blocks in the current look-ahead window and blocks present in the buffer disk. The energy savings estimate of a block in a data disk quantifies the energy consumption reduction produced by fetching the block into a buffer disk. On the other hand, the energy savings estimate of a block in the buffer disk reflects the energy savings value of caching the block instead of evicting it from the buffer disk. The prefetching and energy-saving calculation modules are detailed in Sections 4.2.1 and 4.2.2, respectively.

4.2.1 Prefetching Module

Before presenting the prefetching module of PRE-BUD, we first summarize the notation for the description of the prefetcher in Table 4.1.

Notation	Description
R	Current lookahead. $r \in R$ is a reference in the lookahead
$block(r)$	Block accessed in reference $r \in R$
$disk(r)$	Disk in which $block(r)$ is residing
A	Subset of the lookahead R; for any r in A, $disk(r)$ is active, i.e., $\forall r \in A$: $disk(r)$ is active
G	A set of blocks present in the buffer disk
$E_s(b)$	Energy saving contributed by prefetching block b
A^+	For any $b \in A^+$, we have $disk(b) \in A$, $E_s(b) > 0, b \notin G, and \exists r \in R : block(r) = b$
G^+	The set of blocks with the highest energy savings in $A^+ \cup G$

Table 4.1: Notation for the description of the prefetching module

Figure 4.3 outlines the prefetching module in PRE-BUD. PRE-BUD is energy-efficient in nature, because a request for data in a disk currently in the standby mode

will not have to be spun up to serve the request if the requested block is present in the buffer disk (see Step 4). Buffer-disk resident blocks allow standby data disks to stay in the low-power state for an increased period of time, as long as accessed blocks are present in the buffer disk. There is a side effect of making the buffer disk perform I/O operations while placing data disks in standby longer; that is, the buffer disk is likely to become a performance bottleneck. To properly address the bottleneck issue, we design the prefetcher in such a way that the load between the buffer and data disks is balanced, if the active data disk can achieve a shorter response time than the buffer disk we don't rely on the buffer disk (see step 2). In addition to load balancing, utilization control is introduced to prevent disk requests from experiencing unacceptably long response times. In light of the utilization control, the prefetching module ensures that the aggregated required I/O bandwidth is lower than the maximum bandwidth provided by the buffer disk (see Line 11.a in Figure 4.3).

To improve the energy efficiency of PRE-BUD, we force PRE-BUD to fetch blocks from data disks into the buffer disk on a demand basis (see Line 5 in Figure 4.3). Thus, block b is prefetched in Step 10 only when the following four conditions are met. First, a request r in the look-ahead is accessing the block, i.e.,$\exists r \in R : block(r) = b$. Second, the block is not present in the buffer disk, i.e., $b \notin G$. Third, fetching the blocks and caching them into the buffer disk can improve energy efficiency, i.e., $Es(b) > 0$. Lastly, the block is residing in an active data disk, i.e., $disk(b) \in A$. Note that set A^+ (see Table 1) contains all the blocks that satisfy the above four criteria.

To maximize energy efficiency, we have to identify data-disk-resident blocks with the highest energy savings potential. This step is implemented by maintaining a set, G^+, of blocks with the highest energy saving in $A^+ \cup G$. Thus, blocks in $A^+ \cap G^+$ are the candidate blocks to be prefetched in the prefetching phase. A tie of energy savings between a buffer-disk-resident block and a data-disk-resident block can be broken in favor of the buffer-disk-resident block. If two data-disk-resident blocks have the same

Input: a request *r*, parallel disk system with *m* disks

1 if *block(r)* is present in the buffer disk {

2 if *disk(r)* is active and $T_{Disk}(r) \leq T_0(r)$, where $T_{Disk}(r)$ and $T_0(r)$ are response time of *r* when serviced by disk(r) and the buffer disk, respectively

3 The request *r* is serviced by *disk(r)*;

4 else the request *r* is serviced by the buffer disk;

}

5 else { /* *block(r)* is not present in the buffer disk */

/* Initiate the prefetching phase */

6 if *disk(r)* is in the standby state /* spin up disk(r) when it is standby */

7 spin up *disk(r)*;

8 Compute the energy savings of references in $A \subseteq R$, where *A* is a subset of the lookahead *R*, and $\forall$ *r'*$\in$*A: disk(r')* is active;

9 Update the energy savings of blocks in the buffer disk;

10 Fetch blocks in $A^+ \cap G^+$;

11 Evicting the blocks in $G - G^+$ with the lowest energy savings as necessary, where *G* is the set of blocks present in the buffer disk;

A^+ is the set of blocks, such that if block $b \in A^+$, then *b* is referenced by a request in the lookahead, *b* is not present in the buffer disk, *disk(b)* is active (i.e., *disk(b)* $\in$ *A*), and the energy saving $E_s(b)$ of *b* is larger than 0;

G^+ is the set of blocks with the highest energy saving in $A^+ \cup G$,

11.a such that $\sum_{r' \in G^+} \lambda(r') \cdot t(r') < B_0$ /* Bandwidth constraint must be satisfied */

11.b $\sum_{r' \in G^+} s(r') \leq C_0$ /* Capacity constraint must be satisfied */

/* The request r is then serviced */

12 if *block(r)* has not been prefetched

13 The request *r* is serviced by *disk(r)*;

14 else return *block(r)*; /* *block(r)* was recently retrieved; no extra I/O is necessary */

}

Figure 4.3: Algorithm PRE-BUD: the prefetching module

energy saving, the tie is broken in favor of the block accessed earlier by a request in the look-ahead.

In the case that the buffer disk is full, blocks in $G - G^+$ must be evicted from the buffer disk (see Step 11 in Figure 4.3). This is because $G - G^+$ contains the blocks with the lowest energy savings. We assign zero to the energy savings of buffer-disk-resident blocks that will not be accessed by any requests in the look-ahead. The buffer-disk-resident blocks without any contribution to energy conservation will be among the first to be evicted from the buffer disk, if a disk-resident block with high energy saving must be fetched when the buffer disk is full. Blocks that will not be accessed in the look-ahead are evicted in the least-recently-used order.

PRE-BUD can conserve more energy by the virtue of its on-demand manner, which defers prefetching decisions till the last possible moment when the above two criteria are satisfied. Deferring the prefetching phase is beneficial, because (1) this phase needs to spin up a corresponding disk if it is in the standby state, and (2) late prefetching leads to a larger look-ahead for better energy-aware prefetching decisions.

The prefetching module can be readily integrated with a disk scheduling mechanism, which is employed to independently optimize low-level disk access times in each individual disk. This integration is implemented by batching disk requests and offering each disk an opportunity to reschedule the requests to optimize low-level disk access performance.

4.2.2 Energy-Saving Calculation Module

We develop an energy-saving prediction model, based on which we implement the energy-saving calculation module invoked in Steps 8 and 9 in the prefetching module (see Figure 4.3). The prediction model along with the calculation module is indispensable for the prefetcher, because the energy savings of a block represents the importance and priority of placing the block in the buffer disk to reduce the energy

consumption of the disk system. The energy-saving calculation module can illustrate the amount of energy conserved by fetching a block from a data disk into a buffer disk. It also calculates the utility of caching a buffer-disk-resident block rather than evicting it from the buffer disk. Table 4.2 summarizes the notation for the description of the energy-saving calculation module.

To analyze circumstances under which prefetching blocks can yield energy savings, we focus on a single referenced block stored in a data disk. Let $R_j \subseteq R$ be a set of references accessing blocks in the jth data disk. Thus, R_j is a subset of the lookahead R and can be defined as

$$R_j = \{r \,|r \in R \wedge disk(r) = j\text{thdatadisk} \wedge \text{block}(r) = b_{k,j} \wedge b_{k,j} \notin G\}.$$

Given a set $R_{kj} \subseteq R$ of references accessing the kth block b_{kj} in the jth data disk, let us derive the energy saving $E_s(b_{kj})$ achieved by fetching b_{kj} from the data disk into the buffer disk. R_{kj} is comprised of all the requests referencing a common block bkj that is not present in the buffer disk; therefore, R_{kj} can be formally expressed as $R_{k,j} = \{r \,|r \in R \wedge block(r) = b_{k,j} \wedge b_{k,j} \notin G\}$. Intuitively, energy savings $E_s(b_{kj})$ can be computed by considering the energy consumption incurred by each disk request in R_{kj}.

Given a reference list R_j and a block b_{kj}, in what follows we identify four cases where a reference in R_j can contribute to positive energy savings by the virtue of prefetching block b_{kj}. First, we introduce two energy saving principles utilized by PRE-BUD.

Energy Saving Principle 1: To increase the length and number of idle periods larger than the disk break-even time T_{BE}, which is the minimum disk standby time required to offset the cost of entering the standby state. This principle can be realized by combining two adjacent idle periods to form a single idle period that is larger than T_{BE}. PRE-BUD fetches, in advance, a block accessed between two adjacent idle

Notation	Description
R_j	A set of references accessing blocks in the jth data disk
$R_{k,j} \subseteq R$	A set of references accessing the kth block $b_{k,j}$ in the jth data disk
$b_{k,j}$	The kth block in the jth data disk
T_{BE}	Break-even time. Minimum idle time required to compensate the cost of entering the standby state
T_{ij}	Active time period serving the ith request issued to the jth data disk
t_{ij}	Time spent serving the ith request issued to the jth data disk
α_{ij}	Time spent in the idle period prior to the ith request accessing a block in disk j
I_{ij}	An idle period prior to the ith request accessing a block in the jth data disk
n_j	The total number of requests (in the lookahead) issued to the jth disk
Φ_j	A set of disk access activities for references in R_j
$time(b_{kj})$	Active time period to serve a request accessing block b_{kj}
$block(T_{ij})$	A block accessed during the active period T_{ij}
T_D	Time to transition from active/idle to standby
T_U	Time to transition from standy to active mode
E_D	Energy overhead of transitioning from active/idle to standby
E_U	Energy overhead of transitioning from standby to active mode
P_A, P_I, P_S	Disk power in the active, idle, and standby mode

Table 4.2: Notation for the description of the energy-saving calculation module

periods, thereby possibly forming a larger inactivity time that allows the disk to enter the standby state to conserve energy.

Energy Saving Principle 2: To reduce the number of power-state transitions. The energy efficiency of a disk can be further improved by minimizing the energy cost of spinning up and down disks. Disk vendors can provide high quality disks with low spin-up/down energy over-heads; PRE-BUD aims to reduce the number of disk spin-up and spin-down while enlarging disk idle times. We implement this principle in PRE-BUD by combining two adjacent standby periods to eliminate unnecessary state transitions between the two standby periods.

Now we investigate cases which exploit the above energy saving principles to conserve energy in disks. Let $\Phi_j = \{I_{1j}, T_{1j}, I_{2j}, T_{2j}, \ldots I_{ij}, T_{ij}, \ldots I_{nj,j}, T_{nj,j}\}$ be a

set of disk accesses for references in R_j, where for an active period T_{ij}, t_{ij} is the time spent serving the ith request issued to data disk j; for idle period I_{ij}, α_{ij} is the time spent in the idle period prior to the ith request accessing a block in the jth data disk, and n_j is the total number of requests issued to the jth disk. We denote $block(T_{ij})$ as a block accessed during the active period T_{ij}.

The following three cases demonstrate scenarios that apply *energy saving principle 1* to generate longer idle periods (i.e., longer than T_{BE}) by prefetching $block(T_{ij})$ to combine the ith and $(i+1)th$ idle periods. Let us pay attention to the ith active period T_{ij} and the two periods I_{ij} and $I_{(i+1)j}$ (i.e., the ones adjacent to T_{ij}). Cases 1-3 share two common conditions - (1) both I_{ij} and $I_{(i+1)j}$ are larger than zero and (2) the summation of t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$ is larger than the break-even time T_{BE}.

Case 1: Both the ith and $(i+1)th$ idle periods are equal to or smaller than the break-even time T_{BE}. Thus, we have $0 < \alpha_{ij} \leq T_{BE}$, $0 < \alpha_{(i+1)j} \leq T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$.

Case 2: The ith idle period is equal to or smaller than the break-even time T_{BE}; the $(i+1)th$ idle period is larger than T_{BE}. Formally, we have $0 < \alpha_{ij} \leq T_{BE}$,$\alpha_{(i+1)j} > T_{BE}$,and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$.

Case 3: The ith idle period is larger than T_{BE}; the $(i+1)th$ idle period is equal to or smaller than T_{BE}. The conditions for case 3 can be expressed as: $\alpha_{ij} > T_{BE}$,$0 < \alpha_{(i+1)j} \leq T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$.

Now we calculate, in the above three cases, the energy savings produced by fetching $block(T_{ij})$ from the jth data disk to the buffer disk. The calculation makes use of the following definitions:

- Let P_A, P_I, and P_S represent the disk power consumption in the active, idle, and standby modes. Let T_D and T_U be times to transition to the standby and active mode; let E_D and E_U be energy overhead to transition to standby and active.

- E_{WOP} denotes energy consumption of the periods t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$ when PRE-BUD is not applied.

- In case of having $block(T_{ij})$ prefetched, E_{WPF} denotes energy consumption of the jth disk in the periods t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$.

- E_{BUD} represents energy consumption of the buffer disk accessing the prefetched $block(Tij)$.

- For $block_(b_{kj})$, active time spent serving a request accessing the block is denoted by $time(b_{k,j})$.

Energy savings, $E_S(block(T_{ij}))$, contributed by prefetching $block(T_{ij})$ can be written as:

$$E_S(\text{block}(T_{ij})) = E_{WOP} - (E_{WPF} + E_{BUD}) \tag{4.1}$$

Energy savings, $E_S(block(T_{ij})$, in case 1: For case 1, I_{ij} and $I_{(i+1)j}$ are equal to or smaller than T_{BE}. This condition implies that the jth disk is in the idle mode during I_{ij} and $I_{(i+1)j}$. Energy consumption experienced by the disk in active period T_{ij} is $P_A \cdot t_{ij}$. Hence, E_{WOP} in case 1 can be expressed as:

$$E_{WOP} = P_I \cdot (\alpha_{ij} + \alpha_{(i+1)j}) + P_A \cdot t_{ij} \tag{4.2}$$

When $block(T_{ij})$ is prefetched, a large (i.e., larger than T_{BE}) idle period can be formed by combining the periods T_{ij}, I_{ij}, and $I(i+1)j$. Therefore, E_{WPF} can be computed as the energy consumption of the jth disk in the standby mode during T_{ij}, I_{ij}, and $I_{(i+1)j}$. Taking into account energy overhead of power state transitions, we can calculate E_{WPF} using the equation below:

$$E_{WPF} = P_S \cdot (\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} - T_D - T_U) + E_D + E_U \tag{4.3}$$

We assume that the buffer disk and data disks are identical; therefore, energy consumption E_{BUD} of the buffer disk accessing the prefetched $block(T_{ij})$ is

$$E_{BUD} = P_A \cdot t_{ij} \tag{4.4}$$

$E_S(block(T_{ij}))$ in case 1 can be determined by substituting Eqs. (4.2)-(4.4) into Eq. (4.1). Hence, we have:

$$E_S(\text{block}(T_{ij})) = P_I \cdot (\alpha_{ij} + \alpha_{(i+1)j}) - P_S \cdot (\alpha_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) - E_D - E_U \tag{4.5}$$

Energy saving $E_S(block(T_{ij}))$ in case 2: The jth disk in this case is transitioned into standby during $I_{(i+1)j}$, since $I_{(i+1)j}$ is larger than T_{BE}. The energy consumption of the disk in $I_{(i+1)j}$ is expressed as (see the third term on the right hand side of Eq. 4.6 below). Thus, the energy consumption E_{WOP} of the disk in T_{ij}, Iij, and $I_{(i+1)j}$ is:

$$E_{WOP} = P_I \cdot \alpha_{ij} + P_A \cdot t_{ij} + (P_S \cdot (\alpha_{(i+1)j} - T_D - T_U) + E_D + E_U) \tag{4.6}$$

We derive $E_S(block(T_{ij}))$ in case 2 by substituting Eqs. (4.6), (4.3), and (4.4) for for E_{WOP}, E_{WPF}, and E_{BUD}. Thus, we have

$$E_S(\text{block}(T_{ij})) = P_I \cdot \alpha_{ij} - P_S \cdot (\alpha_{ij} + t_{ij}) \tag{4.7}$$

Energy savings, $E_S(block(T_{ij}))$, in case 3: The Energy saving $E_S(block(T_{ij}))$ in this case is very similar to that in case 2 except that the jth disk is transitioned into standby during I_{ij} rather than $I_{(i+1)j}$. Consequently, the energy saving $E_S(block(T_{ij}))$ in case 3 can be written as:

$$E_S(\text{block}(T_{ij})) = P_I \cdot \alpha_{(i+1)j} - P_S \cdot (\alpha_{(i+1)j} + T_{ij}) \tag{4.8}$$

Case 4: The case described here shows a scenario that applies energy saving principle 2 to reduce power-state transitions by prefetching $block(T_{ij})$ to combine two adjacent standby periods I_{ij} and $I_{(i+1)j}$.

Energy saving $E_S(block(T_{ij}))$ in case 4: In this case, both α_{ij} and $\alpha_{(i+1)j}$ are larger than T_{BE}, meaning that the jth disk can be standby in these two time intervals to conserve energy. Formally, we have $\alpha_{ij} > T_{BE}$,$\alpha_{(i+1)j} > T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$. Thus, energy consumption E_{WOP} of the jth disk without a buffer disk is:

$$\begin{aligned} E_{WOP} = P_A \cdot t_{ij} + (P_S \cdot (\alpha_{ij} - T_D - T_U) + E_D + E_U) \\ + \Big(P_S \cdot (\alpha_{(i+1)j} - T_D - T_U) + E_D + E_U\Big) \end{aligned} \tag{4.9}$$

where the second and third term on the right hand side of Eq. (4.9) are the energy consumed by the disk in standby periods I_{ij} and $I_{(i+1)j}$, respectively. With a buffer disk in place, the energy consumption E_{WPF} and E_{BUD} in this case are the same as in case 1 (see Eqs. 4.3 and 4.4). Therefore, the energy savings, $ES(block(T_{ij}))$, in this case is derived from E_{WOP} (see Eq. 4.9), E_{WPF}, and E_{BUD} as:

$$E_S(\text{block}(T_{ij})) = E_D + E_U - P_S \cdot (T_D + T_U + t_{ij}) \tag{4.10}$$

Case 5 below summarizes scenarios where prefetching a block may have negative impacts on the energy efficiency.

Case 5: If the summation of t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$ is smaller than or equal to T_{BE}, i.e., $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} \leq T_{BE}$, then prefetching block bkj causes an negative impact on energy conservation.

Energy savings, $E_S(block(T_{ij}))$ in case 5: Since $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} \leq T_{BE}$, the disk j stays in the idle mode during periods T_{ij}, I_{ij}, and $I_{(i+1)j}$. If the block b_{kj} is prefetched to the buffer disk, the energy consumption E_{WPF} of disk j in the three

periods is:

$$E_{WPF} = P_I \cdot (\alpha_{ij} + t_{ij} + \alpha_{(i+1)j}) \tag{4.11}$$

The values of E_{WOP} and E_{BUD} are the same as those of case 1 (see Eq. 4.2). Applying E_{WOP}, $EWPF$ and $EBUD$ to Eq. (1), we estimate the negative energy-saving impact $E_S(block(T_{ij}))$ as:

$$E_S(\text{block}(T_{ij})) = -P_I \cdot t_{ij} \tag{4.12}$$

In light of the above four cases, the set Φ_{kj} of disk activities for references accessing block b_{kj} in disk j can be partitioned into the following four disjoint subsets,

$$\Phi_{k,j} = \Phi_{k,j,1} \cup \Phi_{k,j,2} \cup \Phi_{k,j,3} \cup \Phi_{k,j,4} \cup \Phi_{k,j,5} \tag{4.13}$$

where $\Phi_{k,j,1}$, $\Phi_{k,j,2}$, $\Phi_{k,j,3}$, $\Phi_{k,j,4}$ and $\Phi_{k,j,5}$ contain active time periods that respectively satisfy the conditions of the four energy-saving cases. The four subsets can be defined as:

$$\begin{aligned} \Phi_{k,j,1} &= T_{ij} | \text{block}(T_{ij}) = b_{k,j} \wedge 0 < \alpha_{ij} \leq T_{BE} \wedge 0 < \alpha_{(i+1)j} \\ &\leq T_{BE} \wedge \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE} \end{aligned}$$

for case 1;

$$\begin{aligned} \Phi_{k,j,2} &= T_{ij} | \text{block}(T_{ij}) = b_{k,j} \wedge 0 < \alpha_{ij} \leq T_{BE} \wedge \alpha_{(i+1)j} \\ &> T_{BE} \wedge \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE} \end{aligned}$$

for case 2;

```
Input: block b_{k,j}, disk j, a set Φ_j of disk access activities; Output: E_S(b_{k,j})
1  Initialize E_S(b_{k,j}) to 0;
2    for (i = 1 to n_j) {
3       if ( α_ij + t_ij + α_(i+1)j > T_BE ) {  /* Cases 1-4 */
4          if ( 0 < α_ij ≤ T_BE ) {
5             if ( 0 < α_(i+1)j ≤ T_BE )  /* Case 1, see Eq. (4.5) */
6                E_S(b_ij) = E_S(b_ij) + P_I · (α_ij + α_(i+1)j) − P_S · (α_ij + t_ij + α_(i+1) − T_U − T_D) − E_D − E_U ;
7             else E_S(b_ij) = E_S(b_ij) + P_I · α_ij − P_S · (α_ij + t_ij) ;/* Case 2, see Eq. (4.7) */
8          }
9          else {
10            if  ( 0 < α_(i+1)j ≤ T_BE ) /* Case 3, see Eq. (4.8) */
11               E_S(b_ij) = E_S(b_ij) + P_I · α_(i+1)j − P_S · (α_(i+1)j + T_ij) ;
12            else E_S(b_ij) = E_S(b_ij) + E_D + E_U − P_S · (T_D + T_U + t_ij). /* Case 4, see Eq. (4.10) */
13         }
14      } /* end Cases 1-4 */
15      else E_S(b_ij) = E_S(b_ij) − P_I · t_ij ; /* Negative energy saving. Case 5, see Eq. (4.12) */
16   } /* end for */
17   return E_S(b_ij) − P_A · time(b_{k,j})
```

Figure 4.4: Algorithm PRE-BUD: the energy-saving calculation module

$$\begin{aligned}\Phi_{k,j,3} &= T_{ij}|\text{block}(T_{ij}) = b_{k,j} \wedge \alpha_{ij} > T_{BE} \wedge 0 < \alpha_{(i+1)j} \\ &\leq T_{BE} \wedge \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}\end{aligned}$$

for case 3;

$$\begin{aligned}\Phi_{k,j,4} &= T_{ij}|\text{block}(T_{ij}) = b_{k,j} \wedge \alpha_{ij} > T_{BE} \wedge \alpha_{(i+1)j} \\ &> T_{BE} \wedge \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}\end{aligned}$$

for case 4; and $\Phi_{k,j,5} = \left\{T_{ij}|\text{block}(T_{ij}) = b_{k,j} \wedge \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} \leq T_{BE}\right\}$ for case 5.

Now we are positioned to show the derivation of energy savings, $E_S(b_{kj})$, yielded by fetching block b_{kj} from disk j to the buffer disk. Thus, $E_S(b_{kj})$ can be derived

from Eqs. (4.5), (4.7), (4.8), (4.10), (4.11) and Eq. (4.12), where the last item on the right hand side is the energy overhead of fetching b_{kj} from disk j to the buffer disk.

$$
\begin{aligned}
E_S(b_{k,j}) &= \sum_{T_{ij} \in \Phi_{k,j}} (E_S(\text{block}(T_{ij}))) \\
&= \sum_{T_{ij} \in \Phi_{k,j,1}} \Big(P_I \cdot (\alpha_{ij} + \alpha_{(i+1)j}) - P_S \cdot (\alpha_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) - E_D - E_U \Big) \\
&+ \sum_{T_{ij} \in \Phi_{k,j,2}} (P_I \cdot \alpha_{ij} - P_S \cdot (\alpha_{ij} + t_{ij})) + \sum_{T_{ij} \in \Phi_{k,j,3}} \Big(P_I \cdot \alpha_{(i+1)j} - P_S \cdot (\alpha_{(i+1)j} + t_{ij}) \Big) \\
&+ \sum_{T_{ij} \in \Phi_{k,j,4}} (E_D + E_U - P_S \cdot (T_D + T_U + t_{ij}).) - P_I \cdot \sum_{T_{ij} \in \Phi_{k,j,5}} t_{ij} - P_A \cdot \text{time}(b_{k,j})
\end{aligned}
\tag{4.14}
$$

Given the kth block bkj residing in disk j, the algorithm used to compute the energy savings of prefetching block b_{kj} from the data disk to the buffer disk is described in Figure 4.4. All of the energy saving cases are handled explicitly from Steps 3 through 14; whereas Step 15 addresses the issue of negative energy savings. The time complexity of the energy-saving calculation module is low, because the time complexity of this routine for each block is $O(nj)$, where nj is the number of requests in the look-ahead corresponding to the jth disk. After the block b_{kj} is fetched to the buffer disk, $= \{I_{1j}, T_{1j}, I_{2j}, T_{2j}, ...I_{ij}, Tij, ...I_{nj,j}T_{nj,j}\}$ the set j of disk access activities for references in R_j must be updated by deleting any $T_{ij} \in \Phi_j$ accessing b_{kj}, i.e., $block(T_{ij}) = b_{kj}$.

