Improving Quality-of-Service of File Migration Policies in High-Performance Servers

Nathaniel Pettis and Yung-Hsiang Lu
School of Electrical and Computer Engineering
Purdue University, West Lafayette, Indiana, USA
{npettis,yunglu}@purdue.edu

Abstract

Previous work demonstrates that migrating files between disks may induce idleness in servers and facilitate power management. However, these studies focus on steady-state power savings and do not consider the energy consumption and performance overhead of migrating files between disks. In this study, we construct a constrained optimization problem of file location for high-bandwidth applications by considering file migration energy, as well as steady-state energy consumption. We present an algorithm that improves quality-of-service by 63% for a streaming video server compared with existing techniques while maintaining comparable energy savings.

1. Introduction

Power management has an extensive history in mobile devices. However, power management is becoming increasingly important in high-performance servers because performance improvements and server density are limited by the excessive heat in the system [1]. Storage devices are large power consumers in modern high-performance servers. For example, a recent TPC benchmark for the Dell PowerEdge 2900 server [2] consumes approximately 3.8 kW of power, with 25% consumed by the server itself [3] and 75% consumed by the attached storage devices [4]. For large data centers, the power costs may be even more significant.

File system power management policies [5], [6] migrate files between physical devices to induce idleness that may be exploited to save energy. File system policies are useful in high-performance servers because insufficient idleness exists to achieve significant energy savings. Figure 1 illustrates how proper file location facilitates power management. Figure 1(a) depicts the bandwidth requirement, B_i, for each disk. W is the sum of the disks' bandwidth requirements. The files on Disk 3 to Disk n may be grouped with the files on Disk 1 and Disk 2, respectively. Figure 1(b) shows the bandwidth for each disk after file migration. The total bandwidth remains the same, but file accesses are concentrated on the first two disks, allowing power management on the remaining disks. Recent studies on server power management have utilized variable speed disks because traditional shutdown-based techniques are able to save energy only for low intensity workloads [5]. Dynamic revolutions per minute disks (called DRPM disks) allow energy to be saved during busy workloads, provided that lower bandwidth and higher latency may be tolerated [7].

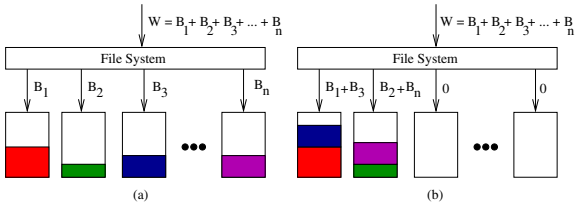


Figure 1. File system policies induce idleness by altering the access patterns for each disk. Bandwidth B_i is required for disk i. W is total bandwidth. (a) Original bandwidth for each disk. (b) Bandwidth allocation after file migration.

Existing file system policies [5], [6] focus on steady-state power savings for specified performance constraints. However, these studies do not consider the significant energy consumption and bandwidth overhead for file migration. For example, our experiments indicate some file system policies require an average of over 41 minutes during an hour interval to migrate files. In this paper, we present a policy that considers both the steady-state energy savings and the file migration energy for high-bandwidth applications. Our policy, called Genetic File System (GFS), uses a

978-1-4244-1890-9/07/$25.00 ©2007 IEEE

genetic algorithm to simultaneously minimize steady-state energy consumption and migration energy while mitigating the problem's complexity. We observe a 63% improvement in quality-of-service compared with existing policies for our streaming video server while maintaining comparable energy savings.

2. Related Work

File system policies save energy by changing the physical locations of files. Colarelli et al. [8] suggest using a cache disk for a massive array of idle disks (MAID) to induce idleness. Pinheiro et al. [5] suggest a method called popular data concentration (PDC) that sorts files according to the number of accesses and moves the most frequently accessed files to the same disk to induce idleness in the other disks. Their work also indicates that PDC achieves significantly more energy savings than MAID. Zhu et al. [6] present a system called Hibernator that randomly shuffles data blocks among disks of a similar speed in a RAID-5 storage network to allow DRPM and provide latency guarantees. In this paper, we build upon the work of Zhu et al. by extending their derivations to include migration energy and reformulating the problem to manage bandwidth-constrained applications rather than low latency applications. Furthermore, our policy simultaneously optimizes the steady-state power consumption and transient migration energy.

