2010 39th International Conference on Parallel Processing

Energy Efficient Prefetching with Buffer Disks for Cluster File Systems

Adam Manzanares, Xiaojun Ruan, Shu Yin, Jiong Xie,
Zhiyang Ding, Yun Tian, James Majors, and Xiao Qin

Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849-5347
Email: {acm0008, xzr0001, szy0004, jzx0009, dingzhi, tianyun, majorjh, xqin}@auburn.edu,
http://www.eng.auburn.edu/∼xqin

***Abstract*—Energy efficient computing is becoming increasingly important as the scale of parallel computing systems is expanding. As the processing power of parallel computing systems has been incremented there has been an increased demand for large scale storage systems to store the output of these parallel computing systems. Data centers are growing at an enormous pace and it is important to investigate a means of managing the energy-efficiency of large scale parallel storage systems. To address these issues we introduce EEVFS (Energy Efficient Virtual File System), which is able to manage data placement and disk states to help improve the energy efficiency of a parallel disk system. EEVFS places data on the storage disks in an energy efficient layout and attempts to predict when each disk will be idle for a large period of time, facilitating a state transition into the standby state. EEVFS should also maintain relatively high performance, so we have built a load balancing policy into the data partitioning of EEVFS. The implementation architecture and measured results are presented to demonstrate the energy-efficiency and performance characteristics of EEVFS.**

I. 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 [1] [2]. 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 [3]. 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 [4][5], power-aware cache management strategies [6], power-aware prefetching schemes [7], software-directed power management techniques [8], redundancy techniques [9] and multi-speed settings [10][11]. 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 [12][13]. 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 [12] 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.

0190-3918/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPP.2010.48

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 III-A.

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 paper is organized as follows: Section II gives an overview of related work in the area energy-efficient storage systems, and then presents the motivation of this research. We discuss in Section III the design issues of EEVFS. Section IV discusses the implementation decisions of EEVFS. Before discussing the performance evaluation, we present in Section V a testbed, metrics, and important parameters. Then, Section VI shows experimental results. Finally, Section VII concludes the paper and presents our future research directions.

II. Related Work and Observations

A. Strengths/Limitations of Related Work

Almost all energy efficient strategies rely on the dynamic power management techniques or DPM [14]. The DPM techniques assume a disk has 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 generating larger idle times for individual disks which include prefetching or caching a subset of the data or by placing data strategically on disks [15] [16].

Memory cache techniques. Energy-aware prefetching can effectively improve the energy efficiency of disk systems, although many existing techniques have focused on low power disks. For example, energy-efficient prefetching was explored by Papathanasiou and Scott [16]. Their techniques relied on changing prefetching and caching strategies within the Linux kernel. Zhu *et al.* developed PB-LRU - another energy efficient cache management strategy [17]. The PB-LRU strategy focused on providing more opportunities for underlying disk power strategies to save energy. Flash drives have been proposed for use as buffers for disk systems (see, for example, [18]), because flash drives are small, light-weight, energy efficient, noiseless, and shock resistant. Shen et al. have studied the issues of energy-efficient caching and prefetching in the context of mobile distributed systems [19]. The aforementioned research projects were focused 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 much smaller than disk capacities. We propose strategies that use a disk - rather than main memory or flash drives - as a cache to prefetch data in a parallel file system. The break-even times of disk drives are usually very high and prefetch data accuracy and size become a critical factor in energy conservation in this study.

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 in hard drives. One successful approach to overcoming large break-even times is to use multi-speed disks [7]. Another way to reduce energy dissipation in storage systems is to replace high-performance disks with new energy-efficient disks [20]. Mobile computing systems have been recognized as platforms where disk energy should be conserved [21]. The mobile computing platforms commonly use low power disks with small break-even times. The weakness of using multi-speed disks is that there are few commercial multi-speed disks currently available on the market. Low power disk systems are an ideal candidate for energy savings, but they may not always be a feasible alternative. The goal of this study is to develop an energy-efficient file system for existing disk arrays without requiring any changes in the storage system hardware.