4.3 Analysis of PRE-BUD

In this section, we analyze the energy efficiency and performance of PRE-BUD. We start the analysis by showing the energy consumption of a full-power baseline system without turning any disks into the standby state. Next, we analyze the energy consumption of a parallel disk system with the dynamic power management (DPM) technique. Last, our analysis will be focused on the energy consumption and response time of parallel disk systems with PRE-BUD.

4.3.1 A Full-Power Baseline System

In this section we describe an energy consumption model, which is built to quantitatively calculate energy consumption of a modeled parallel disk systems. We model the power of a parallel disk system with m disks as a vector $P = (P_1, P_2, ..., P_m)$. The power P_i of the ith disk is represented by three parameters, i.e., $P_i = (P_{A,i}, P_{I,i}, P_{S,i})$, where $P_{A,i}$, $P_{I,i}$, and $P_{S,i}$ are the power of the ith disk when it is in the active, idle, and standby state, respectively. Let $e_{j,i}$ be the energy consumption to serve the jth request served by the ith disk. We denote the energy consumption rate of the disk when it is active by $P_{A,i}$ and the energy consumption $e_{j,i}$ can be written as

$$e_{j,i} = x_{j,i} \cdot P_{A,i} \cdot t_{j,i} = x_{j,i} \cdot P_{A,i} \cdot \left(t_{SK,j,i} + t_{RT,j,i} + \frac{s_j}{B_i} \right) \tag{4.15}$$

where $t_{j,i}$ is the service time of request j on disk i. $t_{j,i}$ is the summation of $t_{SK,j,i}$, $t_{RT,j,i}$, and s_j/B_i, which are the seek time and rotational latency of the request, and the data transfer time depending on the data size s_j and the transfer rate B_i of the disk. Element $x_{j,i}$ is "1" if request j is responded by the ith disk and is "0", otherwise. Since each request can be served by only one disk, we have $\sum_{i=1}^{m} x_{j,i} = 1$.

Given a reference string R, we can compute the energy E_A consumed by serving all requests as

$$\begin{aligned} E_A(P,R) &= \sum_{i=1}^{m} \sum_{j=1}^{n} e_{j,i} = \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{j,i} \cdot P_{A,i} \cdot t_{j,i}) \\ &= \sum_{i=1}^{\mathrm{m}} \sum_{j=1}^{n} \left(x_{j,i} \cdot P_{A,i} \cdot \left(t_{SK,j,i} + t_{RT,j,i} + \tfrac{s_j}{B_i} \right) \right) \end{aligned} \tag{4.16}$$

We define f_j as the completion time of request r_i. in the reference string. Then, we obtain the analytical formula for the energy consumed when disks are idle:

$$E_I(P,R) = \sum_{i=1}^{m} (P_{I,i} \cdot T_{I,i}) \tag{4.17}$$

where $T_{I,i}$ is the time interval when the ith disk is idle. $T_{I,i}$ can be derived from the total disk I/O processing time and completion time of the last request served by the disk. Thus, we have

$$T_{I,i} = \max_{j=1}^{n} (x_{j,i} \cdot f_j) - \sum_{j=1}^{n} \left(x_{j,i} \cdot \left(t_{SK,j,i} + t_{RT,j,i} + \frac{s_j}{B_i} \right) \right) \tag{4.18}$$

where the first term on the right-hand side of Eq. (4.18) is the summation of I/O processing times and disk idle times, and the second term is the total I/O time. The total energy consumption E_{NEC} of a parallel disk system without placing any disk into the standby state is derived from Eqs. (4.16) and (4.17) as

$$\begin{aligned} E_{NEC}(P,R) &= E_A(P,R) + E_I(P,R) \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} e_{j,i} + \sum_{i=1}^{m} (P_{I,i} \cdot T_{I,i}) \end{aligned} \tag{4.19}$$

4.3.2 Dynamic Power Management

Energy in disks systems can be efficiently reduced by employing the dynamic power management (DPM) strategy, which places disks into standby when they are idle. To analyze the energy efficiency of PRE-BUD, it is important and intriguing to model energy consumption in a DPM-based parallel disk system. If there is an idle time of the ith disk that is larger than the break-even time $T_{BE,i}$, then energy conservation can be achieved by putting the disk into the standby state. Otherwise, the energy penalty to transition between the high-power and low-power state is unable to be offset by the energy conserved. Let $P_{TR,i}$ be the power of state transitions in the ith disk. Let $P_{AS,i}$ and $P_{SA,i}$ denote additional power introduced by transitions from active to standby, and vice versa. $P_{TR,i}$ can be derived from $P_{AS,i}$ and $P_{SA,i}$ as

$$P_{TR,i} = P_{AS,i} + P_{SA,i} = \frac{T_{AS,i} \cdot P_{AS,i} + T_{SA,i} \cdot P_{SA,i}}{T_{AS,i} + T_{SA,i}} \tag{4.20}$$

where the numerator is the energy consumption caused by a pair of transitions and the denominator is the transition time. In light of Eq. (4.20), one can calculate the break-even time $T_{BE,i}$ as

$$T_{BE,i} = \begin{cases} (T_{AS,i} + T_{SA,i}) \cdot \left(1 + \frac{P_{TR,i} - P_{A,i}}{P_{A,i} - P_{S,i}}\right) & \text{if} P_{TR,i} > P_{A,i} \\ T_{AS,i} + T_{SA,i} \textbf{ otherwise} \end{cases} \tag{4.21}$$

In what follows, we make use of $T_{BE,i}$ to quantify the energy consumption of a parallel disk system when the DPM technique is employed. Suppose the number of idle time intervals in a disk i is N_i; a sequence of idle periods in the disk can be expressed as $(t_{I,i,1}, t_{I,i,2}, ..., t_{I,i,Ni})$, where $t_{I,i,k}$ represents the length of the kth idle period in the sequence. Let $\widehat{E}_I(P,R)$ be the energy consumed when disks are idle. The expression of $\widehat{E}_I(P,R)$ is given as

$$\begin{aligned} \hat{E}_I(P,R) &= \sum_{i=1}^{m} \left(P_{I,i} \cdot \hat{T}_{I,i}\right) \\ &= \sum_{i=1}^{m} \left(P_{I,i} \cdot \sum_{k=1}^{N_i} (y_{k,i} \cdot t_{I,i,k})\right) \end{aligned} \tag{4.22}$$

where $\hat{T}_{I,i}$ is the summation of small idle time intervals that are unable to compensate the cost of transitioning to the standby state. $\hat{T}_{I,i}$ can be derived from a step function $y_{k,i}$, where $y_{k,i}$ is "1" if the idle interval is smaller than or equal to the break-even time. Otherwise, $y_{k,i}$ is "0". Using the step function $y_{k,i}$, we can express $\hat{T}_{I,i}$ in Eq. (4.22) as $\hat{T}_{I,i} = \sum_{k=1}^{N_i} (y_{k,i} \cdot t_{I,i,k})$.

The energy consumption of the parallel disk system when the disks are in the standby state can be expressed as

$$\begin{aligned} E_S(P,R) &= \sum_{i=1}^{m} (P_{S,i} \cdot T_{S,i}) \\ &= \sum_{i=1}^{m} \left(P_{S,i} \cdot \sum_{k=1}^{N_i} (\bar{y}_{k,i} \cdot (t_{I,i,k} - T_{BE,i}))\right) \end{aligned} \tag{4.23}$$

where $T_{S,i}$ is the time period when disk i is in the standby state. Similar as $\hat{T}_{I,i}$, $T_{S,i}$ is derived from a step function $\bar{y}_{k,i}$, where $\bar{y}_{k,i}$ is "1" if the idle interval is larger than $T_{BE,i}$, and is "0", otherwise. With the step function $\bar{y}_{k,i}$, we can model $T_{S,i}$ in Eq. (4.23) as $T_{S,i} = \sum_{k=1}^{N_i} (\bar{y}_{k,i} \cdot (t_{I,i,k} - T_{BE,i}))$.

Similarly, below we obtain the formula for the energy consumption of disk power-state transitions

$$E_{TR}(P, R) = \sum_{i=1}^{m} (P_{TR,i} \cdot T_{TR,i}) \tag{4.24}$$

where $P_{TR,i}$ is determined by Eq. (4.20). $T_{TR,i}$ is the time interval when disk i is transitioning from one power state into another. $T_{TR,i}$ can be derived from $T_{BE,i}$. Hence, we obtain $T_{TR,i} = \sum_{k=1}^{N_i} (\bar{y}_{k,i} \cdot T_{BE,i})$.

The energy consumption E_{DPM} of the parallel disk system with the DPM technique is the summation of the energy incurred by the disks when they are in the active, idle, standby, and transition states. Thus, E_{DPM} can be derived from Eqs. (4.16), (4.22), (4.23), and (4.24) as

$$E_{DPM}(P, R) = E_A(P, R) + \hat{E}_I(P, R) + E_S(P, R) + E_{TR}(P, R) \tag{4.25}$$

4.3.3 Derivation of Energy Efficiency for PRE-BUD

Now we analyze the energy efficiency of the PRE-BUD strategy. We only analyze the energy consumption of a parallel I/O system with PRE-BUD where an extra disk is added to the system as a buffer disk. PRE-BUD using an existing disk can be modeled similarly and can be derived from the models presented in this section.

First of all, we analyze the energy overhead E_{PF} introduced by prefetching the popular data blocks from data disks to the buffer disk. Let $D = (D_1, D_2, ..., D_q)$ be a set of data blocks retrieved by reference string R. We make use of a predicate $\alpha_{j,i,k}$, which asserts that request r_i is accessing data block k on disk i, to partition the reference string in a way that requests accessing the same kth block on disk i can

be grouped into the one set $R_{k,i}$. Thus, we have

$$R_{k,i} = \{\, r_j \in R \mid x_{j,i} = 1 \wedge \alpha_{j,i,k} = TRUE \,\} \tag{4.26}$$

The sizes of all the requests in $R_{k,i}$ are identical. For simplicity, we denote the size of requests in $R_{k,i}$ as $s_{k,i}$. The following property must be satisfied:

$$\forall 1 \leq j \leq n,\ 1 \leq k \leq q,\ r_j \in R_{k,i} :\ s_j = s_{k,i} \tag{4.27}$$

In most cases, it is impossible for a buffer disk to cache all the popular data sets. Therefore, we introduce the following step function to distinguish data blocks prefetched from data disks to the buffer disk.

$$z_{k,i} = \begin{cases} 1 \textbf{ if block } k \textbf{ on disk } i \textbf{ is prefetched,} \\ 0 \textbf{ otherwise}. \end{cases} \tag{4.28}$$

Energy consumption E_{PF} caused by prefetching contains two components: energy consumption $E_{R,PF}$ of reading frequently accessed data blocks from the data disks and energy consumption $E_{W,PF}$ of placing the data blocks to the buffer disk. Thus, E_{PF} is quantified below

$$\begin{aligned} & E_{PF}(P,D) = E_{R,PF}(P,D) + E_{W,PF}(P,D) \\ & = \sum_{i=1}^{m} \sum_{k=1}^{q} \left(z_{k,i} \cdot P_{A,i} \cdot \left(t_{SK,k,i} + t_{RT,k,i} + \tfrac{s_{k,i}}{B_{R,i}} \right) \right) \\ & + \sum_{i=1}^{m} \sum_{k=1}^{q} \left(z_{k,i} \cdot P_{A,0} \cdot \left(t_{SK,k,0} + t_{RT,k,0} + \tfrac{s_{k,i}}{B_{W,0}} \right) \right) \end{aligned} \tag{4.29}$$

where $P_{A,0}$ is the power of the buffer disk in the active state, $B_{R,i}$ is the read transfer rate of data disk i, and $B_{W,0}$ is the write transfer rate of the buffer disk. Next, let us derive expressions to calculate the energy consumption E_0 in the buffer disk. E_0 is the summation of active, idle, and sleep state energy consumption totals of the buffer

disk, and power state transition overheads. Thus,

$$E_0 = E_{A,0} + E_{I,0} + E_{S,0} + E_{TR,0} \tag{4.30}$$

where $E_{A,0}$, $E_{I,0}$, and $E_{S,0}$ are the active, idle, and sleep state energy consumption totals of the buffer disk. $E_{TR,0}$ is the energy overhead for power state transitions. In what follows, we direct our attention to the analytical formulas of $E_{A,0}$, $E_{I,0}$, and $E_{S,0}$. Given a set D of accessed data blocks, we model energy $E_{A,0}$ of the buffer disk when it is active as

$$\begin{aligned} E_{A,0}(D) &= \sum_{i=1}^{m}\sum_{k=1}^{q}(z_{k,i}\cdot P_{A,0}\cdot T_{A,0}) \\ &= \sum_{i=1}^{m}\sum_{k=1}^{q}\left(z_{k,i}\cdot P_{A,0}\cdot \sum_{r_j\in R_{k,i}}\left(t_{SK,j,0}+t_{RT,j,0}+\frac{s_{k,i}}{B_{R,0}}\right)\right) \end{aligned} \tag{4.31}$$

where $T_{A,0}$ is the time period when the buffer disk is in the active state. $T_{A,0}$ is the accumulated service times of requests processed by the buffer disk.

Let $IS = (t_{I,1}, t_{I,2}, ..., t_{I,N0})$ be a sequence of idle periods in the buffer disk. Eq. (4.32) quantifies energy consumption $E_{I,0}$ of the buffer disk when it is sitting idle.

$$E_{I,0}(IS) = P_{I,0}\cdot \hat{T}_{I,0} = P_{I,0}\cdot \sum_{t_{I,k}\in IS}(y_{k,0}\cdot t_{I,k}) \tag{4.32}$$

where $\hat{T}_{I,0}$ is the summation of small idle time intervals that are unable to compensate the cost of transitioning to the sleep state. $y_{k,i}$ is a step function used in Eq. (4.22).

Energy consumption $E_{S,0}$ in Eq. (4.30) is expressed as

$$\begin{aligned} E_{S,0}(IS) &= P_{S,0}\cdot T_{S,0} \\ &= P_{S,0}\cdot \sum_{t_{I,k}\in IS}(\bar{y}_{k,0}\cdot(t_{I,k}-T_{BE,0})) \end{aligned} \tag{4.33}$$

where $T_{S,0}$ is the total time when the buffer disk is in the sleep mode. $T_{S,0}$ is derived from the break-even time given by Eq. (4.21) and the step function is used in Eq.

(4.23). The energy overhead $E_{TR,0}$ for power state transitions is expressed as follows

$$\begin{aligned} E_{TR,0}(IS) &= P_{TR,0} \cdot T_{TR,0} \\ &= P_{TR,0} \cdot \sum_{t_{I,k} \in IS} (\bar{y}_{k,0} \cdot T_{BE,i}) \end{aligned} \tag{4.34}$$

Energy consumption E_D of the data disks with the dynamic power management technique can be determined by applying Eq. (4.25).

Now we are in a position to obtain the energy consumption total of the parallel I/O system, $E_{PRE-BUD}$, with an extra buffer disk from Eqs. (4.25), (4.29), and (4.30). Thus,

$$E_{PRE-BUD} = E_{PF}(P, D) + E_0(D, IS) + E_D(P, R) \tag{4.35}$$

4.3.4 Derivation of Response Time for PRE-BUD

Now we are in a position to derive the response time approximation of the PRE-BUD architecture. By definition, the response time of a disk request is the interval between its arrival time and finish time. The response time can be calculated as a sum of a disk request's wait time and I/O service time. Let $D_0 = \{d_1, \cdots, d_k, \cdots, d_{l0}\}$ be a set of data blocks prefetched to a buffer disk. Throughout this section, the subscript 0 is used to represent the buffer disk. Let λ_k and t_k represent the access rate and I/O service time of the kth data block in D_0. Let ρ_0 and Λ_0 be the utilization and aggregate utilization of the buffer disk. Thus, we have

$$\rho_0 = \sum_{d_k \in D_0} (\lambda_k \cdot t_k) \textbf{ and } \Lambda_0 = \sum_{d_k \in D_0} \lambda_k \tag{4.36}$$

The mean service time $\bar{S}_0$ and mean-square service time $\bar{S}_0^2$ of disk accesses to the buffer disk are given as

$$\bar{S}_0 = \sum_{d_0 \in D_0} \left(\frac{\lambda_k}{\Lambda_0} \cdot t_k \right) = \frac{1}{\Lambda_0} \cdot \sum_{d_0 \in D_0} (\lambda_k \cdot t_k) \tag{4.37}$$

$$\bar{S}_0^2 = \sum_{d_0 \in D_0} \left(\frac{\lambda_k}{\Lambda_0} \cdot t_k^2 \right) = \frac{1}{\Lambda_0} \cdot \sum_{d_0 \in D_0} \left(\lambda_k \cdot t_k^2 \right) \tag{4.38}$$

where λ_k/Λ_0 is the probability of access to data block d_k in the buffer disk.

We model each disk in a parallel disk system as a single M/G/1 queue, which has exponentially distributed inter-arrival times and an arbitrary distribution for service times of disk requests. Consequently, we can obtain the mean response time $\bar{T}_0$ of accesses to the buffer disk from Eqs. (4.36), (4.37) and (4.38) as

$$\bar{T}_0 = \bar{S}_0 + \frac{\Lambda_0 \cdot \bar{S}_0^2}{2 \cdot (1 - \rho_0)} \tag{4.39}$$

In what follows, let us derive mean response time $\bar{T}_j$ of accesses to disk j. We denote $D_j (1 \leq j \leq m)$ as a set of data blocks stored in the jth disk. Let $D_j^{PF} \subseteq D_j (1 \leq j \leq m)$ be a set of data blocks in D_j prefetched to a buffer disk. Similarly, let $D'_j \subseteq D_j (1 \leq j \leq m)$ be a set of data blocks that has not been prefetched. For the jth disk, we have $D_j = D_j^{PF} \cup D'_j$. Let ρ_j and Λ_j represent the utilization and aggregate utilization of the buffer disk. ρ_j and Λ_j can be expressed as:

$$\rho_j = \sum_{d_k \in D'_j} (\lambda_k \cdot t_k) \text{and} \Lambda_j = \sum_{d_k \in D'_j} \lambda_k \tag{4.40}$$

The mean and mean-square service times (i.e., $\bar{S}_j$ and $\bar{S}_j^2$) of disk accesses to disk j are given as

$$\bar{S}_j = \sum_{d_0 \in D'_j} \left(\frac{\lambda_k}{\Lambda_j} \cdot t_k \right) = \frac{1}{\Lambda_j} \cdot \sum_{d_0 \in D'_j} (\lambda_k \cdot t_k) \tag{4.41}$$

$$\bar{S}_j^2 = \sum_{d_0 \in D'_j} \left(\frac{\lambda_k}{\Lambda_j} \cdot t_k^2 \right) = \frac{1}{\Lambda_j} \cdot \sum_{d_0 \in D'_j} \left(\lambda_k \cdot t_k^2 \right) \tag{4.42}$$

We can derive the mean response time $\bar{T}_j$ of accesses to data disk j from the above equations as

$$\bar{T}_j = \bar{S}_j + \frac{\Lambda_j \cdot \bar{S}_j^2}{2 \cdot (1 - \rho_j)} \tag{4.43}$$

Therefore, the overall mean response time of a parallel disk system with a buffer disk is written as below, where $\Lambda = \sum_{j=0}^{m} \Lambda_j$ is the aggregate access rate of the parallel disk system.

$$\bar{T} = \frac{1}{\Lambda} \cdot \sum_{j=0}^{m} \left(\Lambda_j \cdot \bar{T}_j \right) \tag{4.44}$$

4.4 Experimental Results

In this section we present our experimental results for the proposed PRE-BUD energy efficient prefetching approach for parallel disk systems. First we provide information about our simulation environment and parameters that were varied for our experiments. Next, we compare PRE-BUD with PDC and DPM - two well known energy conservation techniques for parallel disks [21]. Then, we study the impacts of various system parameters on energy efficiency and the performance of parallel disks.

4.4.1 Experiment Setup

Extensive experiments were conducted with a disk simulator based on the mathematical models presented in Sections 4.2 and 4.3. Our disk model (see Table 4.3) is based on the IBM Ultrastar 36Z15, which has been widely used in data-intensive environments [18]. Our simulator was implemented in JAVA, allowing us to quickly and easily change various system parameters. Both synthetic and real-world traces are used to evaluate PRE-BUD.

For comparison purposes, we consider a parallel I/O system (referred to as Non-Energy Aware) where disks are operating in a standard mode without employing any energy-saving techniques. In other words, disks are in the busy state while serving requests, and are in the idle state when not serving a request. Two PRE-BUD configurations are evaluated; the first configuration PRE-BUD1 adds an extra disk to be used as the buffer disk and the second configuration called PRE-BUD2 designates an existing disk as the buffer disk. Note that the term "hit rate" used throughout this

section is defined as the percentage of requests that can be served by the buffer disk. One of the goals of our experiments is to identify the parameters that are crucial to energy efficient disk storage systems.

Parameter	Value	Parameter	Value
Transfer Rate	55 MB/S	Spin Down Time:T_D	1.5 s
Active Power:P_A	13.5 W	Spin Up Time:T_U	10.9 s
Idle Power:P_I	10.2 W	Spin Down Energy:E_D	13.0 J
Standby Power:P_S	2.5 W	Spin Up Energy:E_U	135 J

Table 4.3: Disk Parameters (IBM Ultrastar 36Z15)

4.4.2 Comparison of PRE-BUD and PDC

Figure 4.5 shows the energy efficiency comparison results of our PRE-BUD strategy and the PDC [26] energy saving technique. PDC attempts to move popular data across the disks, such that the first disk has the most popular data, while the second disk has the second most popular set of data and so forth. We fixed the data size to be 275 MB and the hit rate is 95% for PRE-BUD. Since the data could potentially be anywhere in the disk system, the PDC strategy causes data to be moved within the disk system.

Figure 4.5 shows that PRE-BUD is more energy efficient than PDC if PDC has to move a large amount of data within the storage system. PDC may have a much higher initial energy penalty when a large amount of data must be moved within the storage system. PRE-BUD has a fixed amount of buffer disk space; for this example, it is fixed at 10% of the total data in the storage system. PRE-BUD can be adaptively tuned to find the particular amount of buffer disk capacity that will yield the largest amount of savings. PRE-BUD only needs to move blocks that can provide energy savings. In contrast, PDC makes no guarantees about the energy impact of moving data within the storage system. PDC does not adapt as quickly as our PRE-BUD strategy to changing workload conditions. The look-ahead window we

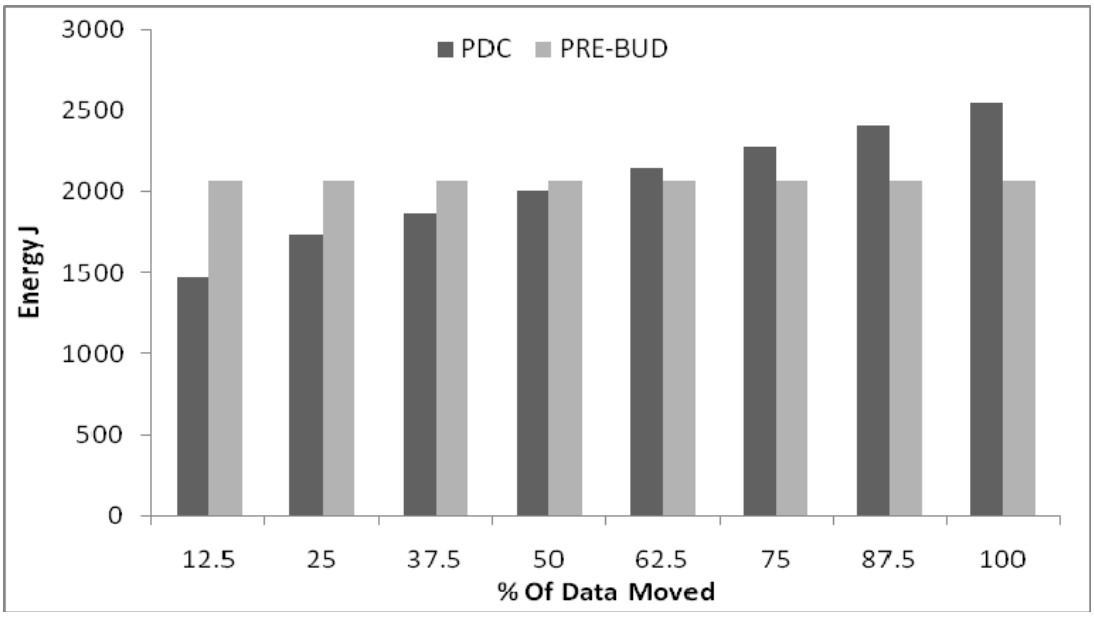


Figure 4.5: PDC and PRE-BUD Comparison

employ can amortize the expense of moving frequently accessed data into the buffer disk. PDC attempts to move frequently accessed data at one time, which can cause large over-heads when the workload of the parallel disk system changes frequently.