Several papers have focused on power management for high performance servers. Ranganathan et al. [9] propose enclosure-level power management for blade servers. However, they focus mainly on managing processor power. Papathanasiou et al. [10] employ aggressive prefetching to induce additional disk idleness. Their prefetching utilizes disk bandwidth more efficiently, increasing the usable bandwidth for file system policies. Cai et al. [11] and Zhu et al. [12] provide caching algorithms that increase idleness for server disks. These studies complement our file system policy by decreasing disk utilization.

Determining the file layout that minimizes energy consumption is a direct application of the multiple constrained knapsack problem, which is known to be NP-complete [13]. Several NP-complete problems, such as the traveling salesperson problem [14] and multiple knapsack problem [15], have been solved using genetic algorithms to avoid exhaustive searches. GFS uses a genetic algorithm to mitigate the complexity of solving the multiple constrained knapsack problem exactly.

The contributions of this paper include (a) an analytical formulation of file system policies that considers migration power in high-bandwidth applications, (b) a policy called GFS that reduces performance overhead by jointly reducing steady-state power consumption and migration power, and (c) an evaluation of GFS for a streaming video server to demonstrate the applicability to high-bandwidth applications.

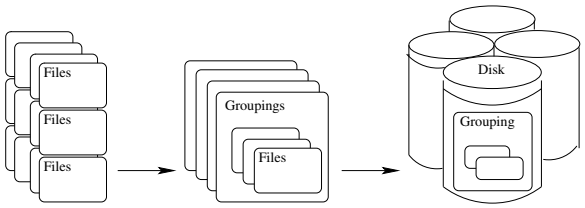


Figure 2. Steps involved in file system policies. Files are organized into groupings. Each group is assigned to a disk in a one-to-one mapping.

Symbol	Description	Units
$\bar{P}$	Total average power	W
$P_{ij,a}$	Active power for group i at speed j	W
$t_{ij,a}$	Active time for group i at speed j	s
$P_{ij,r}$	Idle power for group i at speed j	W
$t_{ij,m}$	Migration time for group i at speed j	s
$E_{i,c}$	Speed change energy for disk i	J
$t_{i,c}$	Speed change delay for disk i	s
T	Evaluation interval	s
b_k	Bandwidth required for file k	B/s
B_{ij}	Max bandwidth for disk i at speed j	B/s
S_i	Max capacity of disk i	B
s_k	Size of file k	B
X_{ik}	Indicator for file k in group i	—
M_k	Indicator for file k migrating	—
n_d	Number of disks	—
n_f	Number of files	—
Y_{id}	Assignment of group i to disk d	—

Table 1. Symbols used in formulation.

3. Problem Formulation

Our goal is to minimize energy consumption while meeting specified bandwidth constraints. File system policies consist of two steps: grouping and assignment. Figure 2 illustrates this process. Grouping collects all the files on the file system into a logical set that should be stored on the same disk. For example, files may be collected into groups of descending popularity. Grouping determines the steady-state power consumption of the file system. Assignment performs a one-to-one mapping between a group and a disk. Assignment determines the transient power consumption while files are migrated between disks. In this section, we describe file system policies in a mathematical context to support our policy design in Section 4. Table 1 contains all symbols used in the formulation as a reference.

3.1. File Grouping

Energy-efficient file grouping minimizes the average power consumption subject to bandwidth constraints. The problem is formally stated as:

$$\underset{j,X_{ij}}{\textbf{minimize}} \quad \bar{P} = \frac{1}{T}\sum_{i=1}^{n_d}\Big[P_{ij,a}t_{ij,a} + E_{i,c} + P_{ij,a}t_{ij,m} + P_{ij,r}(T - t_{ij,a} - t_{ij,m} - t_{i,c})\Big] \quad (1)$$

$$\textbf{subject to} \quad \sum_{k=1}^{n_f} b_k X_{ik} < B_{ij},\ i \in \{1,2,\ldots,n_d\} \quad (2)$$