Disk as cache. The MAID (Massive Arrays of Idle Disks) system was proposed by Colarelli and Grunwald to use a subset of disk drives as cache for a larger disk system [4]. MAID was designed to conserve energy of mass storage systems with the performance goal of matching tape-drive systems. The main difference between our EEVFS and MAID is two-fold. First, the caching policies implemented in EEVFS are significantly different from that in MAID (see Section IV-A for details on the caching policies). For example, MAID caches blocks that are stored in a LRU order. Our strategy attempts to analyze requests look-ahead window and prefetch any file blocks that will be capable of reducing the total energy consumption of the disk system. Second, the energy-efficient prefetching mechanism is implemented at the file-system level EEVFS; the prefetching mechanism in MAID is implemented at the storage-system level.

Pinheiro and Bianchini developed the PDC technique to migrate sets of data to different disk locations [15]. The goal of PDC 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. PDC is a migratory

strategy that can cause large energy overheads when a large amount of data must be moved within a disk system. PDC also requires the overhead of managing metadata for all of the blocks in the disk system, whereas our strategy implemented in EEVFS only needs to manage metadata for the file blocks in buffer disks.

Parallel file systems Large scale parallel storage systems are an important research topic as I/O has typically been one of the neglected areas of cluster systems. Lustre is a popular production file system that is widely used to support high-performance computing systems [22]. PVFS - a virtual file system that is used for large scale clusters - is also widely used [23]. These two large-scale file systems are both performance oriented whereas our goal is to develop an energy-efficient cluster file system. The Blue File System, or BlueFS, developed at the University of Michigan is an energy efficient distributed file system that is targeted at mobile devices [24]. BlueFS can significantly reduce energy usage of mobile devices. Unlike BlueFS, EEVFS aims at improving energy efficiency of cluster storage systems.

B. Observations

With the previously mentioned limitations of energy efficient research and cluster file systems we propose an energy-efficient file system in which the centerpiece is a novel prefetching strategy that can improve the energy efficiency of cluster storage systems. Our research differs from the previous research on the following key points.

- We developed an Energy Efficient Virtual file system (EEVFS), which is capable of producing energy efficiency gains in large scale storage systems. EEVFS is a distributed virtual file system that manages data placement and disk states to conserve energy.
- We develop a prefetching mechanism - a centerpiece of EEVFS - that tries to move popular data into a set of buffer disks without affecting the data layout of any of the data disks.
- We developed a prototype implementation of our strategy and provide implementation details and a response time analysis. In addition, we tested our implementation using the web file access pattern from the Berkeley File System Workloads technical report [25].
- The prefetching mechanism in EEVFS has the added benefit of not requiring any changes to be made to the overall architecture of an existing cluster storage systems. Previous work has focused on redesigning a data storage system or replacing existing disks to produce energy savings. In EEVFS, one can either add extra disks or use the current disks to produce energy savings for cluster storage systems under certain conditions.

III. 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.

A. 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.

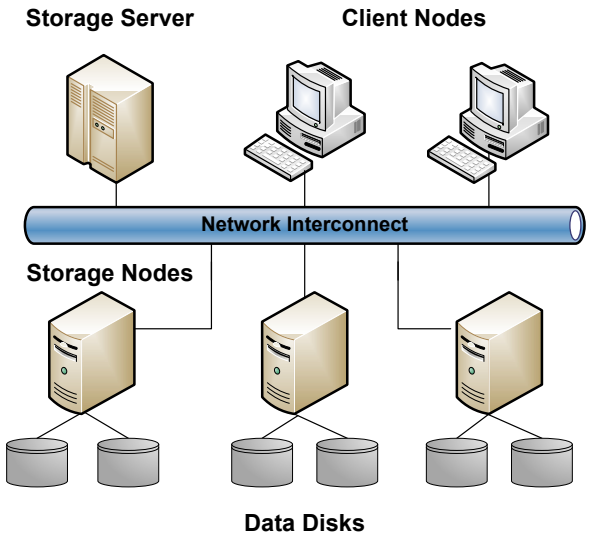


Fig. 1: Architecture of EEVFS. The storage server manages metadata (e.g., data location and file size).Each storage node manages multiple disks, which are separated into two groups: buffer disks and data disks.Client nodes can directly access storage servers through the network interconnect.