4.4.3 Impact of Data Size

The second set of experiments focused on evaluating the impact that the data size of the requests has on the energy savings of DPM and PRE-BUD. For these set of experiments we fixed the number of disks at 12. The hit rate of the buffer disk is varied from 85% to 100%. Figure 4.6 reveals that the data size has a huge impact on the energy efficiency of DPM and our PRE-BUD strategy when the hit rate is lower than 100%. If the hit rate is 100% for the buffer disk, data disks can sleep for a long period of time regardless of the data size.

The results depicted in Figure 4.6 indicate that our PRE-BUD strategy performs best with data-intensive applications that request large files. Thus, multimedia storage systems would be a perfect candidate for the PRE-BUD energy saving strategy. The data size has such a large impact on energy savings because of the break even time, TBE, which is 14.5 seconds for the chosen disk model. Large data sizes take a longer time to serve; consecutive buffer hits for a large data size meet the break even time. Conversely, small data sizes produce little or no energy efficiency gains. These experimental results confirm that the data size together with the hit rate combine to produce a probability of meeting TBE, which is the break-even time, with higher hit rates and large data sizes being the ideal combination for energy savings. PRE-BUD1 consumes more energy than DPM when the data size is 1MB or smaller. This is because PRE-BUD1 adds an extra disk to the disk system, and with a small data size energy efficient opportunities to put a disk to sleep are rare. This set of experiments leads us to the conclusion that large data sizes are conducive to energy efficiency in PRE-BUD.

4.4.4 Impact of Number of Data Disks

Now we evaluate the impact of varying the ratio of data disks to buffer disks. The number of buffer disks is fixed at 1; the number of data disks is set to 4, 8, and 12. The hit rate is fixed at 95% and the data size is varied from 1MB to 25MB. Not surprisingly, we discover from Figure 4.7 that as we increase the number of data disks per buffer disk, the energy savings becomes more pronounced for PRE-BUD. This energy efficiency trend is expected because increasing the number of disks makes each individual disk less heavily loaded.

The buffer disk simply prefetches blocks that can produce energy savings; lightly loaded disks are more likely to be switched into the standby mode to conserve energy. PRE-BUD, of course, has to prefetch a smaller amount of data from each disk to

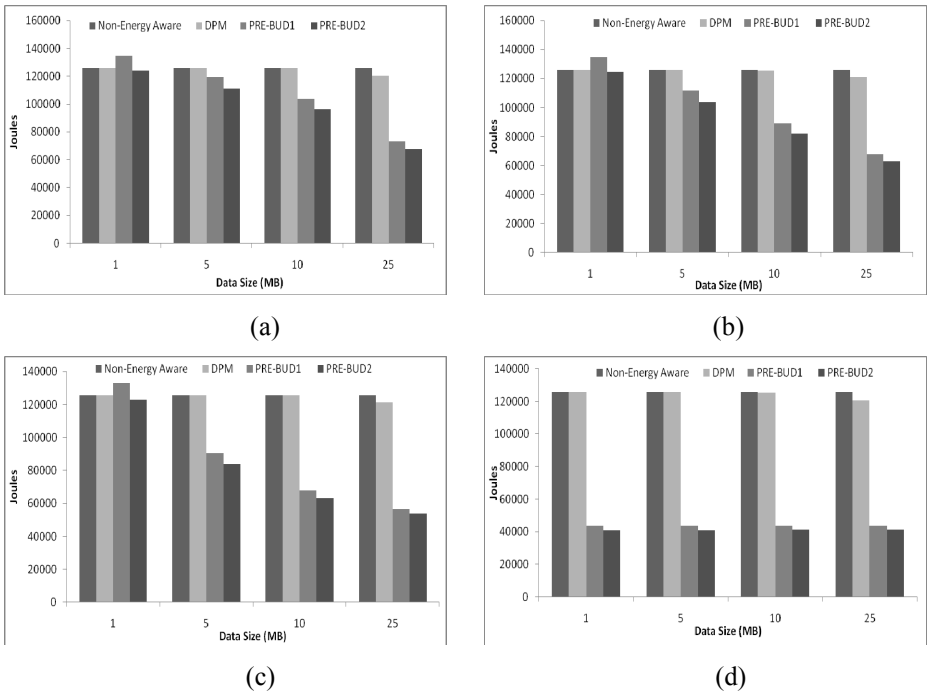


Figure 4.6: Total Energy Consumption of Disk System while Data Size is varied for four different values of the hit rate: (a) 85 %, (b) 90 %, (c) 95 %, and (d) 100%

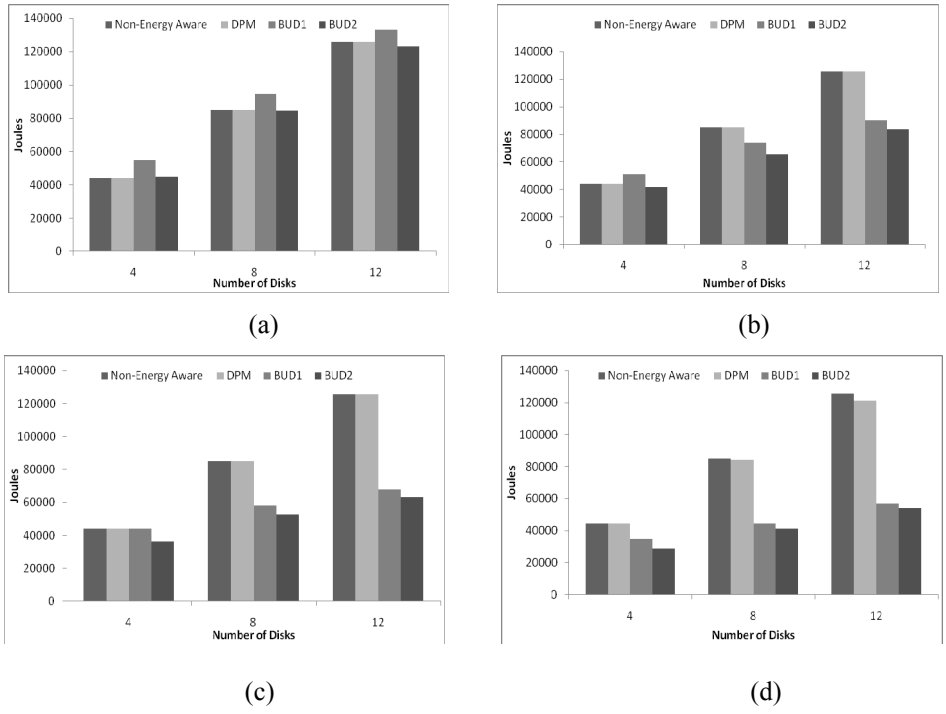


Figure 4.7: Total Energy Consumption of Disk System while the number of data disks is varied. Data size is fixed at: (a) 1MB, (b) 5MB, (c) 10MB, and (d) 25MB

achieve this high energy efficiency. If the number of data disks is increased, we must be sure that the performance is not negatively impacted. When more data disks are added into a parallel disk system, the buffer disk is more likely to become the performance bottleneck. Moreover, Figure 4.7 shows that a large data size makes PRE-BUD more energy efficient. This result is consistent with that plotted in Figure 4.6.

4.4.5 Impact of Hit Rate

In this set of experiments we chose to investigate the impact the buffer disk hit rate has on the energy efficiency of the parallel disk system. Again, the data size is varied from 1 to 25 MB. The number of data disks is set to 12. We observe from Figure 4.8 that higher hit rates enable PRE-BUD to save more energy in the parallel disk system. This is expected because with a high hit rate, we heavily load the buffer disk while allowing data disks to be transitioned to the standby state. A low hit rate means a data disk must be frequently spun up to serve requests, incurring energy penalties. The longer a disk can stay in the standby state, the more energy efficient a parallel disk will be. Note that hit rates of 100% are not realistically achievable if the disk requests require all disks to be active. A 100% hit rate can only be accomplished if the overall load on the entire disk system is fairly light. It has been documented that some parallel workloads are heavily skewed towards a small percentage of the workload, thereby making 80% hit rates feasible. With the varying data sizes, we notice that the energy savings becomes more significant for larger data sizes. Having a larger data size is similar to increasing the hit rate of buffer disks operating on smaller data sizes.

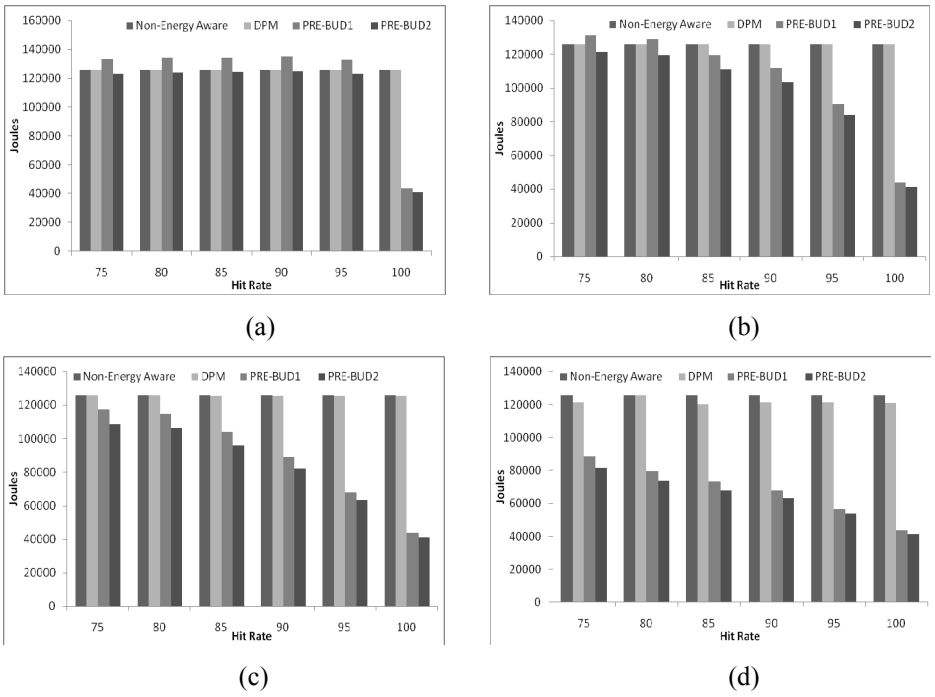


Figure 4.8: Total Energy Consumption for different hit rate values where the data size is fixed at: (a) 1MB, (b) 5MB, (c) 10MB, and (d) 25 MB

4.4.6 Impact of Inter-Arrival Delays

In these experiments, we study the impact that the inter-arrival rates of the requests have on the energy savings of PRE-BUD. Figure 4.9 shows the energy consumption totals of the disk system with four different values of the inter-arrival delay. The number of disks was fixed at 12, the data size was fixed at 1MB, and the hit rate was varied from 85% to 100%. When there is no inter-arrival delay, DPM will not yield any energy savings. This is because there are no idle-windows large enough for disks to spin down. PRE-BUD1 ends up consuming more energy than DPM in this case, because PRE-BUD1 adds the over-head of an extra disk and the energy required to prefetch the data. PRE-BUD2 is the most energy efficient, since there is no need to add an extra disk. If the inter-arrival delay is 100 ms, we have a similar situation, except that PRE-BUD1 is now able to produce a small amount of energy savings.

When the inter-arrival delay becomes 500 ms, DPM begins to produce energy savings. However, such energy savings pales in comparison to PRE-BUD. When the delay is increased to 1 Sec., the results look similar to the results for a 500 ms delay. Although DPM in this case can result in more energy savings, PRE-BUD1 and PRE-BUD2 significantly outperform DPM in terms of energy efficiency. These results fit our intuition about the behavior of the PRE-BUD approach. DPM needs large idle times between consecutive requests to achieve energy savings, heavily depending on the break even time of a particular hard drive. PRE-BUD is more energy efficient than DPM, because PRE-BUD proactively provides data disks with larger idle windows by redirecting requests to the buffer disk.

4.4.7 Power State Transitions

In this section of our study, we investigate the relationship between the number of power state transitions and energy efficiency. Figure 4.10 depicts the number of

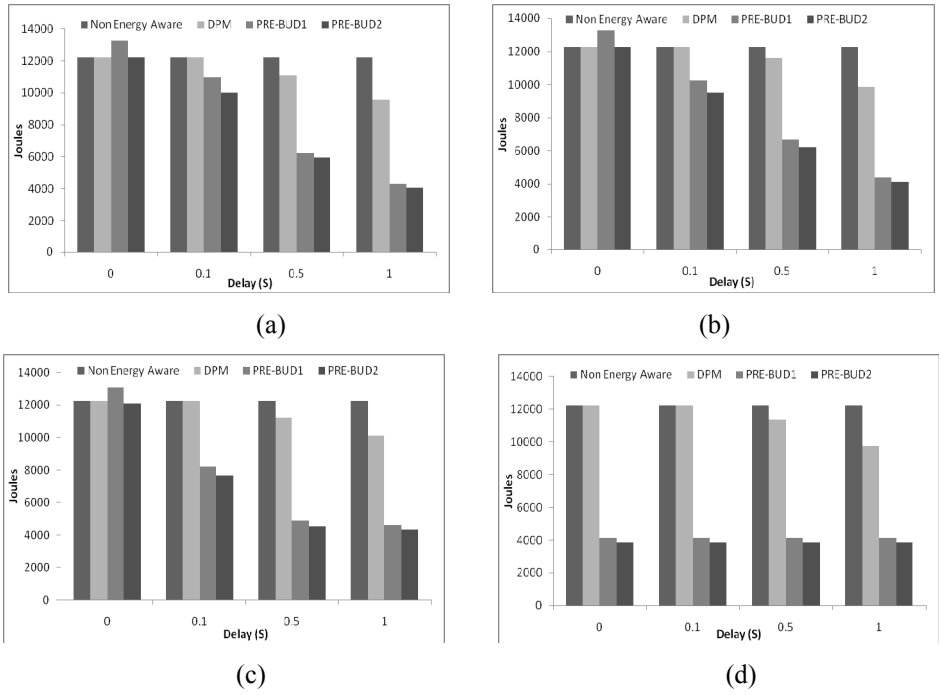


Figure 4.9: Total Energy Consumption for different delay values where the hit rate is (a) 85%, (b) 90%, (c) 95%, and (d) 100%.

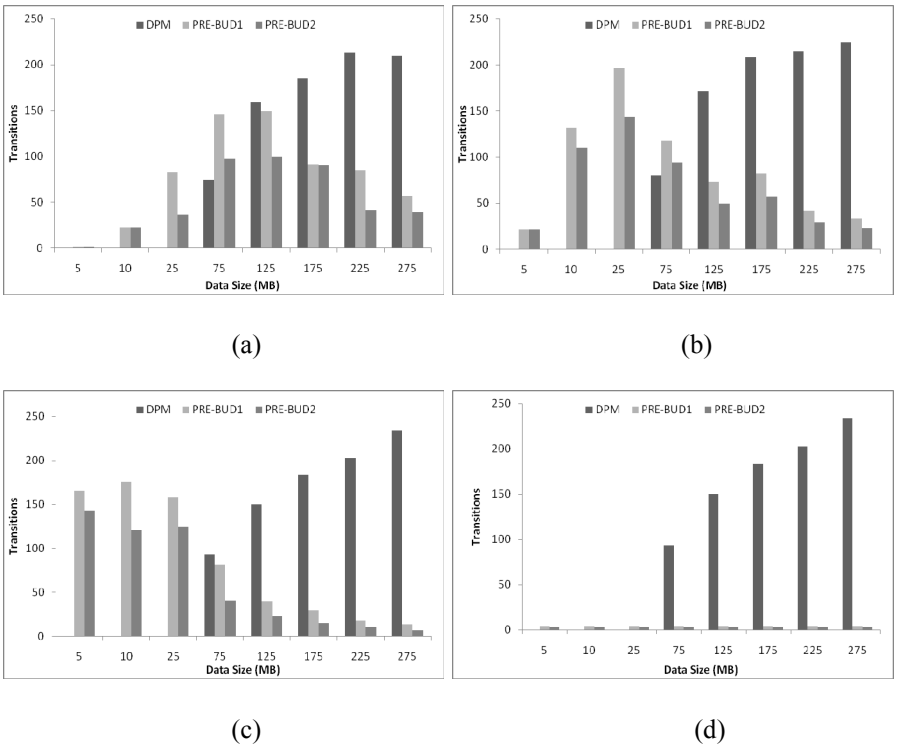


Figure 4.10: Total disk state transitions for different data sizes where the hit rate is: (a) 85%, (b) 90%, (c) 95%, and (b) 100%

power state transitions triggered by DPM, PRE-BUD1, and PRE-BUD2 when the data size and hit rate are varied. The number of state transitions caused by DPM is zero when the data size is smaller than or equal to 25MB. There is no power state transitions for small data sizes, because no idle time periods of data disks are long enough for DPM to justify transitioning to the standby state. When the data size is larger than 25MB, the number of power state transitions quickly rises with increasing data sizes. If DPM triggers transitions, it is able to improve the energy efficiency of the disk system.

Interestingly, the number of transitions for PRE-BUD slowly increases at first and then starts dropping when the data size is larger than 125MB. The transition number increases when data size is small because many small idle periods in data disks are merged by PRE-BUD creating new opportunities for data disks to sleep.

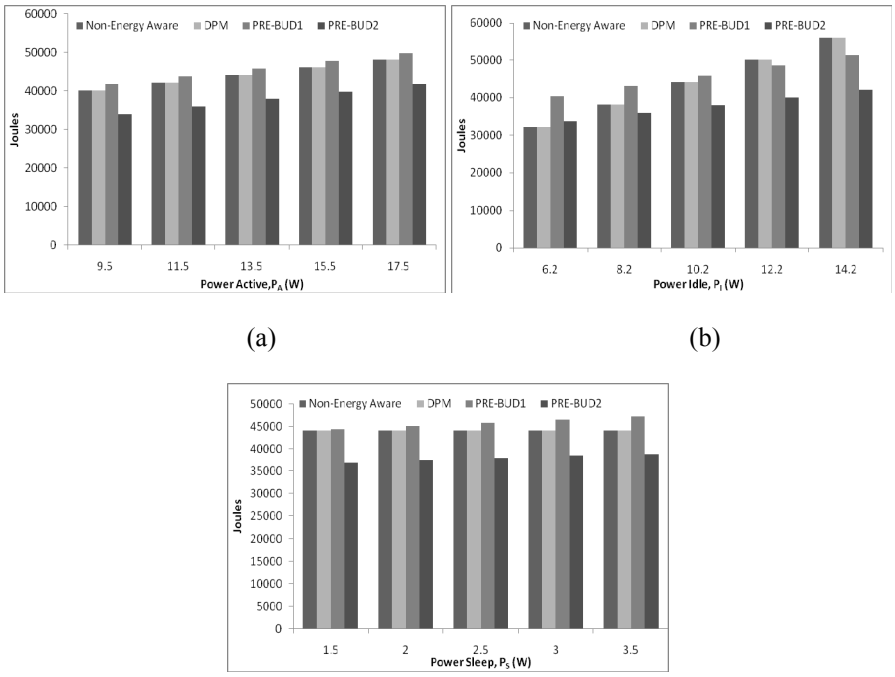
Since the small idle intervals tend to be spread out data disks experience many power state transitions. The buffer disk reduces the number of transitions for data disks when the data size is large, because a buffer disk generates larger and fewer idle time periods in data disks. A few very large idle time periods lead to a small number of transitions.

One of the problems with DPM is that it will transition a disk many times, which may decrease the reliability of the disk. Unlike DPM, PRE-BUD can improve the reliability of the disk system by lowering the number of transitions when data sizes of requests are very large. As such, PRE-BUD is conducive to improving both energy efficiency and reliability for data-intensive applications with large data requests.

4.4.8 Impact of Disk Power characteristics

To examine the effect that manipulating disk power characteristics has on PRE-BUD, we varied the active power, idle power, and standby power, for three separate experiments respectively. The number of data disks is fixed at 4 and the data size is 25MB.

Figure 4.11(a) shows that for all the four schemes, increasing the active power of a disk results in a continuous increase of energy consumption across the four different strategies. Results plotted in Figure 4.11(a) indicate that PRE-BUD is more energy efficient for parallel disks with low active power. For example, if the active power is 9.5W, PRE-BUD2 saves 15.1% of the energy consumption total over DPM. If the active power is increased to 17.5W, then PRE-BUD2 improves energy efficiency over DPM by only 13.0%. Figure 4.11(b) shows the impact of varying the idle power parameter of a disk has on the energy efficiency of PRE-BUD. Compared with active power, idle power has a greater impact on the energy savings achieved by PRE-BUD. If the idle power is very low, PRE-BUD2 has a negative impact. If the idle power is increased to 14.2 W, PRE-BUD2 can save energy over DPM by 25%. Figure 4.11(c)



(a) (b)

(c)

Figure 4.11: Total Energy consumption for various values of the following disk parameters: (a) power active, (b) power idle, and (c) power standby

shows that standby power also has a significant impact on PRE-BUD. Specifically, the energy savings starts at 16.3% and drops to 11.7% with increasing standby power.

These results illustrated in Figure 4.11 indicate that parallel disks with low active power, high idle power, and low standby power can produce the best energy-saving benefit. This is because PRE-BUD allows disks to be spun down in standby during times they would be idle using DPM. The greater the discrepancy between idle and standby power, the more beneficial PRE-BUD becomes. Lowering active power also makes PRE-BUD more energy efficient because the amount of energy consumed prefetching and serving requests can be reduced.

Throughout our experiments it was realized that the main factor limiting the energy savings potential of PRE-BUD is the large break-even times of disks. A large break-even time of a disk reduces opportunities for DPM to conserve energy if there are a large number of idle periods that are smaller than the break-even time. PRE-BUD alleviates this problem of DPM by combining idle periods to form large idle windows. Unfortunately, PRE-BUD inevitably reaches a critical point where energy savings are no longer possible. To further improve energy efficiency of PRE-BUD, we have to rely on disks that are able to quickly transition among power states - one of the dominating factors in energy savings for disks.

4.4.9 Real World Applications

To validate our results based on synthetic traces, we evaluated eight real-world application traces. The applications are parallel in nature; thus, all of the applications used eight disks, with the Titan and HTTP application being the exceptions and only used seven disks. Note that results plotted in Figure 4.12 generally represented the worst case for PRE-BUD. Figure 4.12 shows that PRE-BUD1 consumes more energy than DPM for most applications except for the Cholesky and LU Decomposition applications. When applications are very I/O-intensive, adding an extra disk leaves

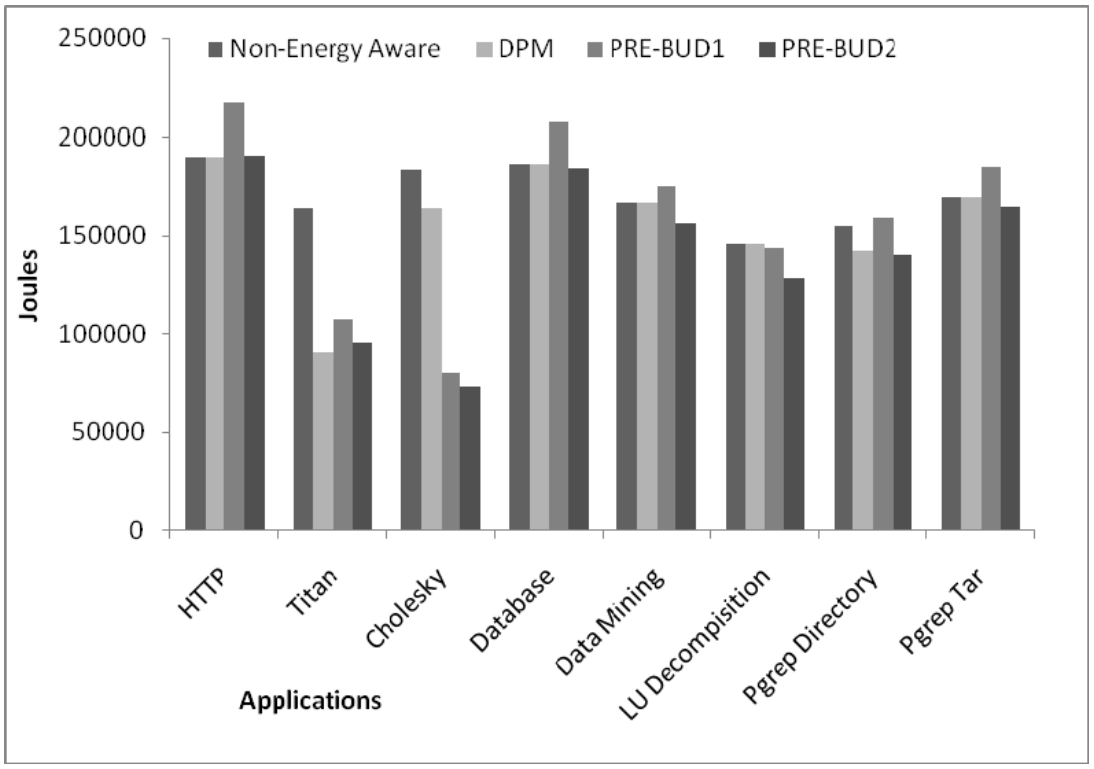


Figure 4.12: Total Energy Consumed for Real World Traces

no opportunity to conserve energy. Figure 4.12 also shows PRE-BUD2 noticeably improves energy efficiency over DPM for most applications. The results confirm that PRE-BUD can generally produce energy savings under both low and high disk workloads, even though the energy savings is relatively small for high workloads.