$$\sum_{k=1}^{n_f} s_k X_{ik} \le S_i,\ i \in \{1,2,\ldots,n_d\} \quad (3)$$

$$\sum_{i=1}^{n_d} X_{ik} = 1,\ k \in \{1,2,\ldots,n_f\} \quad (4)$$

Equation (1) defines the average power $\bar{P}$ for all file locations. This equation consists of four energy components, divided by the evaluation interval T. Each of the power measurements, $P_{ij,a}$ and $P_{ij,r}$, and the times, $t_{ij,a}$ and $t_{ij,m}$, are dependent upon the disk speed j. The first component is the group's active energy: $P_{ij,a}t_{ij,a}$. The active energy is computed from the amount of time that the group's disk is actively serving requests. The second component is the speed change energy: $E_{i,c}$. This energy occurs when the group's disk motor changes speeds. If the disk does not change speeds, $E_{i,c} = 0$. The speed transition time $t_{i,c}$ is a hardware-specific constant. The third component is the energy required to migrate files: $P_{ij,a}t_{ij,m}$. The migration energy represents additional disk activity while files are moved between groups' disks. The final component is the idle energy: $P_{ij,r}(T - t_{ij,a} - t_{ij,m} - t_{i,c})$. This energy is consumed when no requests are issued. The time component is the remaining time in the evaluation interval when requests are not being issued, migration is completed, and state changes have completed.

The times $t_{ij,a}$ and $t_{ij,m}$ are directly related to the file grouping and the group's disk bandwidth B_{ij}. We compute the active time as $t_{ij,a} = \frac{T}{B_{ij}}\sum_{k=1}^{n_f} b_k X_{ik}$ for disk i, where the indicator variable $X_{ik} = 1$ when the file k is located within group i and $X_{ik} = 0$ otherwise. The time $t_{ij,m}$ includes the time required to migrate files into the group and out of the group. For simplicity, we assume that the times $t_{ij,a}$ and $t_{ij,m}$ are disjoint because seek operations are likely to separate the requests. We will discuss the migration time $t_{ij,m}$ further in Section 3.2.

Equation (2) represents the bandwidth constraints for file groupings. The bandwidth required to read each file, b_k, must not exceed the total bandwidth B_{ij} of the group's disk. The bandwidth B_{ij} is dependent on disk speed j. Equation (3) represents the capacity constraints for file grouping. The total size of each file, s_k, must not exceed the available storage capacity S_i for the group's disk. Equation (4) indicates that each file must be located exclusively on one disk. We do not consider file duplication within this formulation.

3.2. Disk Assignment

As described in Section 2, existing approaches to minimizing average power focus on computing a file grouping that minimizes the active energy and idle energy. Such a minimization neglects the significant energy consumption due to file migration. In this section, we formulate the assignment of file groups to physical disks in greater detail. We assume that disk groupings have been constructed. Then, we can determine an optimal permutation of disk assignments that minimizes the migration energy component $P_{ij,a}t_{ij,m}$. Let Y_{id} be an indicator variable when the group i is assigned to disk d:

$$\textbf{minimize} \sum_{d=1}^{n_d}\sum_{i=1}^{n_d} P_{ij,a}t_{ij,m}Y_{id} \quad (5)$$

$$\textbf{subject to} \sum_{i=1}^{n_d} Y_{id} = 1, d \in \{1,2,\ldots,n_d\} \quad (6)$$

$$\sum_{d=1}^{n_d} Y_{id} = 1, i \in \{1,2,\ldots,n_d\} \quad (7)$$

Equation (5) defines the energy to migrate data between disks. The migration time $t_{ij,m}$ depends on the size of files s_k that change between each group and the disk speed j. Let $M_k = 1$ indicate that file k is migrated into group i or out of group i; otherwise, $M_k = 0$. Using this definition, the migration time is $t_{ij,m} = \frac{1}{B_{ij}}\sum_{k=1}^{n_f} s_k M_k$. Equation (6) and Equation (7) define a one-to-one correspondence between disk grouping and disk assignment. That is, each group is assigned to exactly one disk.

3.3. Grouping and Assignment Joint Solution

The previous formulation describes file system policies as a sequence of two independent problems. However, a close relationship exists between the optimality of a file grouping X and the disk assignment Y for a file system. The file grouping determines the steady-state power consumption for each disk. The disk assignment

```
GFS(files, disks)
 1  pool = BUILD-INITIAL-POOL(files, disks)
 2  N = SIZE(pool)
 3  while VARIANCE(pool) < TOLERANCE and runtime < 600 s
 4        do new_pool = {BEST-FIT(pool), BEST-FIT(pool)}
 5           for i in 1 to N/2 − 1
 6                 do (s1, s2) = SELECT-SOLUTIONS(pool)
 7                    CROSSOVER(s1, s2)
 8                    MUTATE(s1, s2)
 9                    new_pool = new_pool ∪ {s1, s2}
10           pool = new_pool
```

Figure 3. Pseudocode for Genetic File System.

indicates which files must be migrated between disks, influencing transient power consumption and creating additional disk traffic. In the next section, we combine the two components and solve the problems simultaneously.