Fig. 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 III-B below) and (2) metadata for the multiple local disks (see Section IV-D).

B. 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.

C. 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.

IV. 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.

A. 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 IV-B. 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 IV-C.

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.

B. 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 server 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.

C. 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

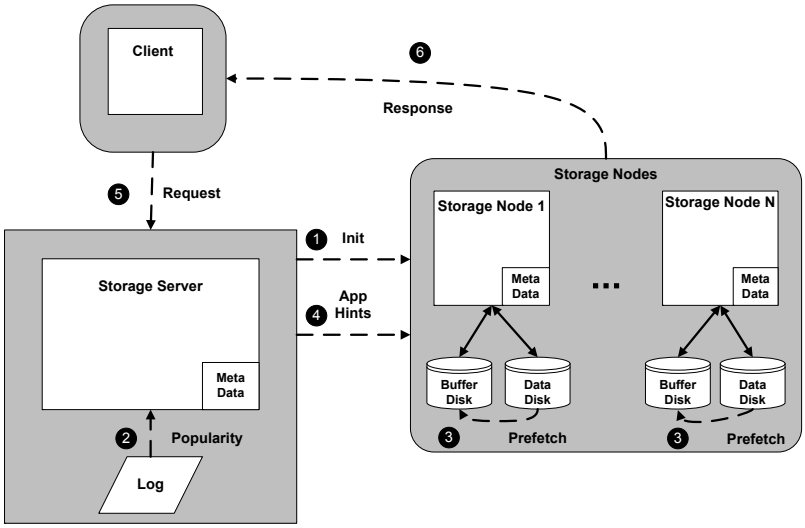


Fig. 2: EEVFS Process Flow Chart. Step 1: initialization phase; step 2: storage server generates file popularity; Step 3: prefetch popular files from data disks to a buffer disk; Step 4: applications provide access hints; Step 5: applications submit file requests; Step 6: storage nodes return data to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

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 their are any opportunities to save energy and if their 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 their 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.

D. 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 [26]. We plan to integrate their dynamic metadata management scheme in our energy-efficient cluster storage system.

V. Evaluation Methodology

A. 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 I for details on the configuration of the testbed). In our storage cluster system, there is one server node and eight storage nodes. The server node has a 2.0 GHz Pentium 4 processor, 2 Gbytes of RAM, a 1 Gbits/sec Intel EtherExpress Pro Fast-Ethernet network card, and a 120 GB SATA disk. There are two types of storage nodes in the cluster system. Each Type 1 storage node has a 3.2 GHz Pentium 4 processor, 1 Gbytes of RAM, a 1 Gbits/sec Fast-Ethernet network card, and a 80 GB ATA/133 disk. Each Type 2 storage node contains a 2.4 GHz Pentium 4 processor, 512 Gbytes of RAM, a 100 Mbits/sec Fast-Ethernet network card, and an 80 GB ATA/133 disk. 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.

B. System and Workload Parameters

For the collection of our experimental results we have focused on varying five key system and workload parameters (see Table II) 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 II.