A surprising exception is the Titan application, because DPM is more energy efficient than PRE-BUD. In the Titan trace, there is one large gap between all of the consecutive requests, allowing DPM an opportunity to put all of the disks into the standby state for a long period of time. PRE-BUD, on the other hand, keeps the buffer disk active all the time to minimize the negative impact on performance. In this special case, the active buffer disk makes PRE-BUD less energy efficient than DPM. The energy efficiency of PRE-BUD can be further improved by aggressively transitioning the buffer disk to the standby state if it is sitting idle.

PRE-BUD Response Time Degradation	5 Disks	10 Disks	15 Disks	20 Disks
10% of Data Accessed in 90% of Trace	6 ms	16 ms	26 ms	36 ms
20% of Data Accessed in 80% of Trace	6 ms	16 ms	26 ms	36 ms
30% of Data Accessed in 70% of Trace	6 ms	16 ms	26 ms	36 ms
40% of Data Accessed in 60% of Trace	32 ms	47 ms	62 ms	79 ms

Table 4.4: Response Time Analysis

4.4.10 Response Time Analysis

In Table 4.4 we present our response time analysis results for the PRE-BUD strategy. We used four different traces, which had a designated set of popular data that varied in size and overall percentage of the entire trace. We also varied the number of data disks that each buffer disk is responsible for prefetching data from. From the table we see that the first three traces have similar response time results for each number of data disks used for the experiments. This tells us that our PRE-BUD strategy is capable of balancing the load and producing energy savings with a minimal impact on the response time of the parallel disk system. For the last trace,

in which 40% of the data is accessed 60% in the trace, we see that our response time degradation is significantly higher when compared to the other traces. This result is expected because the workload does not have an easily identifiable subset of data that can be prefetched to produce energy savings. PRE-BUD relies on the fact that some parallel application I/O operations are heavily skewed towards a small subset of data. From all of the results presented in Table 4.4 we realize that the PRE-BUD strategy produces relatively small response time degradations. This means our strategy will work for applications that can tolerate response degradations and is not suitable for real time applications.

4.5 Chapter Summary

The use of large-scale parallel I/O systems continues to rise as the demand for information systems with large capacities grows. Parallel disk I/O systems combine smaller disks to achieve large capacities. A challenging problem is that large-scale disk systems can be extremely energy inefficient. The energy consumption rates are rising as disks become faster and disk systems are scaled up. The goal of this study is to improve the energy efficiency of a parallel I/O system using a buffer disk to which frequently accessed data are prefetched.

In this chapter, we develop an energy-efficient prefetching algorithm (PRE-BUD) for parallel I/O systems with buffer disks. Two buffer disk configurations considered in our study are (1) adding an extra buffer disk to accommodate prefetched data and (2) utilizing an existing disk as the buffer disk. Prefetching data blocks in the buffer disk provides ample opportunities to increase idle periods in data disks, thereby facilitating long standby times of disks. Although the first buffer disk configuration may consume more energy due to the energy overhead introduced by an extra disk, it does not compromise the capacity of the disk system. The second buffer disk configuration

lowers the capacity of the parallel disk system, but it is more cost-effective and energy-efficient than the first one. Compared with existing energy saving strategies for parallel I/O systems, PRE-BUD exhibits the following appealing features: (1) it is conducive to achieving substantial energy savings for both large and small read requests, (2) it is able to positively impact the reliability of parallel disk systems by the virtue of reducing the number of power state transitions, (3) it prefetches data into a buffer disk without affecting the data layout of any data disks, (4) it does not require any changes to be made to the overall architecture of an existing parallel I/O system, and (5) it does not involve complicated metadata management for large-scale parallel I/O systems.

There are three possible future research directions for extending PRE-BUD. First, we will improve the scalability of PRE-BUD by adding more than one buffer disk to the parallel I/O system. This can be implemented by considering a buffer disk controller which manages various buffer disks each responsible for a set of data disks. In this work we investigate the relationship between buffer disks and data disks, to improve the parallelism of PRE-BUD we need to investigate the relationship between a buffer disk controller and the buffer disks. The number of buffer disks will have to be increased as the scale of the disk system is increased. Second, PRE-BUD will be integrated with the dynamic speed control or DRPM [9] for parallel disks. Last but not least, we will quantitatively study the reliability impacts of PRE-BUD on parallel I/O systems.

Chapter 5

DiskSim Power Models

5.1 Introduction

The previous chapter explored some powerful simulation models and we implemented these models in a simulator that we have developed. It is important to validate the simulator that one has developed and this can be a lengthy process, so we have decided to leverage DiskSim to continue our research. DiskSim is a validated disk system simulator developed by the parallel data lab of Carnegie Mellon University [3]. If we make any changes we would only need to validate the changes that were made to the simulator and not worry about the disk simulation details because they have been detailed by the DiskSim project. DiskSim also collects a large amount of information about disk usage that was not developed for our simulator and this information can be leveraged to further improve the quality of the research.

5.2 Simulation Framework

In order to complete the work for our project we have two main components in our system. The DiskSim simulator, which is responsible for simulating the operation of all the disks and the movement of data blocks in the system, and a block to file translator that is responsible for mapping data blocks to files in our storage system. It is important for us to investigate data movement at the file system for several key reasons. The first reason that we want to develop a storage system simulator at the file system level is that a disk must see a large idle period before it can be placed into the standby state to conserve energy. Data blocks are typically small and it

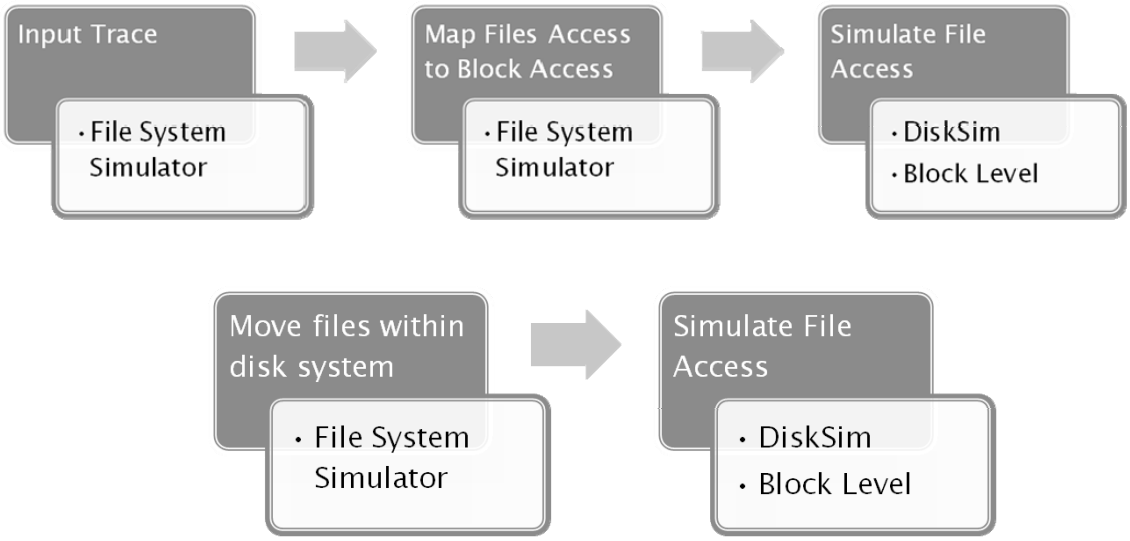


Figure 5.1: File System Simulator and Disk System Interaction

would require us to prefetch a large number of data blocks that could potentially be changed using writes. Secondly, due to the observations made in the previous chapters we have decided that our work is most suited for web applications and the cluster storage systems that support large volumes of data. Data blocks are typically managed within a single computer and we need a higher level mechanism to manage data across cluster storage nodes.

The interaction of our file to block level translator and a modified version of DiskSim is outlined in Figure 5.1. The modifications made to DiskSim are outlined in Section 5.4. The process begins with the use of a file-system level trace, generated using our trace generator, or from a real-world application trace. We then use our file to block level translator to generate a block level trace for DiskSim. The file to block level translator keeps tracks of all the files in the storage system and which node they are located on and simulates the movement of files across nodes. Once the block level trace is produced it is used by DiskSim to collect power and response time statistics. The movement of files between disks and nodes in the storage system is important to support our energy-efficient pre-fetching strategies, which rely on data movement.

5.3 DiskSim Limitations

DiskSim is a powerful tool for simulating the operation of disks in large scale storage systems, but it has one fundamental flaw that limits its use for our research purposes, there are no energy models for disk systems. There have been two research papers that have implemented power models within Diskim[29] [38]. We were fortunate that the authors of the SODA paper were able to provided us with source code of power models developed for and older version (version 2.0) of DiskSim. This got us closer to our goal of implementing power models into DiskSim, but we had to adapt the energy models in SODA to a newer version, 4.0, of DiskSim and we also had to simulate Disk state transitions which are critical to our disk energy savings strategies.

5.4 DiskSim Modifications

Parameter	Description
dm_power_active	Power Consumption in the Active State
dm_power_idle	Power Consumption in the Idle State
dm_power_standby	Power Consumption in the Standby State
dm_spin_down_time	Time to transition from Actve/Idle to Standby
dm_spin_up_time	Time to transition from Standby to Active/Idle
dm_spin_down_penalty	Energy required to transition from Actve/Idle to Standby
dm_spin_up_penalty	Energy required to transition from Standby to Actve/Idle

Table 5.1: Key Model Variables Added To DiskSim

The first area where we had to make changes was in the disk model code of DiskSim. The major changes are included in the Appendix A. The current disk models did not support our energy efficiency models so we had to add seven key parameters into the disk model to support our power requirements. The parameters added are summarized in Table 5.1. The first parameter that we have added is the power active, dm_power_active, and this is how much power is consumed once the disk is busy working on a request. The second is the idle power, dm_power_idle, which is

how much energy the disk consumes when the platters are spinning but the disk is not serving a request. The next parameter, dm_power_standby, is how much energy the disk consumes when disk is transitioned into the standby state and the platters are now stopped. The fourth parameter and fifth parameters, dm_spin_down_time and dm_spin_up_time, are the amount of time it takes to spin down and spin up the disk respectively. Spinning down the disk takes it from the active/idle state to the standby state and spinning up the disk takes it from the standby state to the active/idle state. The last parameters are, dm_spin_down_penalty and dm_spin_up_penalty, which are the energy requirements to spin down and spin up the disk respectively.

DiskSim is a discrete event based simulator, so modifications had to be made to support disk state transitions. The first area that we had to modify was to implement a timer to check to see if the disks have been idle for a certain period of time. This was implemented by adding a TIMER_EXPIRED event to the event queue of DiskSim and this code is outlined in the Appendix A with the disksim_power.c code that I wrote. This timer runs every 500ms and then runs the disk_update_idle function which is used to check all of the disks that are currently being simulated and check how long they have been idle. If the disks have been idle longer than an idle threshold value then the disks are transitioned into the standby state if they are currently in the active/idle state. To make the state transition complete I implemented another event type in DiskSim named the SPIN_DOWN_COMP event which is executed after the spin down time penalty is complete. Once this event is pulled off of the DiskSim event queue then the disk is placed into the standby state.

To wake up a disk I had to modify code in the disksim_diskctlr.c file. This file manages the disk and controls the disk when a read or write request makes it to the disk. If a request comes in for a disk that is currently in the standby state the wakeup_disk_sleep function is called in the disk_sim_power.c file. This function calculates how much time it will take to wake up a disk using the spin up penalty

and schedules a wake up complete event, SPIN_UP_COMP, that is similar to the SPIN_DOWN_COMP code described in the previous paragraph. There are also two special cases to check, the first if the disk is currently being spun up, and second if the disk is being spun down. The first case adds the difference of the current time and the spin up complete event as delay and keeps moving through the rest of the code. The second case requires letting the spin down complete and then scheduling a spin up complete event, and adding the delay for the spin down and spin up to complete. All of this code is detailed in the disksim_power.c file in the appendix.

The last area that had to be modified was to keep track of the energy consumed in the various states that the disk could be in. The disk would be active as its seeking, rotating, and transferring the data, the code for this is contained in disksim_disk.c, and these times were passed into the activeEnergyStat function in disksim_power.c. The idle time must also be accounted for and the idle time is gathered before a seek is found in disksim_diskctlr.c and passed to the idleEnergyStat function which is contained in disksim_power.c. The last place to check the idle time is when the simulation has finished we check the difference between the end of the simulation and the last I/O request for a disk in disksim_disk.c and calculate the idle energy using idleEnergyStat. The last thing to note is that we must also take care to measure the amount of time that the disk has spent in the standby state and this is controlled by the wakeup disk functions in the disksim_power.c. After the trace is finished the total standby time is multiplied by the standby state energy to produce a total energy consumption number in Joules. Similar calculations are made for the idle and active times and these numbers are combined to produce the overall energy output of the disk system.

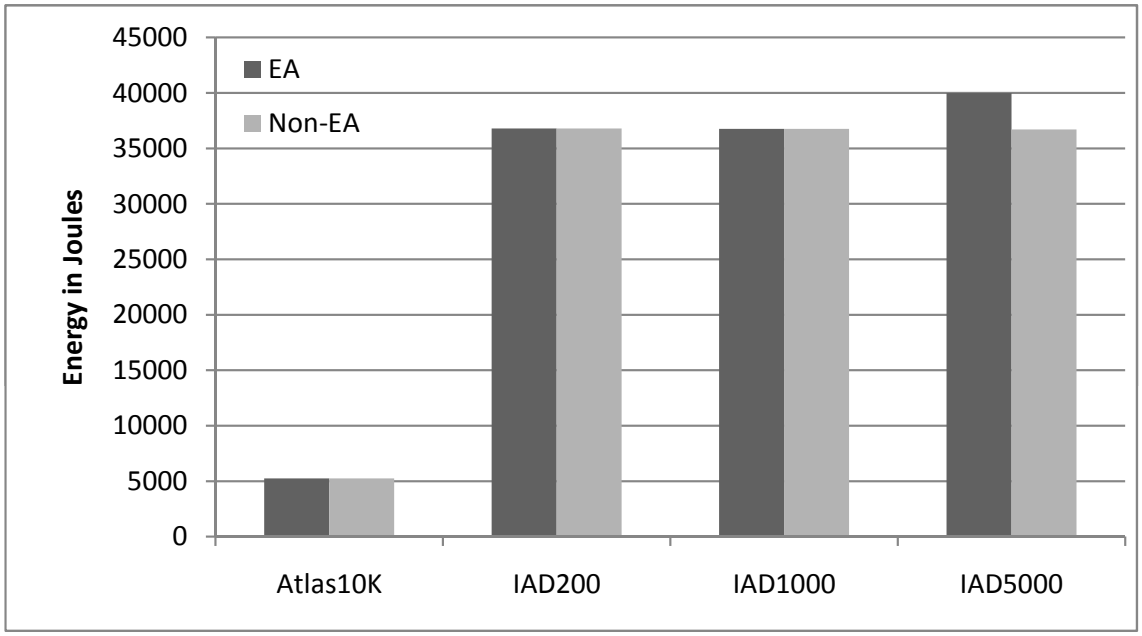


Figure 5.2: Energy Consumption Results of Modified DiskSim

Response Times (ms)		
Trace	Energy Aware	Non-Energy Aware
Atlas10k	3.97	3.97
IAD200	2.65	2.65
IAD1000	2.66	2.66
IAD5000	4461	2.59

Table 5.2: DiskSim Response Time Results

5.5 Generated Results

Figure 5.2 presents the results we collected when running our modified version of DiskSim and Table 5.2 presents the response times we collected. The first trace that we used was the Atlas10K trace which is used for validating the DiskSim simulation environment. We made our changes to the disk model for the Atlas10K trace and ran the trace with the modified and unmodified versions of the code. The energy consumption and response time results were exactly the same for the Energy and Non-Energy aware approaches. This result was expected because we noticed from the simulation results that the disk was not being transitioned into the standby state

because the inter-arrival delay of requests in the trace was very low. Since the disk was not being transitioned to the idle state this validated that our changes made no negative impact to Disksim when no energy savings opportunities were available.

The disk will transition to the idle state only after the idle threshold is reached, which is 5 seconds for our experiments, and the IAD200 and IAD1000 experiments produced no state transitions. IAD200 and IAD1000 are tested with synthetically generated traces with the inter-arrival delay set at 200 and 1000 ms, respectively. These traces produced results that are similar to the case for the Atlas10K trace and we are able to validate the operation of our modified version of DiskSim in the case that energy savings is not possible. The last results that we produced were using an inter-arrival delay of up to 5000 ms, which was able to produce 147 state transitions. Unfortunately the traces we were working with are at the block level and the data size of requests is relatively small, from 1 to 50 blocks. This caused the disks to be transitioned with a high probability that a request would arrive while the disk was transitioning to the standby state, which caused an enormous jump in the response time of the energy-aware approach. Using the disk parameters from Table 3.2 the time to spin down and then immediately spin up takes 12.4 seconds and incurs an energy penalty of 148 Joules. Since the disk spent relatively little time in the standby state the energy consumption of the energy aware approach is higher than the non-energy aware approach.

5.6 Conclusion

This chapter introduced an improved simulation framework that can be used to simulate energy aware disk systems. The framework that we have introduced in this chapter is valuable because DiskSim allows one to test a large number of disks in a quick manner. Along with our source modifications this allows us to quickly prototype large scale energy-aware storage systems. The disks must be spun down to

conserve energy and the currently used disk model has a high state transition penalty in terms of energy and time. If the disks are not managed carefully energy-aware approaches can quickly degrade response times and also consume more energy than non-energy aware approaches. DiskSim focuses on block level requests which are generally small and through our experimental results we have concluded that large data sizes are conducive to energy savings approaches that place a disk in the stanby state for energy savings.

To remedy these deficiencies we decided to move away from modeling and simulation and produce a prototype energy aware virtual file system. This will eliminate any errors that are introduced by disk power models and any disk simulators that have to be modified to support disk power models. Our goal was to develop a system that would be capable of producing energy savings and it was time to move towards a real-world implementation. A virtual file system will allow us to work with larger data sizes, place data across multiple storage nodes, and implement energy efficient strategies across all of the storage nodes.

Chapter 6

Energy Efficient Virtual File System

6.1 Introduction

Large-scale cluster storage systems are becoming ubiquitous because of the large amount of data required for search engines, multimedia websites, and data-intensive high-performance computing [10] [9]. These large-scale cluster storage systems typically are extremely inefficient concerning energy consumption. With data centers quickly growing in scale, it is important to develop energy-efficient tools to keep the cost of operating cluster storage systems down. Improving the energy efficiency of cluster storage systems is important because storage systems can account for 27% of the total cost to operate a data center [16]. The demands for increased performance and storage capacities of large-scale storage systems exacerbate the high energy consumption problems associated with cluster storage systems.

A handful of novel techniques developed to conserve energy in storage systems include dynamic power management schemes [6][19], power-aware cache management strategies [40], power-aware prefetching schemes [31], software-directed power management techniques [32], redundancy techniques [27] and multi-speed settings [11][14]. These energy-saving techniques can significantly enhance the energy efficiency of disk drives under workload conditions where idle periods between groups of disk accesses are substantial.

One fundamental drawback of cluster storage systems is that large data sets are partitioned and distributed across multiple storage nodes in a cluster; it is difficult for the storage nodes to energy-efficiently coordinate and handle parallel data management. A promising approach to improving the energy efficiency of cluster storage

systems is to implement energy-saving techniques in file systems. In the absence of an energy-efficient cluster file system, energy conservation is commonly achieved in individual storage nodes in a non-collaborative manner. Relying on low-power storage components to save energy in clusters not only limits I/O performance, but also loses opportunities to conserve energy by considering file accesses across multiple storage nodes.

The goal of our research is to develop an energy-efficient virtual file system - called EEVFS - for large computing clusters. EEVFS is a cluster file system that energy-efficiently processes file accesses across multiple storage nodes in a computing cluster. The salient features of EEVFS lie in its high energy efficiency, fast I/O processing, and scalability potential. In other words, EEVFS can provide significant energy savings for cluster storage systems while achieving high I/O performance.

EEVFS achieves high energy efficiency through a BUD disk architecture [21][28]. In the BUD architecture, each storage node contains m buffer disks and n data disks. We choose to use log disks as buffer disks in each storage node, because data can be written onto the log disks in a sequential manner to improve performance of the buffer disk. In most cases, the number of buffer disks m is smaller than the number of data disks n.

To fully utilize buffer disks, we have investigated an energy-aware prefetching strategy (see [21] for details on the prefetching algorithm called PRE-BUD) to dynamically fetch the most popular data into buffer disks, thereby making data disks stay in the standby mode for long period of time to conserve energy. We evaluated the impact the PRE-BUD algorithm on the overall energy efficiency of parallel disks within a storage node. The research on PRE-BUD has led us to discover that file access patterns, data size, inter-arrival delays, and disk drive energy parameters combine to produce opportunities to transition hard drives into lower energy consuming states. There is energy, performance, and reliability penalties associated with

transitioning disks into the various power states and; therefore, it is imperative to investigate techniques that are able to offset these penalties. It is desirable to minimize the amount of state transitions to provide a balance between the energy efficiency, performance, and reliability of parallel disks.

In our previous studies on PRE-BUD, we have conducted extensive simulations to estimate performance and energy-efficiency of our prefetching schemes. Simulation results show that PRE-BUD is conducive to conserving energy in parallel disks. These findings motivate us to build the EEVFS file system, in which an energy-efficient prefetching mechanism is implemented for cluster storage systems. EEVFS keeps track of file locations and disk states of all the storage nodes in the file system. The system architecture for the EEVFS file system contains two different components - storage servers and storage nodes. The EEVFS architecture details are further outlined in Section 6.2.1.

Apart from high energy efficiency, EEVFS has high scalability. This extreme scalability is possible, because EEVFS coordinates a large number of storage nodes, each of which is managing an array of disk drives (see Fig. 1). The EEVFS file system is running on storage nodes connected over a switching fabric. EEVFS is responsible for balancing the I/O load across storage nodes. The I/O load of individual disks within a storage node is balanced by storage node component of EEVFS.

The rest of the chapter is organized as follows: We discuss in Section 6.2 the design issues of EEVFS. Section 6.3 discusses the implementation decisions of EEVFS. Before discussing the performance evaluation, we present in Section 6.4 a testbed, metrics, and important parameters. Then, Section 6.5 shows experimental results. Finally, Section 6.6 concludes the chapter and presents our future research directions.

6.2 Design

We outline in this section the design issues of the Energy-Efficient Virtual File System (EEVFS). In this study, we paid particular attention to the implementation of energy-efficient prefetching with buffer disks in EEVFS.

6.2.1 System Architecture

Like PVFS, EEVFS was designed to improve the performance of cost-effective cluster storage systems. In addition to achieving high performance, reducing energy consumption in cluster storage systems is a primary design goal of EEVFS.

Fig. 6.1 illustrates the architecture of EEVFS, where nodes are divided into three main groups - compute nodes (clients), storage nodes, and a storage server. The storage server is responsible for handling incoming file requests for data reads or writes from the compute nodes. The storage server needs to determine the storage node that contains the data that is requested by a client. When the number of storage nodes scales up, the storage server might become a performance bottleneck, we address this issue by simplifying the functionality of the storage server. Thus, the storage server only has to manage metadata such as data location and file size. To achieve high scalability of cluster storage systems, we allow each storage node to manage (1) multiple data disks (see Section 6.2.2 below) and (2) metadata for the multiple local disks (see Section 6.3.4).

6.2.2 Data Placement

If the storage server is given previous knowledge about the popularity and access patterns of the data blocks, the server distributes the data blocks to storage nodes in a round-robin fashion based on file popularity. After the storage server has distributed the data across the storage nodes, it splits the file access patterns based on the data

distribution. The server then forwards the corresponding access patterns to each storage node.

A storage node manages the states of multiple hard drives and also performs load balancing based on the file popularities determined by the storage server. As data is placed on each storage node by the storage server, the storage node places the data on its N disks in a round-robin order. Since the first data placement request contains the most popular data, the second request contains the second most popular data, and so forth, the storage node load balances the data placement request on its local data disks. The storage node receives file access pattern information from the storage server about each file that is stored within the storage node.

6.2.3 Power Management

The storage node uses the file access pattern to predict periods when each of its data disks will be idle for long periods of time. If there are any periods of time larger than a threshold value, the storage node will transition a data disk into the standby period. The storage node uses an energy prediction model that takes into account the number of files to prefetch and the file access pattern. If there are consecutive requests for data in the predicted prefetch area then the storage node marks points in time when the data disks should be transitioned to the standby state to conserve energy.