4. Genetic File System

The exact solution to the file location problem is a direct application of the multiple constrained knapsack problem, which is known to be NP-complete [13]. For our particular experiments, 204 videos on seven disks produce 2.51×10^{172} ($= 7^{204}$) possible solutions for file groupings. Furthermore, the problem must be solved periodically because access patterns may change over time. To manage complexity, we solve the optimization problem using a genetic algorithm, creating a Genetic File System (GFS) to avoid an exhaustive search of $n_d^{n_f}$ possible solutions.

4.1. Genetic Algorithm

In GFS, we represent each solution as a tuple $(a_1, a_2, \ldots, a_i, \ldots, a_{n_f})$. The i-th element a_i represents the disk assignment for the i-th file sorted in order of decreasing bandwidth requirement. We refer to the set of N candidate solutions as the mating pool. Each iteration of the genetic algorithm is called a generation. The next generation of the genetic algorithm is constructed by performing three operations: crossover, mutation, and fitness evaluation. Crossover mixes two existing solutions with probability p_c and acts as a "mating" process between two solutions. Mutation randomly changes the assignment of a single file within a solution with probability p_m. Fitness evaluation computes the energy consumption for each solution using Equation (1).

Figure 3 describes GFS in pseudocode. The GFS algorithm begins by constructing an initial mating pool. We construct our initial pool based on observations of Equation (1) and our experimental hardware. The active energy $P_{ij,a}t_{ij,a}$ is lower for high-speed disks than low-speed disks because the ratio between bandwidths (B_{high}/B_{low}) exceeds the ratio between power consumptions ($P_{high,a}/P_{low,a}$). Hence, we construct initial solutions that are likely to group popular files together on high-speed disks. If $P_{ij,a}t_{ij,a}$ is higher for high-speed disks than low-speed disks, GFS may simply be altered to balance files rather than cluster files. We induce these two behaviors by using a random number generator where the probability of assigning a file to disk d is $p(d) = \frac{1}{a^d} / \sum_{i=1}^{n_d} \frac{1}{a^i}$. We will refer to this random number generator as the "power law generator." The value of a determines how likely files are grouped together. As a approaches one, the solutions become more evenly balanced across disks. As a becomes large, the power law generator produces solutions more similar to PDC. For our experiments, we use $a = 1.6$ to induce a more diverse mating pool than larger values of a and rely on the crossover operation to group files. We also include the PDC solution and the current file assignment in the initial mating pool. These two special solutions represent the extremes of the algorithm, minimizing steady-state energy and minimizing migration energy, respectively.

Lines 3 – 10 loop until the mating pool's variance is less than a specified tolerance or 600 seconds have elapsed. GFS uses a tolerance of 0.01 W^2. GFS retains the best known solution between each iteration (line 4) to ensure that solutions do not worsen over time and then fills the remainder of the next generation's mating pool with new solutions. Line 6 chooses two solutions from the mating pool. We use the roulette wheel selection method where the probability of choosing a solution is proportional to its fitness [16]. Line 7 performs a crossover between the two solutions with probability p_c. We use the one-point crossover method, where the solutions are split at a random point and the trailing elements are swapped between the two solutions [16]. GFS uses a uniformly distributed random number generator to determine where to split the solutions. If crossover produces a solution that exceeds the bandwidth or capacity constraints on a disk, we randomly assign a new disk for overallocated files using the power law generator. Line 8 mutates one file assignment within each solution with probability p_m. The mutated assignment is selected using the uniform random generator, but the new disk assignment is computed by the power law generator. After generating each solution for the next generation's mating pool,

the solution's disks are assigned the lowest speeds that satisfy bandwidth constraints. Then, the fitness of each solution is computed using Equation (1).

4.2. Example

This section provides examples to illustrate the methods described in the GFS algorithm more clearly.