Parameter	Storage Server Node	Storage Node Type 1	Storage Node Type 2
CPU Type and Clock Speed	P4 2.0 GHz	P4 3.2 GHz	P4 2.4 GHz
Memory (MB)	2000	1000	512
Network Interconnect (Mb/s)	1000	1000	100
Disk Type	SATA	ATA/133	ATA/133
Disk Capacity	120 Gbyes	80 Gbyes	80 Gbyes
Disk Bandwidth	100 Mbytes/sec	58 Mbytes/sec	34 Mbytes/sec

TABLE I: Configuration of the Testbed - a Cluster Storage System with Multiple Types of Storage Nodes.

- **Data Size.** We conducted extensive experiments using both real-world traces (see Section VI-D) and synthetic file traces (see Sections VI-A - VI-C). 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 II: System and Workload Parameters.

C. Metrics

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

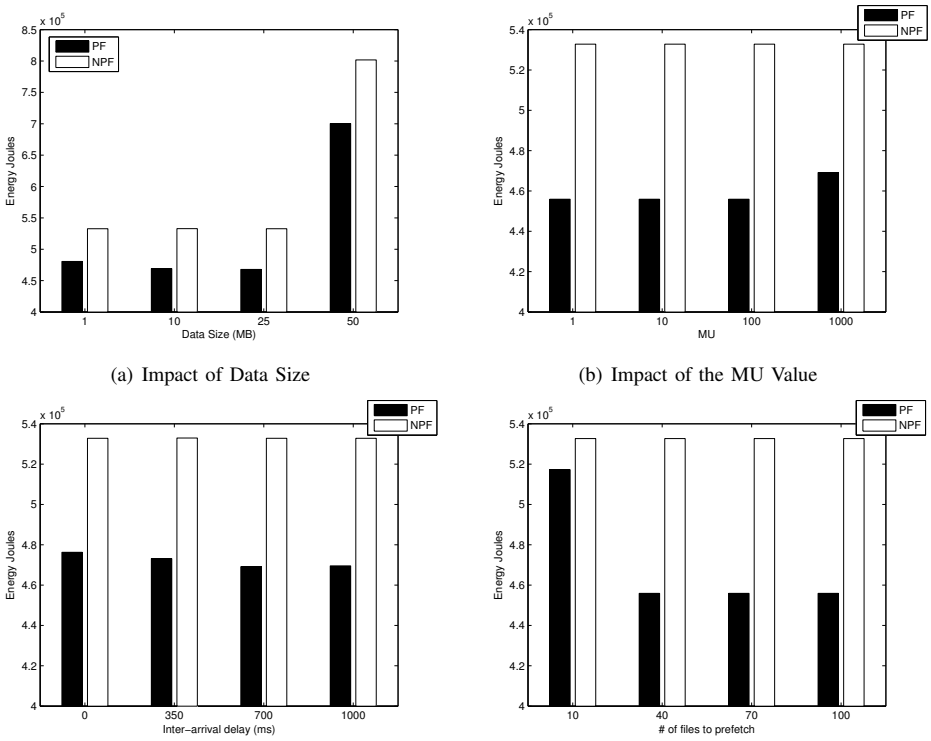
- **Energy Savings (see Section VI-A).** We compared the energy consumption of the cluster storage system with the energy-efficient prefetching mechanism against that of the same system without employing the prefetching mechanism.
- **Number of Power State Transitions (see Section VI-B).** The total 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 VI-C).** 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.

VI. Experimental Results

A. Energy Savings

Fig. 3 plots energy consumption of the tested cluster storage system as a function of the four workload and system parameters. From Fig. 3 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 Fig. 3(a), 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 a 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 700 ms.