Within each storage node the disks are separated into two groups, namely, buffer disks and data disks. The buffer disks are responsible for holding copies of popular data from the data disks. Our goal is to keep the buffer disk active and keep the data disks as lightly loaded as possible, thereby allowing EEVFS to transition data disks into the standby state to produce energy savings. The buffer disk used in the current incarnation of EEVFS relies on the local file system to manage buffers residing on the buffer disk. The buffer disk, of course, must constantly be available for the Linux

operating system running in the storage node. It is worth noting that placing the buffer disk into the standby state is not feasible under heavy loads, because power state transitions in the buffer disk can adversely affect the performance of the storage node. If the buffer disk has any available space, the free space should be used as a write buffer area for the other data disks contained in the storage node.

6.3 Implementation

We implemented a prototype of EEVFS on a cluster storage system. The implementation uses an append-only log of requests to keep track of file access patterns, which assists the storage server in determining the needs for prefetching popular files or data blocks from data disks to buffer disks. This section discusses several important implementation issues.

6.3.1 Process Flow

The process flow of EEVFS is presented in Fig. 2. The first step of the process is the initialization phase, which consists of the storage server connecting to all of the storage nodes in the system. The server creates a separate thread for each storage node and then establishes a TCP/IP connection to each storage node. The second step is that the storage server gets popularity information from a log of file access patterns. The prototype implementation uses a trace to replay file access patterns and bases the file popularity on information gathered from traces.

In step 3 files are created on the storage nodes and the server informs the storage nodes if they should perform prefetching, which is explained further in 6.3.2. The storage server attempts to load balance the files among the storage nodes based on the popularity information gained from step 2. The most popular data is placed on storage node 1 and the second most popular data is placed on storage node 2 and so on. The storage node also tries to load balance among the attached data disks.

This is achieved because the first create file request a storage node sees contains a file that is guaranteed to be more popular than the file contained in the second file create request. The first file a storage node creates is then placed on the first storage disk and the second file a storage node creates is placed on the second storage disk. In step 4 the server passes application hints to the storage nodes which is elaborated on in 6.3.3.

In step 5 the client requests information from a file and sends a request to the storage server node. The client can not access any of the content in the storage node without first going through the storage server node. The storage server node contains the storage node location of a file, but does not know which data disk the file is located on or if the file has been prefetched. The storage node passes information about the client to the storage node that contains the file and the storage node then establishes a connection with the client and passes the data to the client which is outlined in step 6.

6.3.2 Prefetching

Prefetching is an important part of the EEVFS architecture because it allows the data disk an opportunity to see large idle windows. If a buffer disk can serve a disk request then the corresponding data disk sees an idle window increase as opposed to the disk serving the request and resetting the idle window timer that each disk keeps. Our current version of prefetching is based on file access patterns and we derive a popularity based on the number of accesses over a given period of time. This information is passed to storage nodes and if they are instructed to prefetch then they will place a copy of popular data into the buffer disk. The current version of EEVFS uses the hard drive that also runs the operating system as the buffer disk, which allows us to use an existing disk as the buffer disk. If this is not possible due to space limitations it may be possible to add another extra disk to be used as the

buffer disk, but our experiments and previous simulation results indicate you would need many data disks to amortize the energy cost of adding an extra disk.

6.3.3 Application Hints

Assuming that the programmer of an application using EEVFS or the creator of files in the EEVFS system can pass information about the application that is going to be run on EEVFS we can further improve the energy efficiency of EEVFS. The application hints are used to predict idle windows to increase the energy efficiency of EEVFS while providing minimal delays to response time. The application hints can be extremely useful because they allow us to predict if there are any opportunities to save energy and if there are none then EEVFS will not place disks into the standby state. It allows EEVFS to operate in a more conservative manner as opposed to not knowing application hints and relying solely on the idle window timers. EEVFS can operate without the application hints, but there may be situations where a request comes in immediately after the idle window threshold is reached, causing a negative impact to energy savings and response time.

6.3.4 Distributed Metadata Management

To alleviate the metadata management burden of the storage server, we effectively distribute metadata across storage nodes in a cluster. The storage server simply manages metadata that provides hints as to which storage nodes contain files that can handle requests submitted from clients. Each storage node maintains metadata that can locate files on local disks in the node to respond requests forwarded from the storage server.

The goal of the distributed metadata management is to balance the metadata management load among the storage nodes. The storage server is unaware of the individual disks in each storage node, primarily acting as a load balancer and access

point for all of the storage nodes. The storage server does not need to know any information about the exact disk location of the data within each storage node. The implementation of our metadata management subsystem can be further improved in our future studies. For example, Miller *et al.* developed a scalable metadata management strategy for large-scale file systems [36]. We plan to integrate their dynamic metadata management scheme in our energy-efficient cluster storage system.

6.4 Evaluation Methodology

Parameter	Storage Server Node	Storage Node
CPU Type and Clock Speed	Celeron 2.2 GHz	Celeron 2.2 GHz
Memory (MB)	2000	2000
Network Interconnect (Mb/s)	1000	1000
Disk Type	SATA	SATA
Disk Capacity	160 Gbyes	480 Gbyes
Disk Bandwidth	126 MB/s	126 MB/s

Table 6.1: Configuration of the Testbed Nodes.

6.4.1 Testbed

We built a cluster storage system serving as a testbed to evaluate the energy efficiency and performance of the energy-efficient prefetching mechanism implemented in EEVFS. This cluster storage system was configured as follows at the time when we conducted the experiments (see Table 6.1 for details on the configuration of the testbed). In our storage cluster system, there is one server node and five storage nodes. The server node has a 2.2 GHz Celeron processor, 2 Gbytes of RAM, a 1 Gbits/sec Intel EtherExpress Pro Fast-Ethernet network card, and a 160 GB SATA disk. Each storage node has a 2.2 GHz Celeron 4 processor, 2 Gbytes of RAM, a 1 Gbits/sec Fast-Ethernet network card, and three 160 GB SATA disks. One disk is used as the operating system host and as the buffer disk while the other two disks are used as data disks. All the nodes were running Linux 2.4.20 rather than Linux

2.6, because we experienced disk transition inconsistencies running recent Linux 2.6 kernels.

6.4.2 System and Workload Parameters

For the collection of our experimental results we have focused on varying five key system and workload parameters (see Table 6.2) that noticeably affect both energy efficiency and performance of the EEVFS system. These five important parameters are: (1) average data size, (2) file access popularity (i.e., the MU value), (3) inter-arrival delays (i.e., arrival rate), (4) number of files to be fetched, and (5) disk idle threshold. In what follows, let us describe these parameters summarized in Table 6.2.

- **Data Size.** We conducted extensive experiments using both real-world traces (see Section 6.5.4) and synthetic file traces (see Sections 6.5.1 - 6.5.3). For synthetic file traces, the mean data size of files is varied from 1MB to 50MB. When it comes to real-world traces, data sizes are obtained from the traces.

- **File Popularity Rate - The MU Value.** The second parameter that we have chosen to vary is the MU value for the Poisson distribution of file requests that are fed into the storage server. This value was varied from 1 to 1000 and with 1 skewing the file accesses patterns to a small number of files and 1000 spreading out the distribution of files accessed.

- **Arrival Rate.** For synthetic traces, we used the inter-arrival delay to represent the arrival rate of file requests submitted from applications to the cluster storage system. We used four different synthetic workload scenarios by varying the inter-arrival delay of the file requests. We have added 0 to 1000 ms of inter-arrival delay between requests to represent lighter to heavier loads respectively. Note that we have also set a default inter-arrival delay at 700 ms to keep our

queue from growing too large and our response times growing too large for the energy and non-energy aware comparisons.

- **Number of Files to Prefetch.** The last parameter that we have varied is the number of files to prefetch and we have varied this from 10 to 100. The total number of files in our test file system is 1000 files for testing purposes. EEVFS with the prefetching flag set is represented as PF in the figures and NPF represents EEVFS without prefetching.

- **Disk Idle Threshold.** If disks are sitting idle for a certain period of time (i.e., Disk Idle Threshold), the disks are switched to the standby mode to conserve energy.

Parameter	Values
Data Size(MB)	1, 10, 25, 50
File Popularity Rate - The MU Value	1, 10, 100, 1000
Inter-arrival Delay(ms)	0, 350, 700, 1000
Number of Files to Prefetch	10, 40, 70, 100
Disk Idle Threshold (sec)	5

Table 6.2: System and Workload Parameters.

6.4.3 Metrics

To quantify the energy efficiency improvement and performance impacts of our prefetching scheme, we used the following three metrics in the experimental evaluation.

- **Energy Savings (see Section 6.5.1).** We compared the energy consumption of the cluster storage system with the energy-efficient prefetching mechanism against that of the same system without employing the prefetching mechanism.

- **Number of Power State Transitions (see Section 6.5.2).** The total number of power state transitions can closely reflect overhead incurred by switching

the power state of the disks between the active and standby mode. We evaluated the impacts of workload parameters on the overhead introduced by power state transitions.

- **Response Time (see Section 6.5.3).** Our energy-efficient prefetching mechanism aims to improve energy efficiency of cluster storage systems while minimizing performance penalties. We measured the performance penalties caused by the prefetching mechanism in terms of the increase in response time.

6.5 Experimental Results

6.5.1 Energy Savings

From Figure 6.5.1 we discover that EEVFS with prefetching significantly improves the energy efficiency of the disk system. Power measurements were collected from the individual storage client nodes running the experiments and combined for our results. Taking a look at Figure 6.5.1, which varies the data size used for the experiment; we realize that larger data sizes produce larger energy efficiency gains. If the data size is 1MB we produce an 11% energy efficiency gain and when the data size is 50MB the energy savings produced is 15%. The other interesting thing to note about the data size experiments is that the overall energy output of EEVFS with PF and no PF significantly increases when the data size is 50MB. This is produced because our default inter-arrival delay of 700 ms is too low and the queue for the storage client nodes becomes quite large and the test runs longer than the original trace time causing the overall energy output to increase. Even though the test ran longer for the PF and no PF cases the energy efficiency gain produce by EEVFS with PF was still the largest for 50 MB. For the data size experiments MU was fixed at 1000, the number of files to prefetch was 70, and the inter-arrival delay is set at 700ms.

Figure 6.5.1 shows the impact of popularity rate (i.e., the MU value) on the energy efficiency of the cluster storage system. From this figure we realize that the larger MU value produces a smaller energy efficiency gain. This is caused by the fact that many files are requested in the trace and the probability that the data required for the trace will be prefetched is smaller with larger values of MU. Using our default prefetch size value of 70 files we are able to produce the same amount of energy savings when MU is 100 or smaller. The reason for this similarity in energy consumption is the fact that when MU is 100 or smaller EEVFS is able to prefetch all of the required data and sleeps the disks at the beginning of the trace execution and is able to keep the disks in the standby state for the entirety of the trace. For the MU experiments the data size was fixed at 10 MB, the number of files to prefetch was 70, and the inter-arrival delay is set at 700ms.

Figure 6.5.1 reveals the impact of the inter-arrival delay on energy efficiency. The results plotted in Figure 6.5.1 indicate that we are able to produce larger energy efficiency gains when the inter-arrival delay is increased. Intuitively this makes sense because large inter-arrival delays produce lighter workloads. Light workloads generally produce more opportunities for the data disks to be placed into the standby state. The interesting thing to note is that the overall energy efficiency actually seemed to level off around the 700ms inter-arrival delay value. When the inter-arrival delay value is increased to 1000ms we actually see a small decrease in energy efficiency. This could likely be caused by the fact that we sleep a disk as a particular request enters the storage client node, and if the requests are spaced further apart it will take slightly longer for the disks to transition to the standby state. For the inter-arrival experiments the data size was fixed at 10 MB, the number of files to prefetch was 70, and MU is set to 1000.

Figure 6.5.1 evaluates the effect of the number of files to prefetch into a buffer disk. In this experiment, the data size was fixed at 10 MB, the inter-arrival delay is

700 ms, and MU is set to 1000. The results show that as the number of prefetched files is increased, EEVFS produces larger energy savings. When the number of files to prefetch is 10 (i.e., 1% of the total files in a storage node), our prefetching strategy can only improve energy efficiency by 3%. This result is expected because larger amounts of data prefetched increase the chance that EEVFS is able to serve a request from the buffer disk. Once the number of files to be prefetched is increased to 40 and above, the prefetching mechanism can provide significant energy savings due to the fact that a vast majority of requests can be served by the buffer disk.

6.5.2 Power State Transitions

Figure 6.5.2 displays the total number of state transitions for the data size experiment. For the data size experiments we notice that the number of state transitions decreases as the data size is increased. This result confirms that EEVFS can place the data disks into the standby state fewer times and for longer periods of time. This is intuitive because increasing the data size causes each request to be served longer and consecutive hits in the buffer disk produced large idle windows for the data disks.

Looking at Figure 6.5.2 it is interesting to note that no state transitions will produce no energy savings, so there is a balance between energy efficiency and the number of state transitions. We aim to minimize the number of disk spin up operations while being able to place the disks in the standby state for optimal energy savings. As the inter-arrival delay is increased, it produces a similar pattern as compared to the data size experiments. The number of state transitions decreases as the inter-arrival delay is increased. Similarly, as the energy savings is increased the number of state transitions is decreased due to the fact that larger inter-arrival delays produce lighter loads for the data disks in the storage client node.

Figures 6.5.2 and 6.5.2 show that the number of state transitions produced by varying MU and the number of files prefetched are very similar because they produce

a situation where the data disks are transitioned to the standby state for the entire trace. This occurs for small values of MU and for larger values of the number of files to prefetch. The interesting thing to note is that when the number of prefetch files is 10, this situation produces the largest amount of state transitions for all of the tests, 447. This same case also produced the smallest energy savings with only a 3% increase in energy efficiency. This small amount of energy savings may not be worth the stress put on the hard drives from the large amount of state changes. The idle threshold can be increased to prevent disks from transitioning frequently and producing a small amount of energy savings.

6.5.3 Response Times

Now we analyze the performance penalties caused by the prefetching mechanism. Figures 6.5.3 - 6.5.3 illustrate the increase in response time due to prefetching. The results collected, concerning MU and the number of files to prefetch, represent two special cases as indicated in the state transition results explanation. When the disks are able to stay in the standby state the entire time there is virtually no response time penalty. This is because the response time penalties are generally a product of the state transitions. If the number of state transitions rises it also causes a response time degradation. This is mainly due to the spin up operations, which average around 2 sec for the disks used in our experiments.

Figsures 6.5.3 and 6.5.3 show the effect of data size and inter-arrival delay on response time. From these two figures we can deduce that there is a linear relationship between the response time of the cluster storage system with prefetching and without prefetching. This is promising due to the fact that it shows that there is a tolerable response time penalty for producing energy efficiency gains.

The response times for the data size of 50 MB were omitted because of the fact that they were much larger than the other values because of the large amount of

queuing that took place on the storage server node. As the data size is increased we produce smaller penalties in the response time degradation. For the data size of 1MB we have a 121% increase in response time, 120 ms to 265 ms, but for the largest data size we produced only a 4% increase in response time degradation. We believe that the number of state transitions is closely related to the response time penalty and it is interesting to compare the results in Figures 6.5.3 - 6.5.3 with Figures 6.5.2 - 6.5.2. The inter-arrival delay response time pattern closely follows the pattern of the data-size, which is similar to the pattern in the results in the previous sections. As the inter-arrival delay is increased the response time decreases for the prefetching and non-prefetching versions of EEVFS. The response time degradation is 31% for the smallest inter-arrival delay value and 16% for the largest inter-arrival delay value. There seems to be a response time anomaly produced when the inter-arrival delay is 700 ms because the response time degradation is 37% at this point representing the largest response time degradation for the inter-arrival delay experiments. This performance degradation could be caused by the fact that the storage nodes attempt to predict idle window periods that are as large as possible, but aren't guaranteed to be the optimal solution. This might have produced a situation where the wake up transitions may have been skewed towards a disk that takes a longer time to transition from the standby to active/idle state.

6.5.4 Berkeley Web Trace Energy Consumption

The final figure, 6.15, we have produced presents a trace that was taken from the Berkeley file-system trace collection project [7]. The particular trace we used was a section of the web trace collection. For this experiment we set the data size to 10MB and kept the number of prefetch files to 70. The file access patterns were taken directly from the web trace collection but we modified the data size and the inter-arrival delay for requests to prevent a large amount of queuing on the storage

server. Based on the results In Fig. 6 we were able to produce a 17% energy efficiency improvement when prefetching was enabled in EEVFS. This represents a number that is near the maximum that we expect our current test bed to produce using EEVFS. After investigating the Berkeley web trace, it was discovered that we were able to place all of the data disks in the standby for the entirety of the Berkeley web trace. The web trace appeared to be skewed towards a smaller subset of data, but we were unable to find out how many files were contained in their file system.

6.6 Conclusion & Future Work

In this paper we introduced EEVFS - an energy-efficient virtual file system. Based on our experimental results we conclude that EEVFS can boost the energy efficiency of storage systems by more than 17%. We believe this number will increase as more disks are added to each EEVFS storage nodes. Although we were unable to test this theory using our existing testbed, we tested this theory using models and simulation. We evaluated energy efficiency and performance as functions of data size of files, popularity rate (i.e., the MU value), inter-arrival delay, and the number of files to prefetch. The metrics used in each experiment are energy consumption, the number of power state transitions, and response time. Our experimental results confirm that EEVFS is conducive to saving energy with a tolerable impact to the response time of disk requests.

For the future work we intend to develop EEVFS to be a production grade piece of software. We have currently investigated two approaches to improving EEVFS. The first approach involves extending PVFS to handle our energy management strategies. The second method is to extend the source code of EEVFS to make it robust. We also plan to investigate striping techniques within EEVFS that can help improve the performance of EEVFS, while still maintaining energy savings. EEVFS is a

distributed file system and we intend to investigate the performance of EEVFS in a large-scale distributed environment.

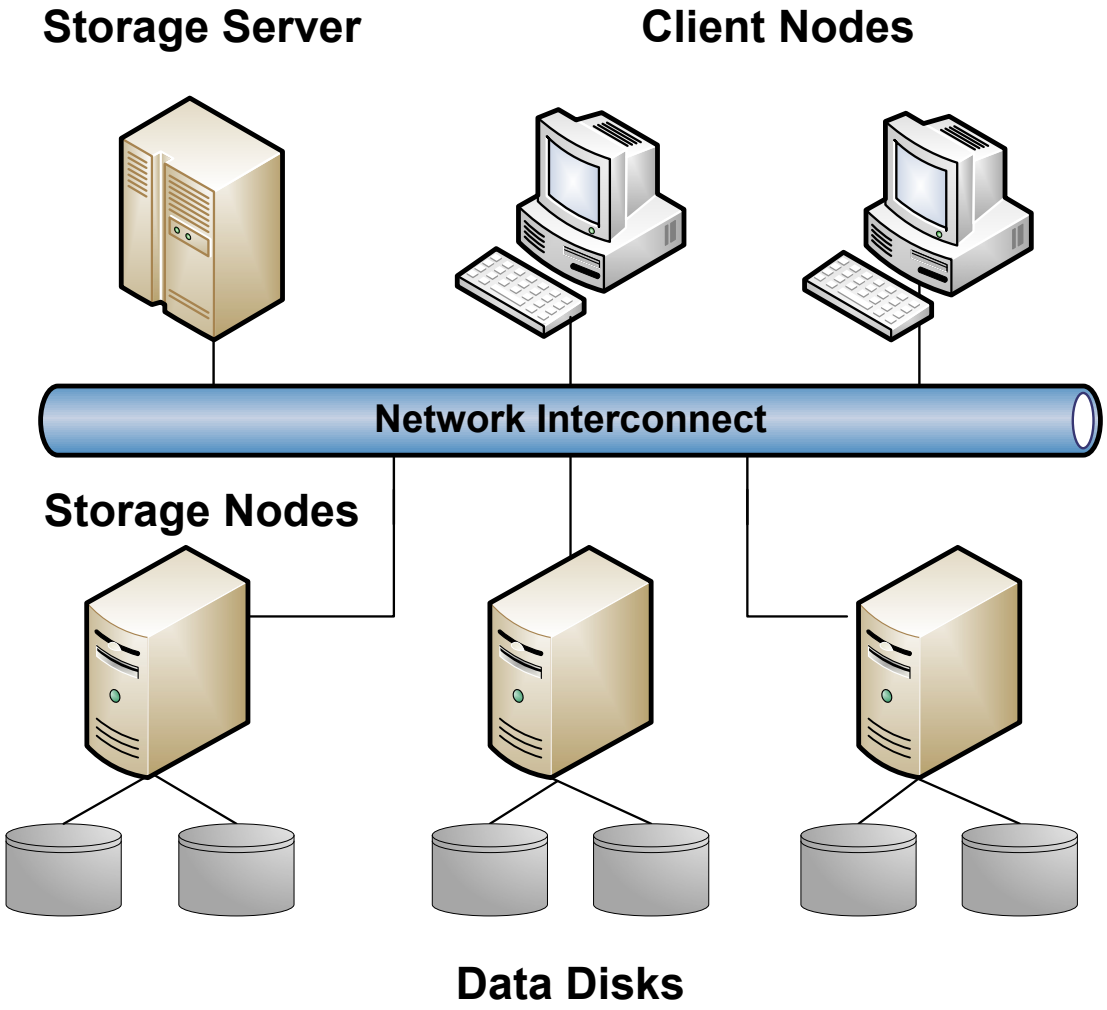


Figure 6.1: Architecture of EEVFS. The storage server manages metadata (e.g., data location and file size). Each storage node manages multiple disks, which are separated into two groups: buffer disks and data disks.Client nodes can directly access storage servers through the network interconnect.

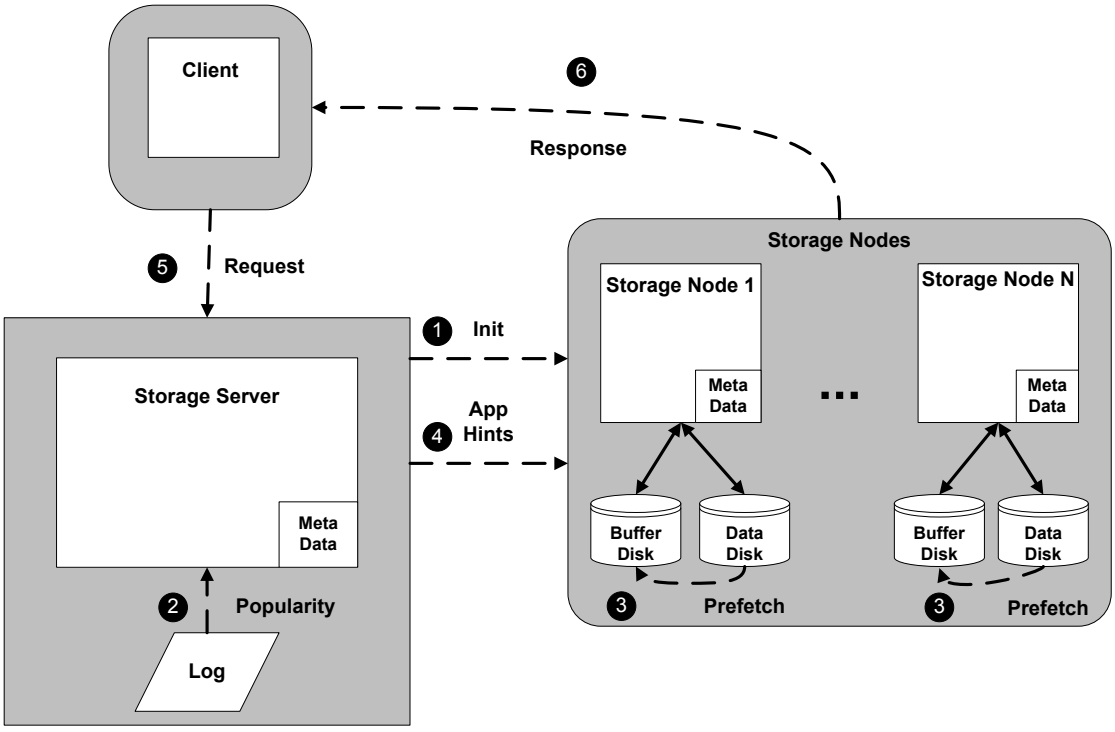


Figure 6.2: EEVFS Process Flow Chart. Step 1: initialization phase; step 2: storage server generates file popularity; Step 3: prefetch popular files from data disks to a buffer disk; Step 4: applications provide access hints; Step 5: applications submit file requests; Step 6: storage nodes return data to applications running on compute nodes (i.e., clients)