Fig. 3(b) 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



(a) Impact of Data Size

(b) Impact of the MU Value

(c) Impact of Inter-Arrival Delay

(d) Impact of the Number of Files to Prefetch

Fig. 3: Energy Consumption of the tested cluster storage system as a function of (a) data size, (b) popularity rate, i.e., MU, (c) inter-arrival dealy, and (d) number of files to prefetch. PF - with prefetching; NPF - without prefetching

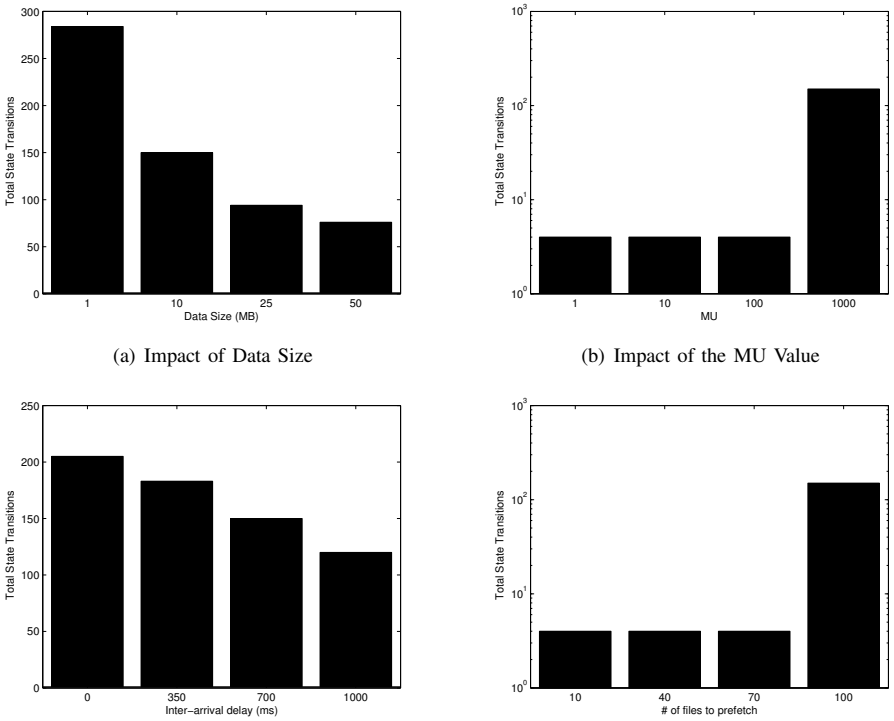
with larger values of MU. Using our default prefetch size value of 70 files we are able to produce the same amount of energy savings when MU is 100 or smaller. The reason for this similarity in energy consumption is the fact that when MU is 100 or smaller EEVFS is able to prefetch all of the required 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.

Fig. 3(c) reveals the impact of the inter-arrival delay on energy efficiency. The results plotted in Fig. 3(c) 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 1000 ms 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.

Fig. 3(d) 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 increases 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.

B. Power State Transitions

Fig. 4 displays the total number of state transitions for each test, of which the results were plotted in Fig. 3. 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.



(a) Impact of Data Size

(b) Impact of the MU Value

(c) Impact of Inter-Arrival Delay

(d) Impact of the Number of Files to Prefetch

Fig. 4: Number of power state transitions as a function of (a) data size, (b) popularity rate, i.e., MU, (c) inter-arrival dealy, and (d) number of files to prefetch.