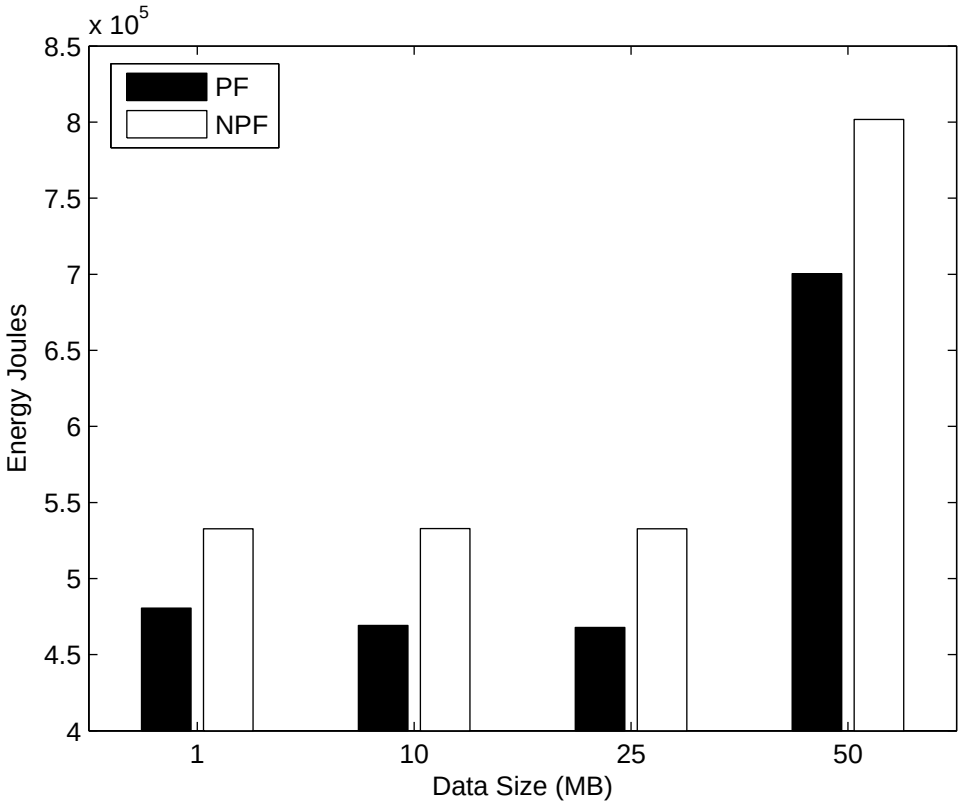


Figure 6.3: Data Size Varied

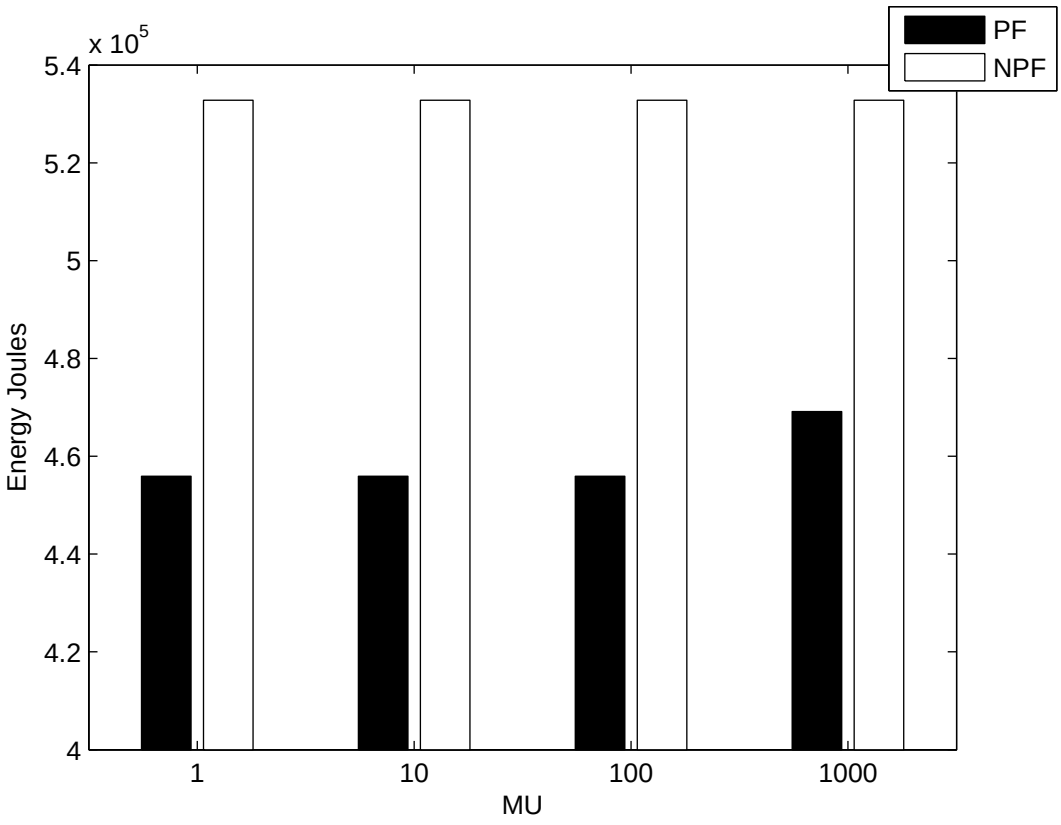


Figure 6.4: MU Varied

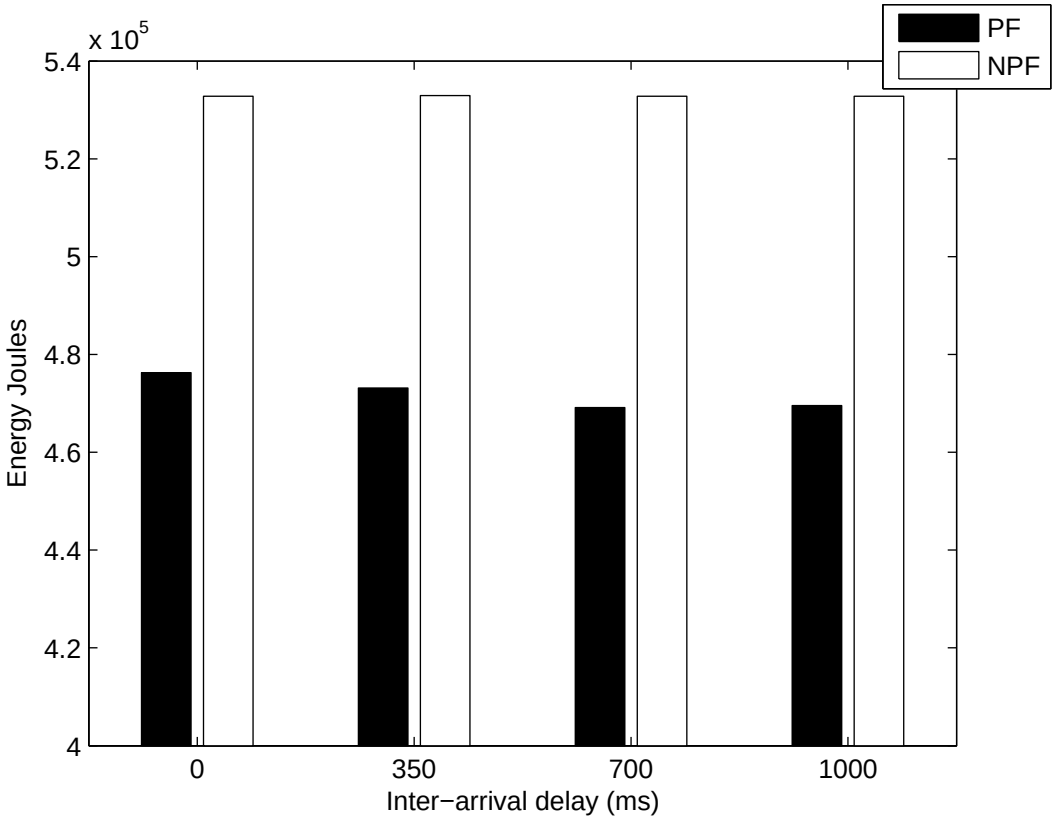


Figure 6.5: Inter-arrival Delay Varied

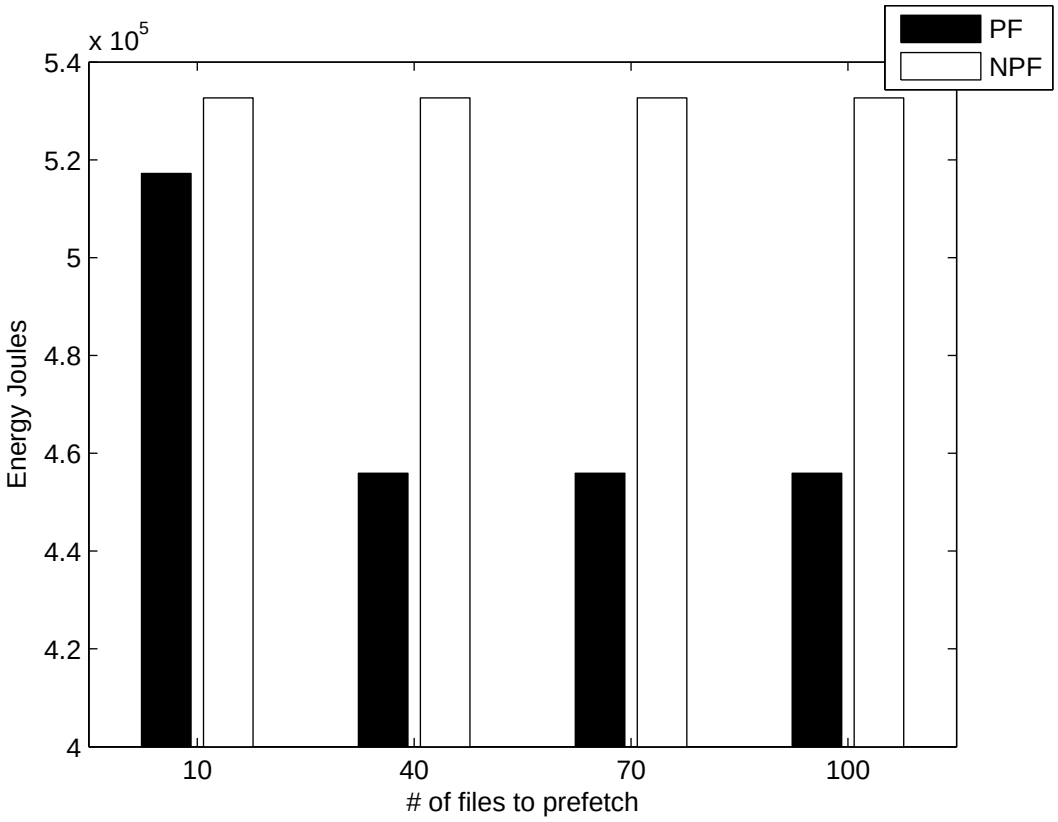


Figure 6.6: # of Prefetched Files Varied

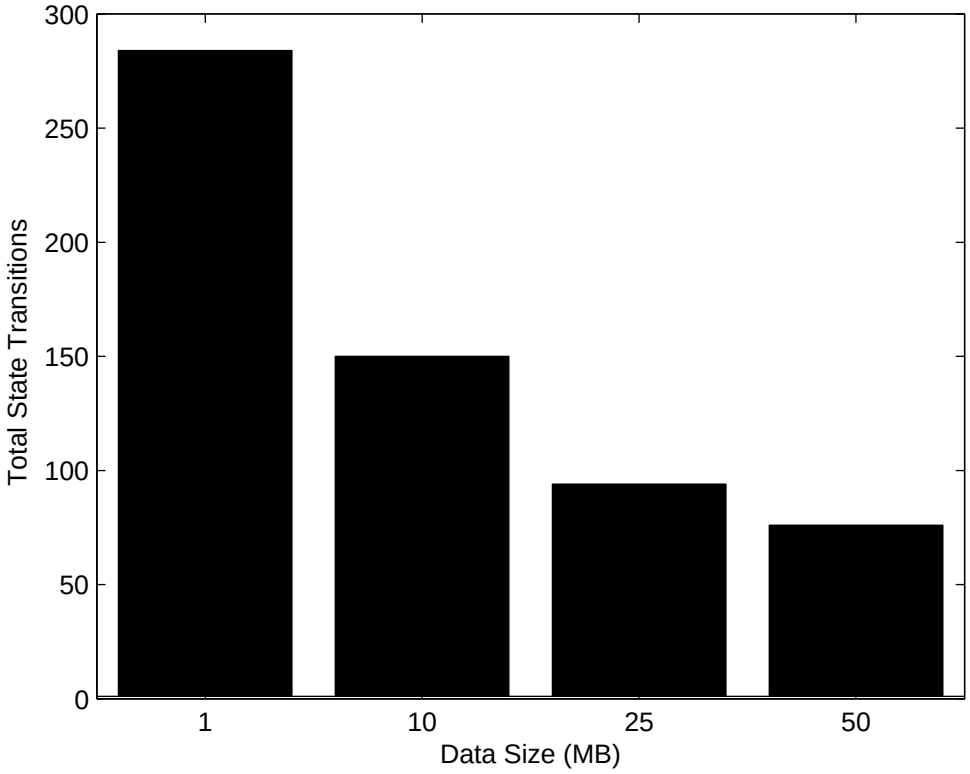


Figure 6.7: Data Size Varied

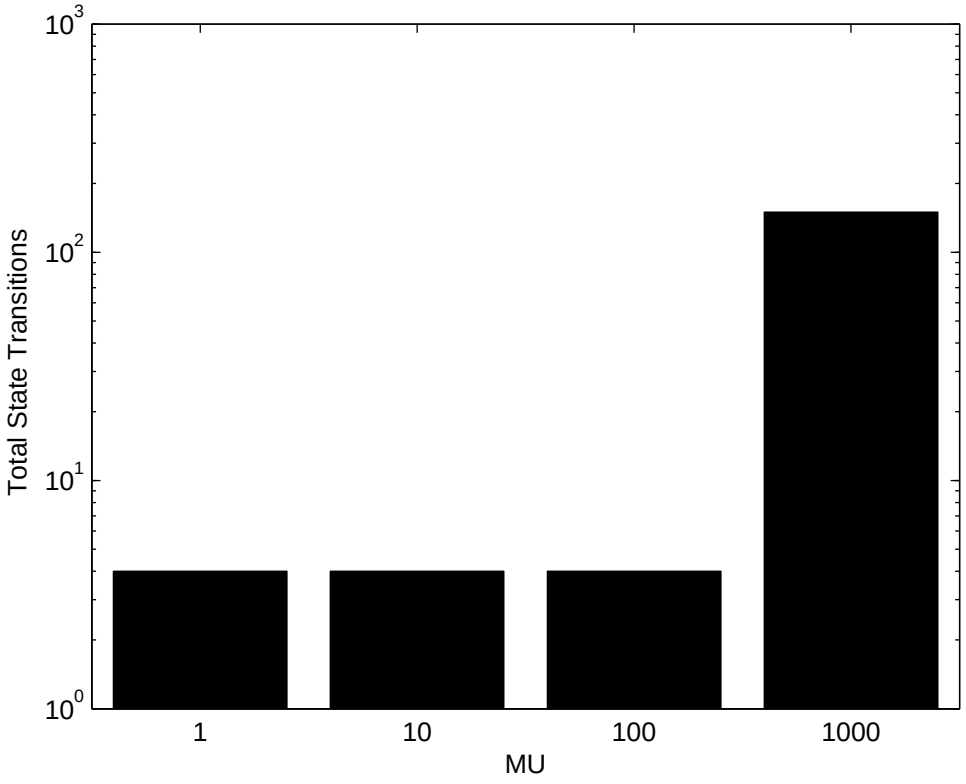


Figure 6.8: MU Varied

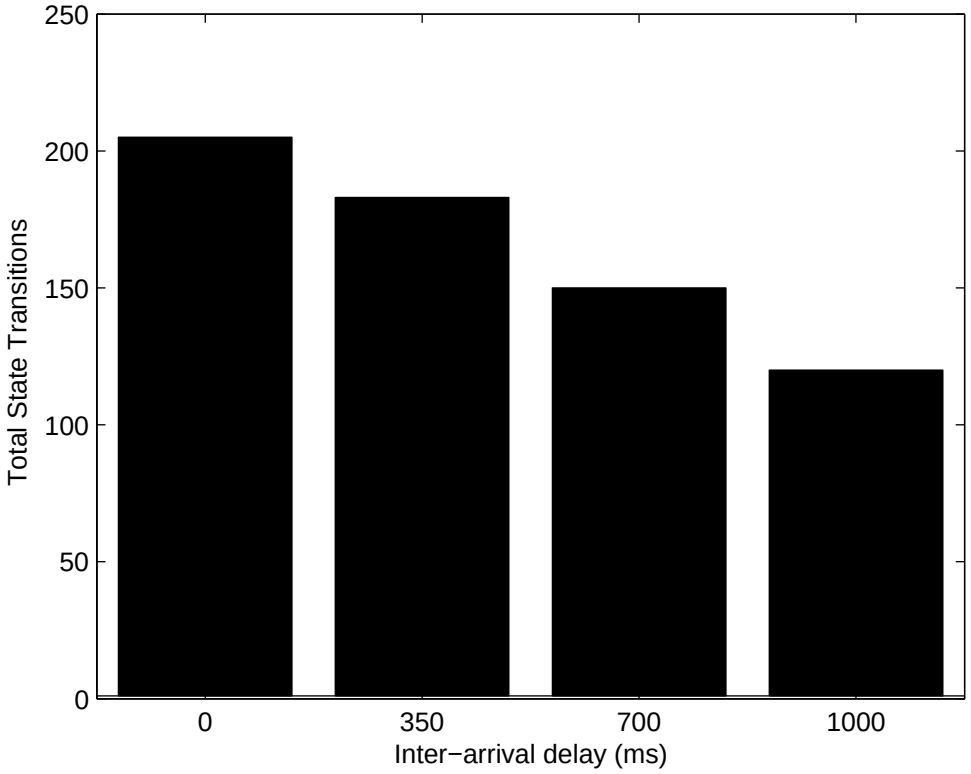


Figure 6.9: Inter-arrival Delay Varied

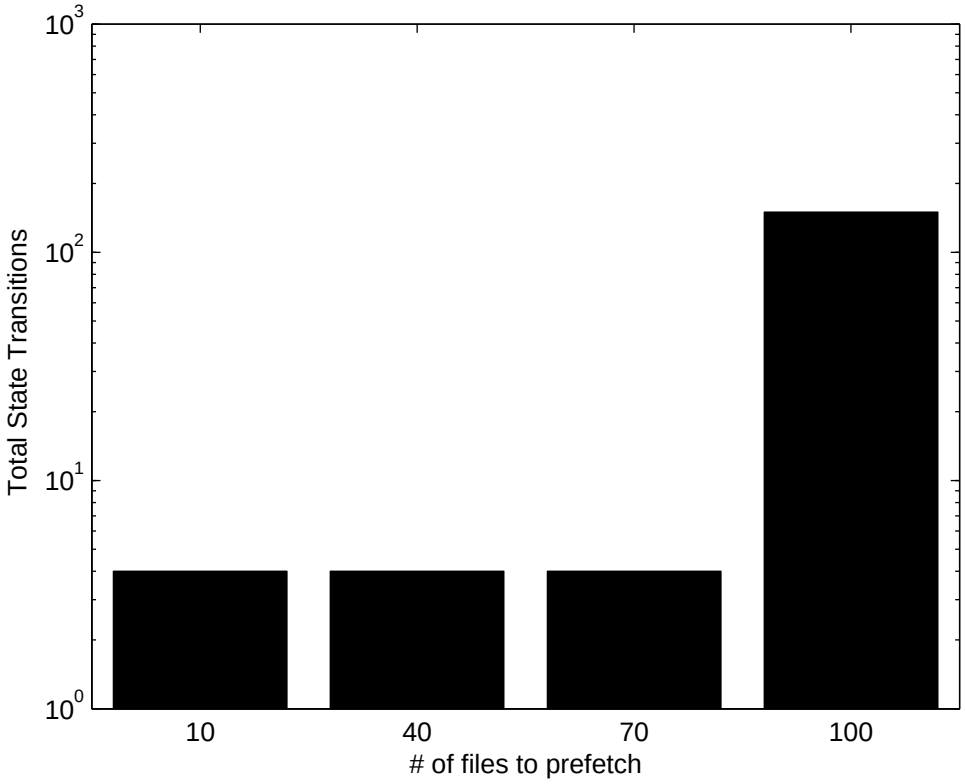


Figure 6.10: # of Prefetched Files Varied

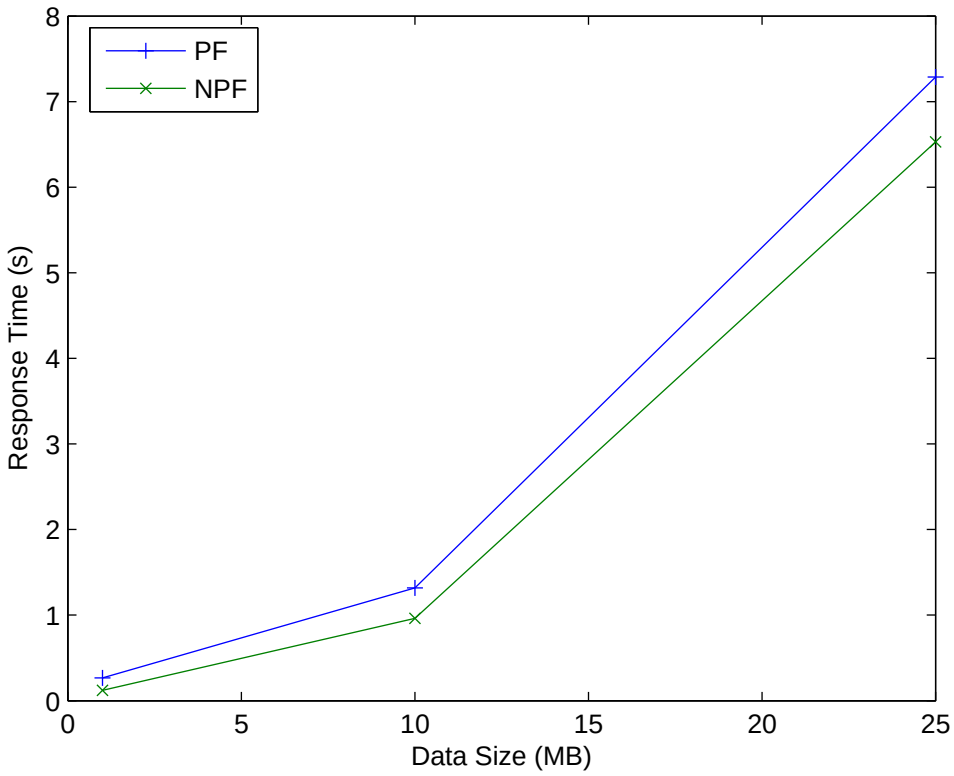


Figure 6.11: Data Size Varied

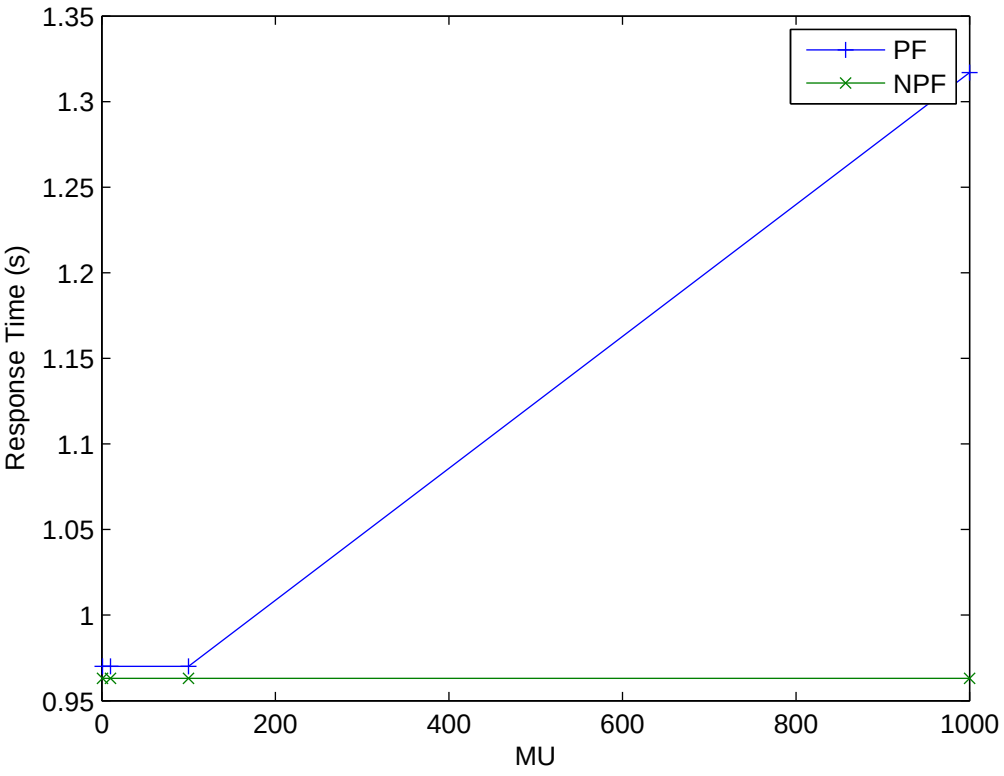


Figure 6.12: MU Varied

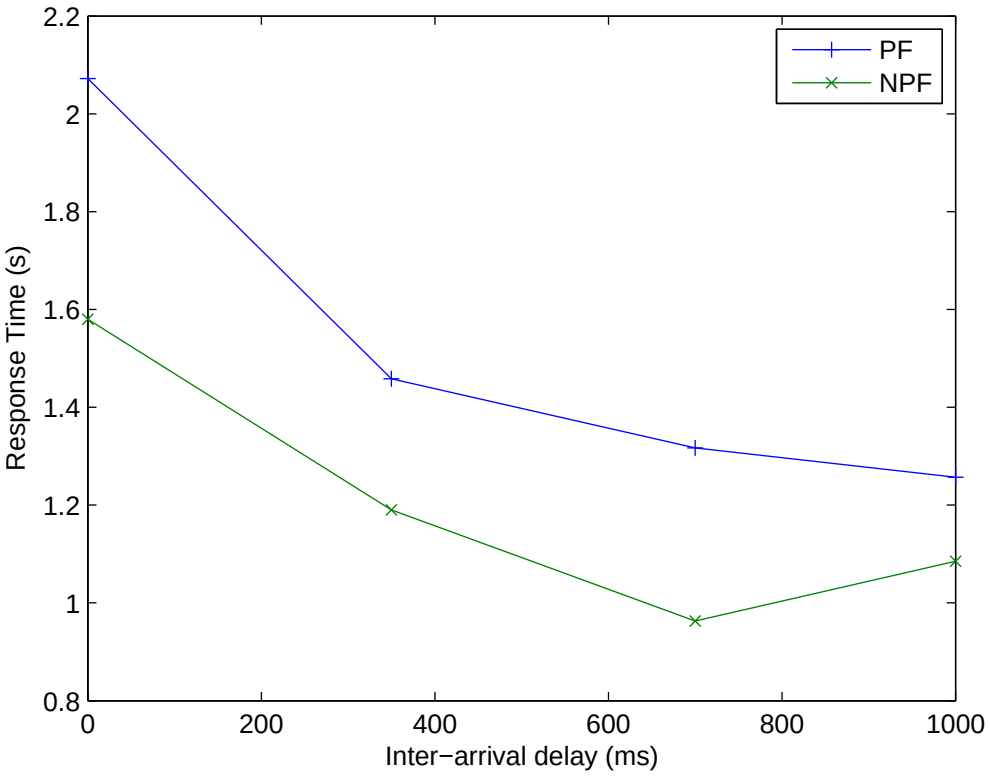


Figure 6.13: Inter-arrival Delay Varied

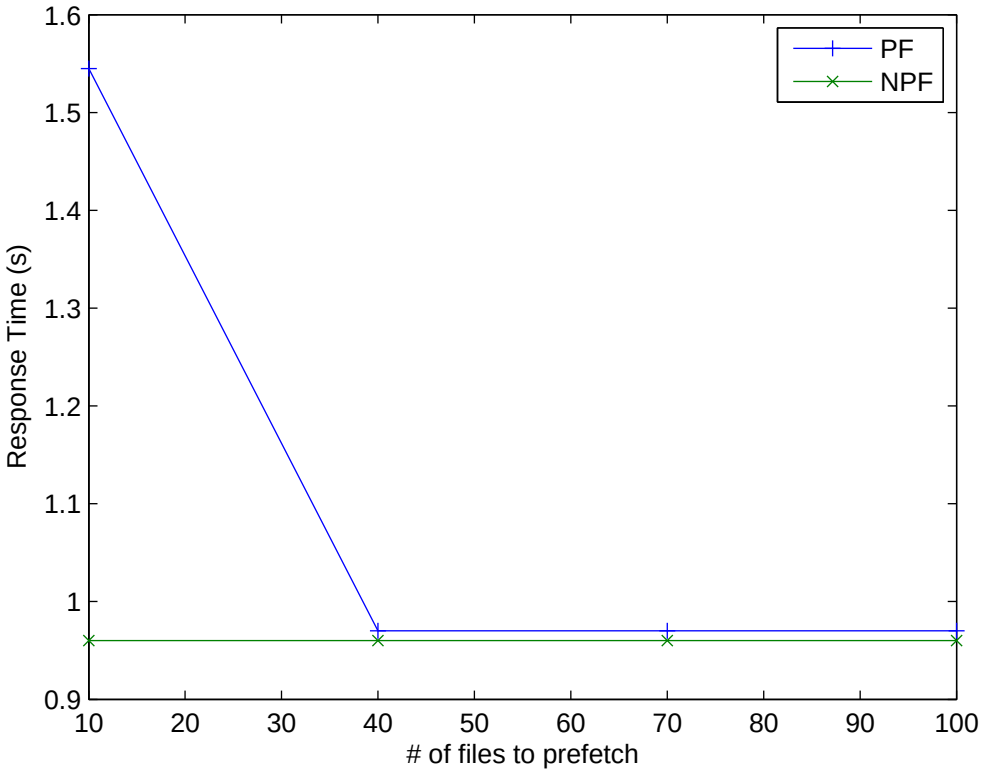


Figure 6.14: # of Prefetched Files Varied

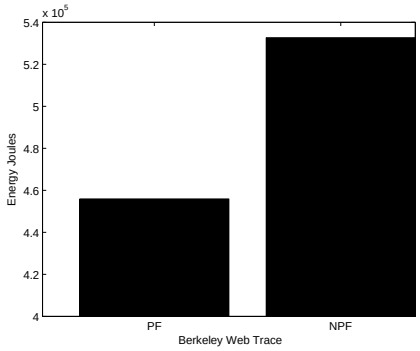


Figure 6.15: Energy Consumption of the tested cluster storage system when the Berkeley Web Trace are considered.

Chapter 7

Parallel Striping Groups for Energy Efficiency

This chapter leverages the work completed in Section 6 and particularly the use of the Energy-Efficient Virtual File system (EEVFS). To improve the performance of EEVFS we introduce in this chapter the novel idea of parallel striping groups. We break storage nodes into groups and stripe files across the groups. This allows us to control the aggregated bandwidth of each group of storage nodes which can be matched to the bottleneck of the system. Each group of disks also has buffer disks which can achieve energy savings and the striping groups help improve the performance of the storage system.

7.1 Parallel Striping Groups

Our first implementation of the EEVFS prototype placed a single file on a single disk and didn't support striping. Striping data across multiple disks is an important idea that has helped to improve the performance of parallel disk systems. This allows us to improve the performance of our EEVFS, while at the same time maintain comparable energy efficiency. It is important to balance the performance and energy efficiency of the file system because one usually comes at the expense of the other. Figure 6.1 provides the high level architecture of the EEVFS system and details can be found in Chapter 6

Figure 7.1 demonstrates the first level of our parallel striping architecture. Disks are grouped into a striping group, so that a file is striped across the data disks in one striping group. We have chosen to implement these striping groups for two main reasons. First striping the data across multiple disks improves the performance of the

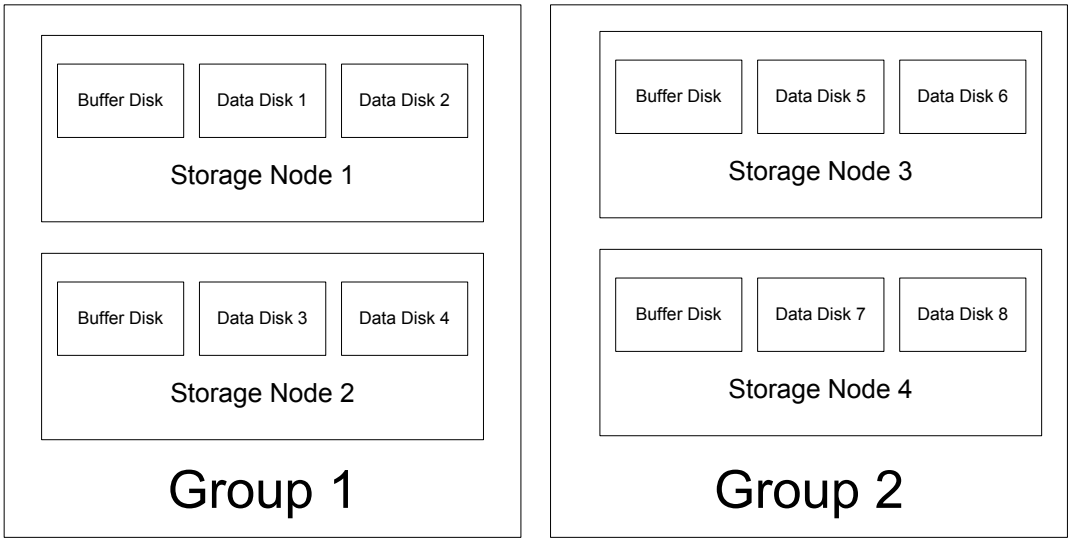


Figure 7.1: Parallel Striping Groups

disk system. Second we split the disks into groups to try to tune the performance of the striping group to the network performance of the particular install. One could stripe a file across all of the data disks in the disk system, but you may find that you encounter a bottleneck in your disk system interconnect. The striping groups are intended to be matched to a particular disk systems interconnect which would allow multiple striping groups to respond to requests at the same time.

Inside of each storage node we have multiple disks and they are broken into two different groups, data disks and buffer disks as shown in Figure 7.2. Data disks are responsible for storing the files and their associated data. They will always have a copy of a file and this copy will only disappear if the user wants to delete the file. When a file is created on a node, EEVFS places data on the data disks in a round robin fashion. This is intended to load balance the load of the data disks. The second type of disk on an EEVFS node is a buffer disk. The buffer disk is used to store a copy of popular data from the data disks to help improve the energy efficiency of the

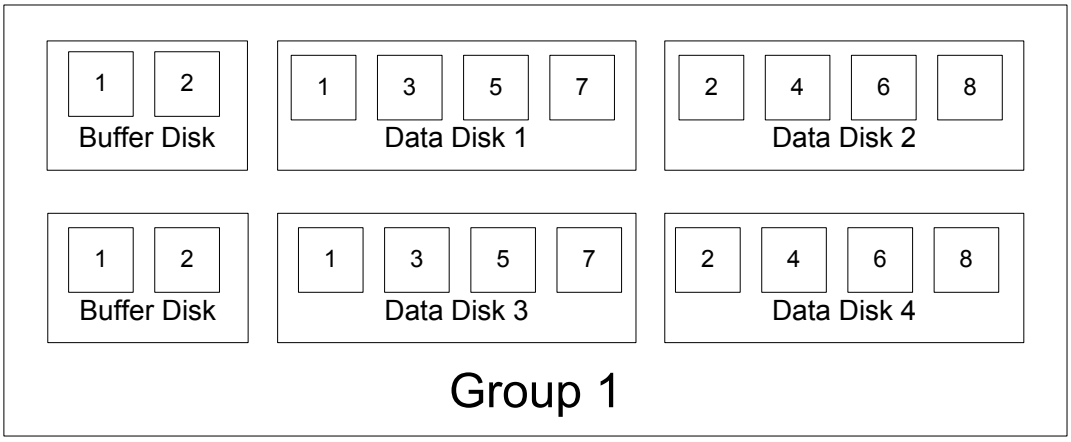


Figure 7.2: Data Striping Within a Group

disk system. If a buffer disk has a copy of a particular file and a request is made for the file, the buffer disk will serve the request. The corresponding data disk will not see the request, allowing the data disk to sit idle for a longer period of time.

If the data disk is idle for a certain period of time, the idle-threshold, then the data disk is transitioned into the sleep state. One has to carefully select the idle-threshold, if it is too low than the disks will spin-down too frequently causing performance degradation and increased energy consumption. If the idle-threshold is too high than the data disks will never transition to the sleep state and EEVFS will be unable to produce any energy savings.

7.2 EEVFS

To support parallel striping groups and our energy efficiency strategies we have created an Energy Aware Virtual File System (EEVFS). We believe a virtual file system implementation is important for several reasons. In the course of our research we have realized that it is not practical to implement our strategies at a disk block level because it is hard to track the popularity of blocks when they could hold different

data depending on how they are managed by higher levels of the operating system. This led us to start to investigate file systems and it was quickly realized that it may not be so feasible to develop a new file system from scratch that will work with the Linux kernel. After investigating virtual files systems, such as the Parallel Virtual File System (PVFS), we decided that a prototype virtual file system would be the most effective method to implement our energy efficiency strategies.

A virtual file system has several properties that led us to choose this method of implementation. First, the virtual file system code is independent of the kernel and this will allow us to simplify some aspects of our implementation and also allows the code to be more portable. Another major benefit of using a virtual-file system is that it will be independent of the underlying file system that is used on the Linux system. A user can determine the Linux file system type to use for the disk system and this allows more flexibility for implementers or our strategy.

The EEVFS system is split up into two groups of nodes, server nodes and storage nodes. The server node is in charge of presenting a user with a single point to access the data in all of the EEVFS storage nodes. The server node handles are operations associated with the data on the storage nodes. It is the single metadata manager for the files in the virtual file system, and its current form represents a single point of failure. We have recognized this shortcoming and plan to distribute the metadata across several nodes to improve future generations of the EEVFS. Since we are currently using a trace driven method to test the performance and energy efficiency of parallel striping groups in the EEVFS, the server node also plays the trace and sends requests to the EEVFS server process.

The server node is in charge of placing data among all of the storage nodes that are connected to the server. The current incarnation of the EEVFS server attempts to derive a popularity value for files based on information collected from traces. After the popularity value is created the EEVFS server than begins sending out file creation

requests to the storage nodes. The server starts a thread for each storage node and places a file creation request into n threads, where n is the stripe size being used in EEVFS. EEVFS will attempt to load balance the files by placing the most popular file on the first striping group and then placing the second most popular data on the second striping group.

The storage nodes in EEVFS are in charge of managing the data disks that are attached to the storage node. The storage client node manages the energy states of the disks and attempts to place idle disks in the standby period to conserve energy. EEVFS client nodes contain an idle threshold value that is compared against the idle time of each disk, if the idle time becomes larger than the idle threshold than the storage node places the disk in the standby state. Placing a disk in the standby state is time consuming and causes energy to be consumed once the disk must be woken up. The idle threshold must be chosen carefully because an aggressive idle threshold will transition disks too frequently causing a significant performance degradation and also causing the disk system to consume more energy. The storage node also attempts to load balance file creation requests among the disks contained in the storage node. Since the EEVFS server sends out file creation requests in order of popularity we can guarantee that the first file creation request seen on an EEVFS storage node is as popular as or more popular than the second file creation request. The storage node than places the first file creation request on the first disk and second file creation on the second disk and so on. This allows the storage node to load balance the files among the data disks on the storage node.

All of the storage nodes also contain a buffer disk which is used to place a copy of popular data spread among the storage disks in a particular storage node. The buffer disk in our particular example is the same disk running the operating system and it also happens to have space available. If space is not available it may be worthwhile to add an extra disk to be used as the buffer disk, but the energy cost must be offset

by the number of data disks you can place into the standby state if they are not being used. Our test bed allows us to only use 3 data disks in each storage node, so the buffer disk must also run the operating system for our implementation to be practical. If the server node indicates that the storage node should prefetch popular data then the storage node also places data on the buffer disk. This is done in order to provide the data disks with the illusion of longer idle periods, if a data request can be served by a buffer disk than the data disk remains idle. If the buffer disk was not present than the data disk idle counter would be reset. The longer the idle period that can be achieved provides the largest energy savings gains.

The storage node is in charge of the data placement and management of the disks and shields the storage server from these details. The storage server only needs to know which nodes contain which data, it is not concerned with the states of the disks in the disk system or which disk holds a file in a storage node.

7.3 Experimental Results

Parameter	Storage Server Node	Storage Node
CPU Type and Clock Speed	Celeron 2.2 GHz	Celeron 2.2 GHz
Memory (MB)	2000	2000
Network Interconnect (Mb/s)	1000	1000
Disk Type	SATA	SATA
Disk Capacity	160 Gbyes	480 Gbyes
Disk Bandwidth	126 MB/s	126 MB/s

Table 7.1: Configuration of the Testbed Nodes

The experimental results were collected on 6 PC's and their performance characteristics are outlined in Table 7.1. The PC's were all connected to the same Gigabit switch and with one node serving as the EEVFS storage server and the other 5 nodes are EEVFS storage nodes. The experimental results focus on varying four key parameters: the striping size of the data, the inter-arrival delay of the requests, the number of files to prefetch, and the MU value of the Poisson distribution used to generate

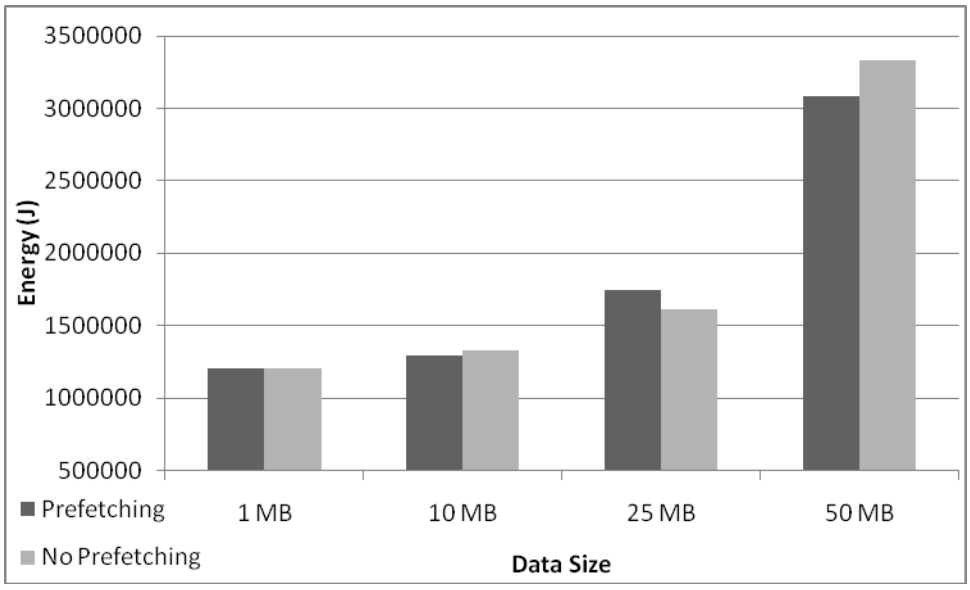


Figure 7.3: Energy Efficiency vs. Data Size

our synthetic traces. In addition we have also included a comparison against our EEVFS without striping groups. Response time graphs are also included to compare the effect that prefetching has on the performance of the disk system. Finally we present results of our system when we use a real-world trace based on the Berkeley web trace collection.

7.3.1 Impact of Data Size

The results in figure 7.3 allow us to make some very interesting observations. Using previous modeling and simulation techniques, we were able to deduce that as the data size increases it follows that our prefetching techniques will produce larger energy efficiency gains. This is caused by the fact that if the data size is larger it means that any particular disk will have to spend extra time reading/writing the file. If it takes longer to process requests for a file it follows that keeping a copy of data in the buffer disk will allow the data disk to sleep for longer periods of time. We are able to apply this intuitive explanation to the 1 MB, 10 MB, and 50 MB data

size results, which produce large energy efficiency gains as the data size is increased. The interesting example takes place when we used 25 MB for the data size. In this case our prefetching strategy actually causes the system to consume more energy as opposed to not prefetching data. At first we thought this was an anomaly and we re-ran our experiment to see if the results would change. We were surprised to realize that we consistently saw that in the 25 MB experiment our prefetching approach was expending more energy as compared to the non-prefetching approach.

We took a close look at our output results and realized that the experiment was actually running quite longer for the 25 MB experiment. This was due to the increased transition periods that were being caused by parameters in the EEVFS. The idle threshold of 5 seconds was too aggressive for this particular set of experimental parameters and was causing the requests to get delayed too often by spin up operations. This is a double negative for our energy consumption total because the longer the experiment runs the larger the energy consumption total is and also because spin up operations consume a significant amount of energy. The 5 second threshold worked well for the rest of our experiments and we decided to keep this number for the rest of our experiments. If the threshold is increased we could see smaller energy efficiency gains, but it is always the safer choice to increase the idle threshold although you may not see any energy efficiency gains.

7.3.2 Impact of the Number of Files Prefetched

The second parameter that we decided to vary is the number of files to prefetch and these results are presented in 7.4. Again we used previous models and simulations and were expecting the energy efficiency to increase as the number of files prefetched was increased. If you increase the number of files that are prefetched you also increase the probability that the data disks will be placed into the standby state. This is caused by the fact that the buffer disk will be more likely to hold a file for a data request if

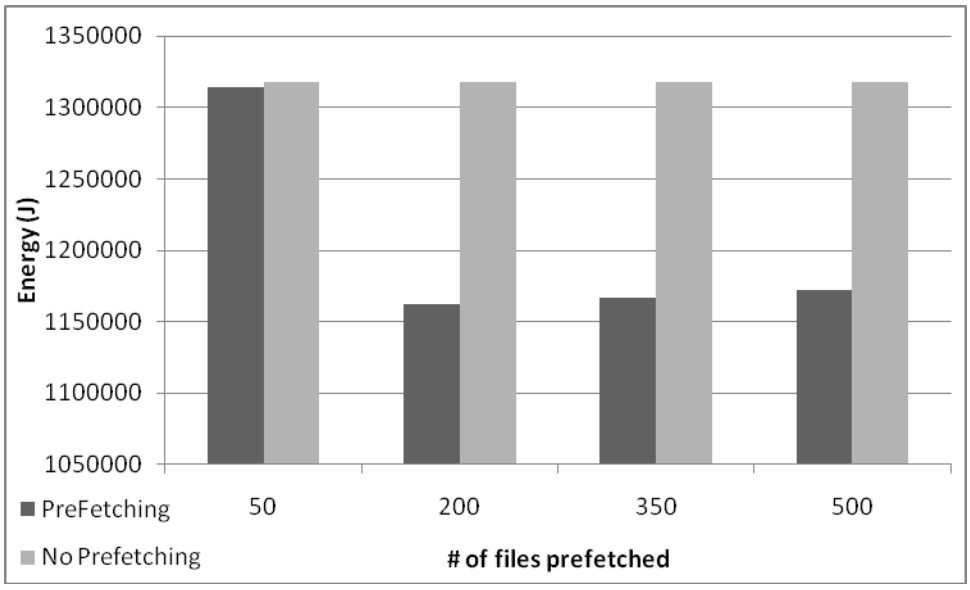


Figure 7.4: Energy Efficiency vs. # of Files Prefetched

it has a larger amount of files. Our experiments confirmed our hypothesis because as the number of prefetched files was increased the energy efficiency of our disk system was increased.

The 50 file prefetch size combined with our idle threshold of 5s caused the prefetching approach to consume slightly more energy. Once the prefetch size was increased beyond this number we actually saw a fairly consistent amount of energy that was consumed. This was caused because our buffer disk was able to cache a significant amount of files that were accessed in trace, causing the data disks to sleep for near the entire trace. There are slight variations among the larger prefetch sizes but this is caused by the variation in the traces that were generated.

7.3.3 Impact of the Inter-Arrival Delay

Figure 7.5 presents our simulation results collected from varying the inter-arrival delay of requests from 0ms to 1000ms. This test was performed to demonstrate how our system behaves as the workload of the system changes with 0ms representing a

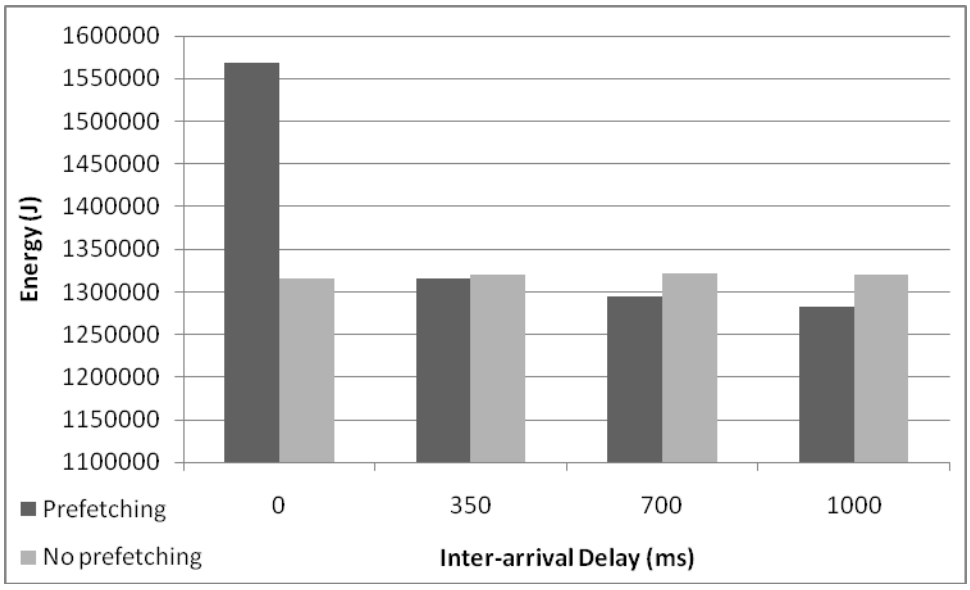


Figure 7.5: Energy Efficiency vs. Inter-Arrival Delay

heavy workload and 1000ms representing a light workload. Again using some results collected from models and simulation we were able to predict that as the workload was lightened the energy efficiency of our system would increase. This is caused because a light load produces more opportunities for transitioning to the standby state and also produces safer opportunities for transitioning to the standby state. For the purposes of our experiments a safe transition is one that allows the disk to stay in the standby state long after it is placed in the standby state. This is opposed to a transition to the standby state that is immediately followed by a data request which causes the disk to be immediately transitioned back to the active state. As predicted the energy efficiency of the disk system increases as the inter-arrival delay is increased. When the inter-arrival delay is 0 seconds the energy efficiency of the disk system is actually decreased. The non-energy aware approach has no opportunities to transition the disks into the standby state which causes the disks to be busy the entire trace. In the case of prefetching the data disks have short idle windows and our 5 second idle-window proves to be too small for the 0ms inter-arrival delay.