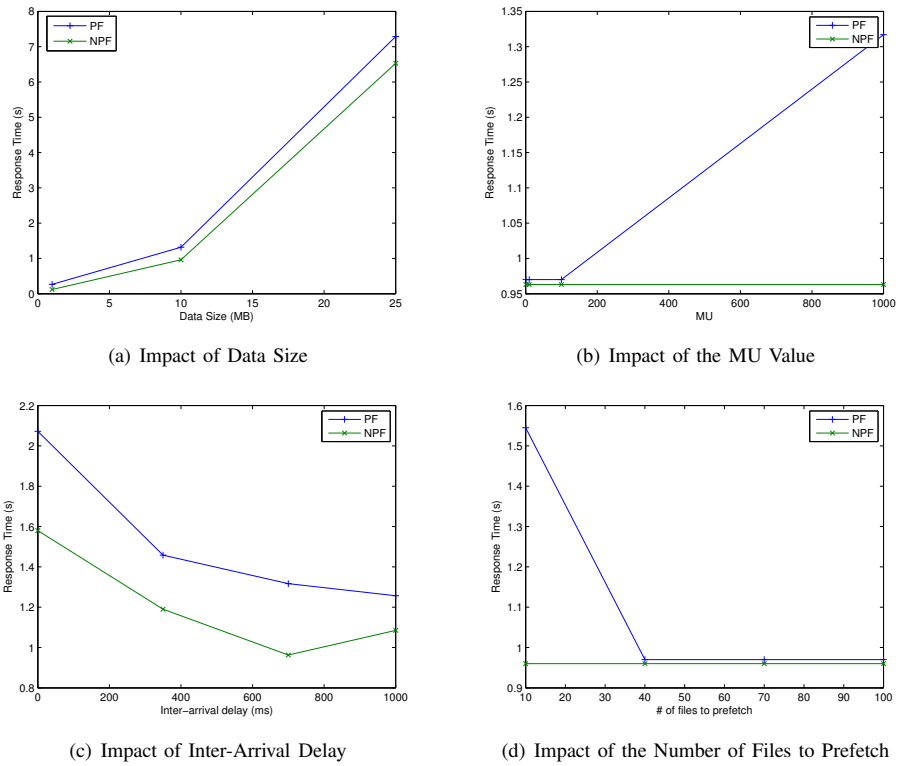
It is interesting to note that no state transitions will produce no energy savings, so there is a balance between energy efficiency and the number of state transitions. We aim to minimize the number of disk spin ups while being able to place the disks in the standby state for optimal energy savings. 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. The MU and number of files prefetched results 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 that when the number of prefetch files is 10 it 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.

C. Response Times

Now we analyze the performance penalties caused by the prefetching mechanism. Fig. 5 illustrates 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.

Figs. 5(a) and 5(c) 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



(a) Impact of Data Size

(b) Impact of the MU Value

(c) Impact of Inter-Arrival Delay

(d) Impact of the Number of Files to Prefetch

Fig. 5: File request response time as a function of (a) data size, (b) popularity rate, i.e., MU, (c) inter-arrival dealy, and (d) number of files to prefetch. PF - with prefetching; NPF - without prefetching

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 Fig. 5 with Fig. 4. 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.

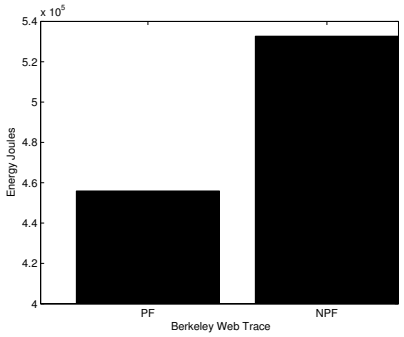


Fig. 6: Energy Consumption of the tested cluster storage system when the Berkeley Web Trace are considered.

D. Berkeley Web Trace Energy Consumption

The final figure, 6, we have produced presents a trace that was taken from the Berkeley file-system trace collection project [25]. 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.

VII. 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.

Acknowledgment

The work reported in this paper was supported by the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0917137 (CSR), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS).

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.

[2] H. Eom and J. K. Hollingsworth, "Speed vs. accuracy in simulation for i/o-intensive applications," in *Proc. Int'l Symp. Parallel and Distributed Processing*. IEEE Computer Society Press, 2000, pp. 315–322.

[3] M. I. Inc., "Power, heat, and sledgehammer," White Paper, April 2002.

[4] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives," in *Proc. 2002 ACM/IEEE Conf. Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–11.

[5] K. Li, R. Kumpf, P. Horton, and T. Anderson, "A quantitative analysis of disk drive power management in portable computers," in *Proc. USENIX Winter 1994 Technical Conf.* Berkeley, CA, USA: USENIX Association, 1994, pp. 22–22.

[6] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing energy consumption of disk storage using power-aware cache management," in *Proc. 10th Int'l Symp. High Perf. Comp. Arch.* Washington, DC, USA: IEEE Computer Society, 2004, p. 118.

[7] S. W. Son and M. Kandemir, "Energy-aware data prefetching for multi-speed disks," in *Proc. 3rd Conf. Computing Frontiers*. New York, NY, USA: ACM, 2006, pp. 105–114.

[8] S. W. Son, M. Kandemir, and A. Choudhary, "Software-directed disk power management for scientific applications," in *Proc. 19th IEEE Int'l Symp. Parallel and Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 4.2.

[9] E. Pinheiro, R. Bianchini, and C. Dubnicki, "Exploiting redundancy to conserve energy in storage systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 15–26, 2006.

[10] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "Drpm: dynamic speed control for power management in server class disks," *Comp. Arch. News*, vol. 31, no. 2, pp. 169–181, 2003.

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

[12] A. Manzanres, X. Ruan, S. Yin, M. Nijim, W. Luo, and X. Qin, "Energy-aware prefetching for parallel disk systems: Algorithms, models, and evaluation," in *Proc. 8th IEEE Int'l Symp. Network Comp. and App.*, July 2009, pp. 90 –97.

[13] X. Ruan, A. Manzanares, S. Yin, Z. Zong, and X. Qin, "Performance evaluation of energy-efficient parallel i/o systems with write buffer disks," in *Proc. 38th Int'l Conf. Parallel Processing*, Sept. 2009, pp. 164 –171.

[14] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *Readings in hardware/software co-design*, pp. 231–248, 2002.

[15] E. Pinheiro and R. Bianchini, "Energy conservation techniques for disk array-based servers," in *Proc. 18th Int'l Conf. Supercomputing*. New York, NY, USA: ACM, 2004, pp. 68–78.

[16] A. E. Papathanasiou and M. L. Scott, "Energy efficient prefetching and caching," in *Proc. USENIX Annual Technical Conf.* Berkeley, CA, USA: USENIX Association, 2004, pp. 22–22.

[17] Q. Zhu, A. Shankar, and Y. Zhou, "Pb-lru: a self-tuning power aware storage cache replacement algorithm for conserving disk energy," in *Proc. 18th Int'l Conf. Supercomputing*. New York, NY, USA: ACM, 2004, pp. 79–88.

[18] F. Chen, S. Jiang, W. Shi, and W. Yu, "Flexfetch: A history-aware scheme for i/o energy saving in mobile computing," in *Proc. Int'l Conf. Para. Processing*. Washington, DC, USA: IEEE Computer Society, 2007, p. 10.

[19] H. Shen, M. Kumar, S. Das, and Z. Wang, "Energy-efficient caching and prefetching with data consistency in mobile distributed systems," in *Proc. Int'l Symp. Parallel and Distributed Processing*, April 2004, pp. 67–.

[20] M. Song, "Energy-aware data prefetching for multi-speed disks in video servers," in *Proc. 15th Int'l Conf. Multimedia*. New York, NY, USA: ACM, 2007, pp. 755–758.

[21] Y.-J. Kim, K.-T. Kwon, and J. Kim, "Energy-efficient disk replacement and file placement techniques for mobile systems with hard disks," in *Proc. ACM Symp. Applied computing*. New York, NY, USA: ACM, 2007, pp. 693–698.

[22] 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.

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

[24] E. B. Nightingale and J. Flinn, "Energy-efficiency and storage flexibility in the blue file system," in *Proc. 6th USENIX Symp. Operating Sys. Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 25–25.

[25] R. D. and A. T. E., "Characteristic of file system workloads," University of California Berkeley Technical Report UCB/CSD-98-1029, 1998.

[26] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in *Proc. ACM/IEEE Int'l Conf. Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, p. 4.