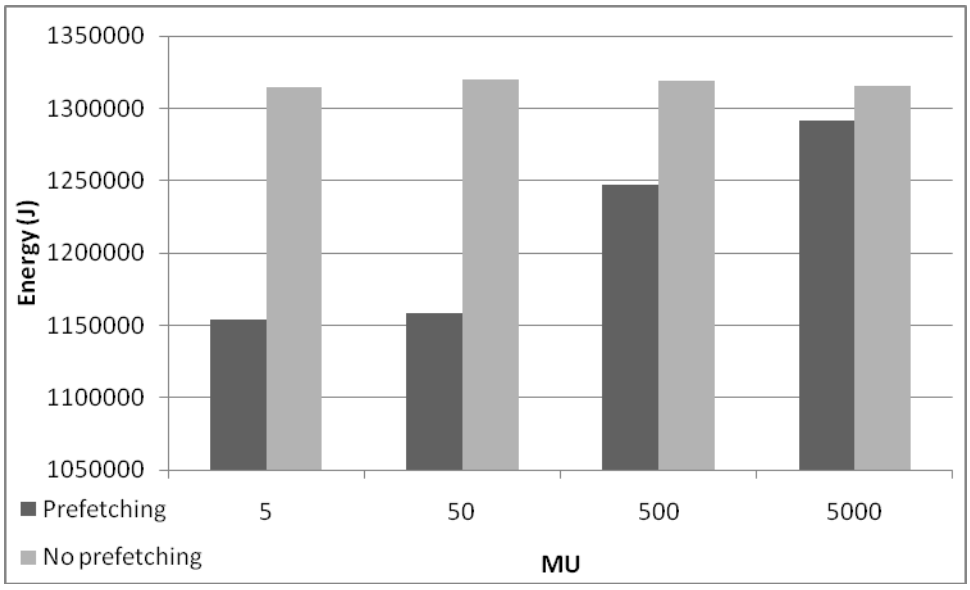


Figure 7.6: Energy Efficiency vs. MU

7.3.4 Impact of MU

Figure 7.6 varies the value MU, which is a parameter for the Poisson distribution used to generate the synthetic traces. Specifically MU is used to determine which files are accessed with a low value of MU skewing many of the requests in the trace towards a small group of files. If a small value of MU is used one expects the prefetching scheme to produce significant energy efficiency gains because it will cause most of the trace to be serviced from a buffer disk. Our experimental results confirm this trend with the MU value of 5 producing the largest energy efficiency gain and the MU value of 5000 producing the smallest energy efficiency gain.

7.3.5 Striping vs. Non-Striping Comparison

Parameter	Striping	No Striping
Energy Consumption (J)	2088113	2100243
Response Time (S)	2.78	13.87

Table 7.2: Results of Striping vs. No Striping

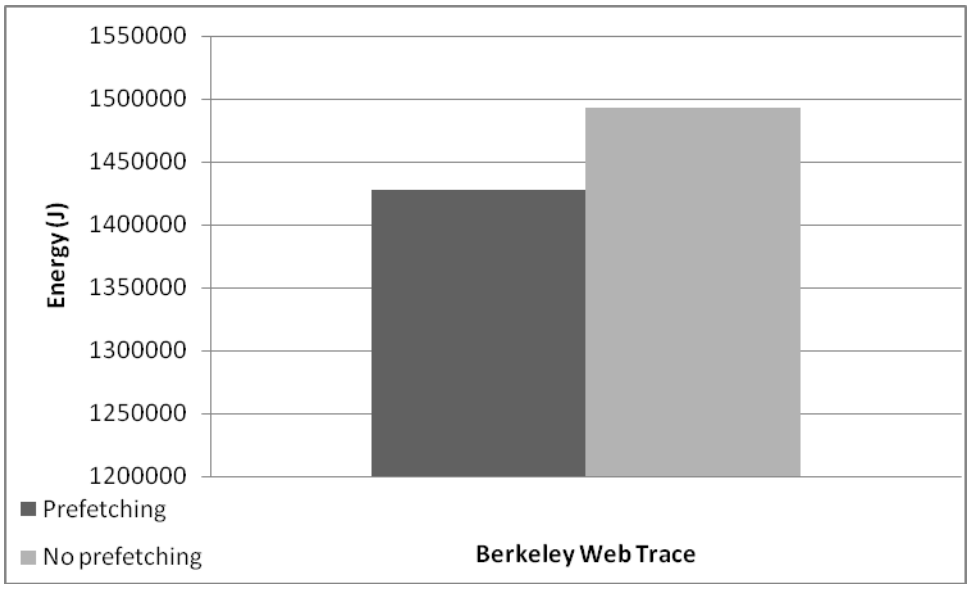


Figure 7.7: Berkely Web Trace Results

Table 7.2 shows that striping can significantly improve the response time of a request and also the energy efficiency of the disk system. For these experiments the file size was set at 250 MB, the inter-arrival delays was set at 700 ms, and the amount of files to prefetch was set to 70 out of 5000 files. As expected the striping groups significantly outperformed the non-striping approach. Using striping we are able to break a file into 5 separate chunks of data that are 50 MB as opposed to trying to read 250 MB from each node. In our experiments we had one storage server which would read the data simultaneously from the storage nodes. This could be improved by allowing multiple readers to connect to multiple writes to further improve the bandwidth utilization. The results are expected because striping has been extensively used to improve the performance of disk systems, we have added buffer disks to improve the energy efficiency and balance the performance and energy efficiency of the storage system.

7.3.6 Berkely Web Trace Results

To improve the validity of our experimental results we decided to conduct experiments based on the Berkeley Web Trace and the results are presented in 7.7. We extracted all of the read operations from the trace and realized that the workload was too light and that our energy-efficient strategy was guaranteed to produce the largest energy efficiency gains. To make the workload heavier we took the file access pattern from the Berkeley web trace and created two file accesses every second. This increased the workload of the trace to help demonstrate that our strategies are able to produce energy savings under heavier workloads. The first time we ran the trace we realized that our prefetching strategy was causing a decrease in the energy efficiency. There were two main factors causing this problem, our prefetch size was set to 20 files and the break-even time of 5 seconds was too aggressive. We decided to increase the prefetch size up to 50 files and also to change the break-even time to 10 seconds, which is a more conservative value. Once we re-ran our test we produced Figure 7.7, which shows that our strategy is able to produce a 4.4% energy efficiency increase.

7.3.7 Chapter Conclusion

This chapter leveraged EEVFS and implemented the idea of parallel striping groups for energy efficiency. This novel idea helps improve the energy efficiency of the storage system, while maintaining performance. Parallel striping groups were designed with the intent to match the performance of the nearest bottleneck focusing on the efficient use of resources. In addition the parallel striping groups leverage the concept of buffer data disks that have been extensively studied in this dissertation. This requires one to coordinate the placement and striping of files across many storage nodes and this is aided by the groundwork laid with EEVFS. Parallel striping groups are an extension to EEVFS that helps to improve the performance of EEVFS while maintaining energy savings. In this chapter we tested EEVFS with parallel striping

groups against striping with no energy efficiency and were able to produce energy savings with a minimal impact to the response times of the storage system.

For future work we need to extend EEVFS to be a production grade system. Currently we have a limited number of storage nodes and parallel striping groups need to be implemented in a large scale storage system to test EEVFS. This work needs to be extended by allowing multiple EEVFS storage nodes connect to multiple clients to move data as quickly as possible. The EEVFS storage server node is currently a bottleneck of the system and we need to update the architecture, so that the storage server node can connect storage clients directly to storage nodes.

Chapter 8

Conclusions and Future Work

In this dissertation, we have investigated techniques to conserve energy in large scale storage systems. This work was first developed using mathematical models, which were then used to develop simulators to test many parameters. Using our simulation results we determined that prefetching data can help conserve a significant amount of energy in disk systems. Based on this conclusion we investigated a means to implement our strategies in a real world system. To implement our ideas some of our algorithms had to be modified for the implementation phase of the work. Our implementation required a means of tracking file access patterns so we developed an Energy Efficient Virtual File System. After this work was completed we used our virtual file system to collect extensive experimental results. In the course of developing EEVFS we also developed a novel striping strategy that is conducive to increased energy efficiency and performance. This chapter is organized as follows, section 8.1 highlight the main contributions of the research presented in this dissertation. Section 8.2, concentrates on future research directions based on the work contained in this dissertation.

8.1 Main Contributions

This dissertation introduced four main contributions that aim to achieve energy efficient storage system while producing minimal response time degradations. The first contribution is the introduction of a model and simulation technique that allowed us to investigate how the data size, file access patterns, and disk parameters impact techniques to conserve energy. To conserve energy disks must be placed in the standby

state which can be costly both in term of time and energy and this penalty must be offset with a long period of sleep time.

After we conducted extensive simulation results using the models that we have developed we began investigating a method to improve the validity of our results. Modeling and simulation of disk drives is a complex task and can be aided with the use of an existing validated disk simulation, DiskSim. DiskSim is a useful piece of software, but it lacked power models and state transitions that were required for our energy saving techniques. To remedy this deficiency I wrote some extensions to the DiskSim simulator that were able to support our energy efficient strategies. This helped improve our simulation results and we decided to work on an implementation of our strategies.

The third main contribution is the use of an Energy Efficient Virtual File System (EEVFS), which manages the location of files and the states of multiple disks connected with a network interconnect. Using the modeling and simulation that was conducted we decided that large file systems and predictable access patters are conducive to energy savings. Prime examples that could benefit from our work include web servers that are primarily used for reads and access patterns can be determined. A service like YouTube fits this profile and many other web servers have the properties that can promote the use of our energy savings principles.

The fourth main contribution of this dissertation is the concept of parallel striping groups for energy efficiency. This work leverages the backbone of EEVFS, but adds striping to help improve the performance of EEVFS. Striping groups can be matched to performance bottlenecks which is physically energy efficient and the buffer disks used also help to lower the energy efficiency through the use of data placement.

8.2 Future Work

All of the techniques mentioned in this dissertation leverage prefetching to improve the energy efficiency of disk systems. The current work focuses on static prefetching of data and all of the techniques in this dissertation could be improved with dynamic prefetching techniques. This is a difficult problem due to the fact that prefetching is a costly operation that can negatively impact the energy efficiency and the response time of storage systems.

Another area to improve the work in this dissertation is the fact that our current implementation of EEVFS is a prototype and work must be completed to improve the prototype to be production grade. There are two main methods to achieve this goal, develop our prototype, or leverage an existing virtual file system, such as PVFS (Parallel Virtual File System). I believe using a well developed virtual file system is the only method available to a small research group, so I have begun investigating PVFS. During the preliminary investigation of PVFS it was realized that the open source property of PVFS would allow us to make changes to support the ideas presented in this dissertation.

Bibliography

[1] Moore B., *Taking the data center power and cooling challenge*, Energy User News, 2002.

[2] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli, *A survey of design techniques for system-level dynamic power management*, (2002), 231–248.

[3] John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger, and Contributors, *The disksim simulation environment version 4.0 reference manual*, Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-08-101, 2008.

[4] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini, *Conserving disk energy in network servers*, IN PROCEEDINGS OF THE 17TH INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, 2003, pp. 86–97.

[5] Feng Chen, Song Jiang, Weisong Shi, and Weikuan Yu, *Flexfetch: A history-aware scheme for i/o energy saving in mobile computing*, ICPP '07: Proceedings of the 2007 International Conference on Parallel Processing (Washington, DC, USA), IEEE Computer Society, 2007, p. 10.

[6] Dennis Colarelli and Dirk Grunwald, *Massive arrays of idle disks for storage archives*, Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing (Los Alamitos, CA, USA), IEEE Computer Society Press, 2002, pp. 1–11.

[7] Roselli D. and Anderson T. E., *Characteristic of file system workloads*, University of California Berkeley Technical Report UCB/CSD-98-1029, 1998.

[8] Jones E., *Epa announces new computer efficiency requirements*, EPA Anouncement, 2006.

[9] Hyeonsang Eom and Jeffrey K. Hollingsworth, *Speed vs. accuracy in simulation for i/o-intensive applications*, IPDPS, IEEE Computer Society Press, 2000, pp. 315–322.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The google file system*, SIGOPS Oper. Syst. Rev. **37** (2003), no. 5, 29–43.

[11] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, *Drpm: dynamic speed control for power management in server class disks*, Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on, June 2003, pp. 169–179.

[12] John Hawkins and Mikael Bodn, *The applicability of recurrent neural networks for biological sequence analysis*, IEEE/ACM Transactions on Computational Biology and Bioinformatics **2** (2005), 243–253.

[13] R. Hedges, B. Loewe, T. McLarty, and C. Morrone, *Parallel file system testing for the lunatic fringe: the care and feeding of restless i/o power users*, april 2005, pp. 3 – 17.

[14] Inki Hong and Miodrag Potkonjak, *Power optimization in disk-based real-time application specific systems*, Proc. IEEE/ACM Int'l Conf. Comp.-Aided Design (Washington, DC, USA), IEEE Computer Society, 1996, pp. 634–637.

[15] Maximum Institution Inc., *Power, heat, and sledgehammer*, White Paper, 2002.

[16] Young-Jin Kim, Kwon-Taek Kwon, and Jihong Kim, *Energy-efficient disk replacement and file placement techniques for mobile systems with hard disks*, SAC '07: Proceedings of the 2007 ACM symposium on Applied computing (New York, NY, USA), ACM, 2007, pp. 693–698.

[17] Thomas T. Kwan, Robert E. Mcgrath, and Daniel A. Reed, *Ncsa's world wide web server: Design and performance*, IEEE Computer **28** (1995), 68–74.

[18] Kester Li, Roger Kumpf, paul horton, and Thomas Anderson, *A quantitative analysis of disk drive power management in portable computers*, In Proceedings of the 1994 Winter USENIX Conference, 1994, pp. 279–291.

[19] W. B. Ligon and R. B. Ross, *Implementation and performance of a parallel file system for high performance distributed applications*, Proc. 5th IEEE Int'l Symp. High Performance Distributed Computing (Washington, DC, USA), IEEE Computer Society, 1996, p. 471.

[20] A. Manzanres, Xiaojun Ruan, Shu Yin, M. Nijim, Wei Luo, and Xiao Qin, *Energy-aware prefetching for parallel disk systems: Algorithms, models, and evaluation*, Proc. 8th IEEE Int'l Symp. Network Comp. and App., July 2009, pp. 90 –97.

[21] Edmund B. Nightingale and Jason Flinn, *Energy-efficiency and storage flexibility in the blue file system*, Proc. 6th USENIX Symp. Operating Sys. Design & Implementation (Berkeley, CA, USA), USENIX Association, 2004, pp. 25–25.

[22] George Panagiotakis, Michail D. Flouris, and Angelos Bilas, *Reducing disk i/o performance sensitivity for large numbers of sequential streams*, Distributed Computing Systems, International Conference on **0** (2009), 22–31.

[23] Athanasios E. Papathanasiou and Michael L. Scott, *Energy efficient prefetching and caching*, ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 2004, pp. 22–22.

[24] David A. Patterson, Garth Gibson, and Randy H. Katz, *A case for redundant arrays of inexpensive disks (raid)*, SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data (New York, NY, USA), ACM, 1988, pp. 109–116.

[25] Eduardo Pinheiro and Ricardo Bianchini, *Energy conservation techniques for disk array-based servers*, ICS '04: Proceedings of the 18th annual international conference on Supercomputing (New York, NY, USA), ACM, 2004, pp. 68–78.

[26] Eduardo Pinheiro, Ricardo Bianchini, and Cezary Dubnicki, *Exploiting redundancy to conserve energy in storage systems*, SIGMETRICS Perform. Eval. Rev. **34** (2006), no. 1, 15–26.

[27] Xiaojun Ruan, A. Manzanares, Shu Yin, Ziliang Zong, and Xiao Qin, *Performance evaluation of energy-efficient parallel i/o systems with write buffer disks*, Proc. 38th Int'l Conf. Parallel Processing, Sept. 2009, pp. 164 –171.

[28] Sriram Sankar, Yan Zhang, Sudhanva Gurumurthi, and Mircea R. Stan, *Sensitivity-based optimization of disk architecture*, IEEE Trans. Comput. **58** (2009), no. 1, 69–81.

[29] H. Shen, Mohan Kumar, S.K. Das, and Z. Wang, *Energy-efficient caching and prefetching with data consistency in mobile distributed systems*, Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, April 2004, pp. 67–.

[30] Seung Woo Son and Mahmut T. Kandemir, *Energy-aware data prefetching for multi-speed disks*, Conf. Computing Frontiers, 2006, pp. 105–114.

[31] S.W. Son, M. Kandemir, and A. Choudhary, *Software-directed disk power management for scientific applications*, Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, April 2005, pp. 4b–4b.

[32] D.B. Trizna, *Microwave and hf multi-frequency radars for dual-use coastal remote sensing applications*, OCEANS, 2005. Proceedings of MTS/IEEE, 2005, pp. 532–537 Vol. 1.

[33] Jun Wang, Xiaoyu Yao, and Huijun Zhu, *Exploiting in-memory and on-disk redundancy to conserve energy in storage systems*, IEEE Trans. Comput. **57** (2008), no. 6, 733–747.

[34] Jun Wang, Huijun Zhu, and Dong Li, *eraid: Conserving energy in conventional disk-based raid system*, Computers, IEEE Transactions on **57** (2008), no. 3, 359–374.

[35] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, and Ethan L. Miller, *Dynamic metadata management for petabyte-scale file systems*, Proc. ACM/IEEE Int'l Conf. Supercomputing (Los Alamitos, CA, USA), IEEE Computer Society, 2004, p. 4.

[36] Tao Xie, *Sea: A striping-based energy-aware strategy for data placement in raid-structured storage systems*, Computers, IEEE Transactions on **57** (2008), no. 6, 748–761.

[37] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang, *Modeling hard-disk power consumption*, FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies (Berkeley, CA, USA), USENIX Association, 2003, pp. 217–230.

[38] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes, *Hibernator: helping disk arrays sleep through the winter*, SIGOPS Oper. Syst. Rev. **39** (2005), no. 5, 177–190.

[39] Qingbo Zhu, Francis M. David, Christo F. Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao, *Reducing energy consumption of disk storage using power-aware cache management*, HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture (Washington, DC, USA), IEEE Computer Society, 2004, p. 118.

[40] Qingbo Zhu, Asim Shankar, and Yuanyuan Zhou, *Pb-lru: a self-tuning power aware storage cache replacement algorithm for conserving disk energy*, ICS '04: Proceedings of the 18th annual international conference on Supercomputing (New York, NY, USA), ACM, 2004, pp. 79–88.

[41] Xiaotong Zhuang and Santosh Pande, *Power-efficient prefetching via bit-differential offset assignment on embedded processors*, LCTES '04: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems (New York, NY, USA), ACM, 2004, pp. 67–77.

[42] Z. Zong, M. Briggs, N. O'Connor, and X. Qin, *An energy-efficient framework for large-scale parallel storage systems*, Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, March 2007, pp. 1–7.

Appendices

Appendix A

DiskSim Source Code Modifications

```
// This code is was written by Adam Manzanares, Auburn University based on the code from Sriam Sankar

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include "disksim_global.h"
#include "disksim_iosim.h"
#include "disksim_power.h"
#include "disksim_stat.h"
#include "disksim_disk.h"


#include "inst.h"

void init_disk_power_model()
{

  int i;
        disk *currdisk;

  int numdisks=disksim->diskinfo->numdisks;

  for(i=0;i<numdisks;i++)
  {
        currdisk = disksim->diskinfo->disks[i];
    currdisk->stat.activeEnergy=0.0;
    currdisk->stat.activeTime=0.0;
        currdisk->stat.idleEnergy=0.0;
    currdisk->stat.idleTime=0.0;
    currdisk->stat.sleepTime=0.0;
    currdisk->stat.sleepEnergy=0.0;
    currdisk->stat.transEnergy=0.0;
    currdisk->idle_time=0.0;
    currdisk->last_access=0.0;
    currdisk->stat.transTime=0.0;
    currdisk->status=DISK_IDLE;
  }
}


void activeEnergyStat(disk *currdisk,double timeVal)
{
  currdisk->stat.activeEnergy+=(timeVal/1000.0)*(currdisk->model->dm_power_active);
  currdisk->stat.activeTime+=(timeVal/1000.0);

}

// Mark beginning and end of each idle period
void idleEnergyStat(disk *currdisk, double timeVal, double idleStartTime)
{
      currdisk->stat.idleTime+= (timeVal/1000.0);
      currdisk->stat.idleStartTime=idleStartTime;
}

// Idle Timer Check, Add another checkpoint and update all disks idle time
// Currently checked every 500ms
void timer_expired(event* curr)
{


  if(disksim->stop_dpm_check==FALSE)
  {

    timer_event* timecheck;

        timecheck=calloc(1,sizeof(timer_event));
        timecheck->time=curr->time+500;
        timecheck->type=TIMER_EXPIRED;

        addtointq((event *) timecheck);

        disk_update_idle(curr->time);
  }
```

```
}

void disk_update_idle(double time)
{
  int i;
    disk *currdisk;

  int numdisks=disksim->diskinfo->numdisks;

  for(i=0;i<numdisks;i++)
  {
        currdisk = disksim->diskinfo->disks[i];
    currdisk->idle_time=time-currdisk->last_access;


    switch(currdisk->status)
    {

      case DISK_IDLE:
      {
      if(currdisk->idle_time>5000) // Idle DPM Threshold
      {
        event* spin_down;
        spin_down=calloc(1,sizeof(event));
        spin_down->time=time+currdisk->model->dm_spin_down_time;
        spin_down->type=SPIN_DOWN_COMP;
        spin_down->temp=i;

        addtointq(spin_down);

        currdisk->status =DISK_SPIN_DOWN;
        currdisk->spin_down_complete=spin_down->time;
        currdisk->stat.transEnergy+=currdisk->model->dm_spin_down_penalty;
        currdisk->stat.transTime+=currdisk->model->dm_spin_down_time;
        currdisk->stat.spin_down_transitions++;
      }

      break;
      }
      case DISK_SPIN_DOWN:
      {
      // Spinning down do nothing
      break;
      }
      case DISK_SPIN_UP:
      {
      // Spinning up do nothing
      break;
      }
      case DISK_STANDBY:
      {
      // Do Nothing
      break;
      }
      default:
      {

      printf("Unknown disk state reached in disk_update_idle\n");
      break;
      }

    }
  }

}

double wakeup_disk_sleep(double time,disk* currdisk)
{
  event* spin_up;
  disk* id;
  int disk_id=0;

  spin_up=calloc(1,sizeof(event));
  spin_up->time=time+currdisk->model->dm_spin_up_time;
  spin_up->type=SPIN_UP_COMP;


  while(currdisk!=disksim->diskinfo->disks[disk_id])
  {
    disk_id++;
  }

  spin_up->temp=disk_id;

  addtointq(spin_up);

  currdisk->status =DISK_SPIN_UP;
  currdisk->spin_up_complete=spin_up->time;
  currdisk->stat.sleepTime+=time-currdisk->spin_down_complete;
  currdisk->stat.transEnergy+=currdisk->model->dm_spin_up_penalty;
  currdisk->stat.spin_up_transitions++;
  currdisk->stat.transTime+=currdisk->model->dm_spin_up_time;
```

```
  return currdisk->model->dm_spin_up_time;
}

double wakeup_disk_spndwn(double time,disk* currdisk)
{
  event* spin_up;
  disk* id;
  int disk_id=0;

  spin_up=calloc(1,sizeof(event));
  spin_up->time=currdisk->spin_down_complete+currdisk->model->dm_spin_up_time;
  spin_up->type=SPIN_UP_COMP;


  while(currdisk!=disksim->diskinfo->disks[disk_id])
  {
    disk_id++;
  }

  spin_up->temp=disk_id;

  addtointq(spin_up);

  currdisk->status =DISK_SPIN_UP;
  currdisk->spin_up_complete=spin_up->time;
  currdisk->stat.transEnergy+=currdisk->model->dm_spin_up_penalty;
  currdisk->stat.spin_up_transitions++;
  currdisk->stat.transTime+=currdisk->model->dm_spin_up_time;

  return spin_up->time-time;
}

double spin_up_delay(double time,disk* currdisk)
{
  return currdisk->spin_up_complete-time;
}
```