4.2.1. Initial Mating Pool. Suppose we have a file system with $n_f = 8$ and $n_d = 3$. Each disk has a high-speed bandwidth of 20 and a low-speed bandwidth of 3. The bandwidth requirement of each file b and bandwidth consumption for each disk B is:

$$b = (10,8,6,5,2,1,1,1)$$
$$B = (0,0,0)$$

We randomly select each file's assignment with the power law generator. By considering the files in order of decreasing bandwidth, we reduce the likelihood that a file does not fit due to bandwidth constraints. After the first four files, the assignment A may be produced:

$$A = (1,2,1,1,-,-,-,-)$$
$$B = (21,8,0)$$

The '–' symbol indicates that the elements have not been assigned. Assigning the element $a_4 = 1$ exceeds the bandwidth for Disk 1. The total bandwidth requirement for a_1, a_3, and a_4 is 21. However, the maximum high-speed bandwidth is 20. Since this assignment exceeds the bandwidth for Disk 1, we generate another assignment from the power law generator. Performing this process for the remaining elements may generate the following assignment:

$$A = (1,2,1,2,1,2,3,1)$$
$$B = (19,14,1)$$

The disk assignments require that Disk 1 and Disk 2 operate at high-speed. However, Disk 3 has a low bandwidth requirement, allowing the disk to save energy by operating at low-speed.

4.2.2. Crossover. Suppose that two solutions A and C are selected for crossover:

$$A = (a_1, a_2, \ldots, a_i, a_{i+1}, \ldots a_{n_f})$$
$$C = (c_1, c_2, \ldots, c_i, c_{i+1}, \ldots c_{n_f})$$

	3000 rpm	15000 rpm
Seek time	3.4 ms	3.4 ms
Rotation time	10 ms	2 ms
Effective bandwidth	6.2 MB/s	42.1 MB/s
Active power	6.1 W	13.5 W
Idle power	2.8 W	10.2 W
Speed change time	12.4 s	12.4 s
Speed change power	152 J	152 J

Table 2. Energy parameters for disks [6]. Effective bandwidth is maximum sustainable bandwidth without dropping video frames.

The split point i is randomly selected by the uniform random generator. Then, the solutions A' and C' inserted into the next generation's mating pool are:

$$A' = (a_1, a_2, \ldots, a_i, c_{i+1}, \ldots c_{n_f})$$
$$C' = (c_1, c_2, \ldots, c_i, a_{i+1}, \ldots a_{n_f})$$

4.2.3. Mutation. Suppose that a solution A is selected for mutation. The mutated element a_i is selected by the uniform random generator. The new disk assignment d for a_i is selected by the power law generator. The new solution:

$$A' = (a_1, a_2, \ldots, d, a_{i+1}, \ldots, a_{n_f})$$

is inserted into the next generation's mating pool.

5. System Implementation

5.1. Hardware Setup

Our experimental machine is a Dell PowerEdge 2900 server with two quad-core 2.6 GHz processors, 8 GB RAM, and eight 37 GB 15000 rpm SAS disks. One disk is used as a system disk. The remaining seven disks contain data for our server. The server hosts 204 MPEG-2 streaming videos ranging in size from 111 MB to 443 MB. To generate workloads for our server, we use up to 130 media clients that select files based upon a Zipf distribution: $p(k) = \frac{1}{k^\alpha} / \sum_{i=1}^{n_f} \frac{1}{k^\alpha}$, where k is the relative popularity of a file and α is a shape parameter. Gao et al. [17] suggest $\alpha = 0.739$ to model typical user behavior for video files. The number of clients over time is illustrated in Figure 4.

We use the power parameters for DRPM disks shown in Table 2 and emulate different disk speeds by delaying each disk request in the kernel's disk queue, as suggested by Zhu et al. [6] Similarly, we emulate speed transitions by delaying one request by the transition time $t_{i,c}$. We determine the effective bandwidth at different DRPM speeds by computing the total bitrate of videos that may be serviced from a disk

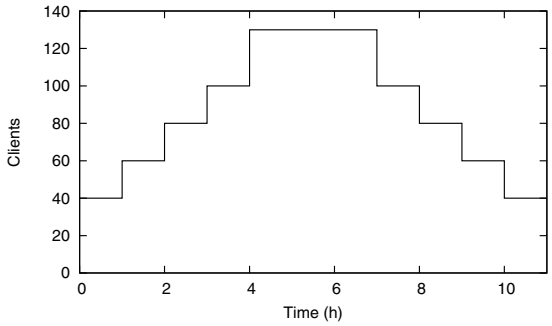


Figure 4. Number of clients during experimental workload.

without dropping frames. Our experiments assume that the servers have completed a one-hour warm-up period before file migration.

5.2. Recording File Bandwidth

Our policy requires the bandwidth consumed by each file to determine file location. We adopt the approach by Pinheiro et al. to compute each file's bandwidth by dividing each file's size by its average access interarrival time [5]. We use a software framework called HAPPI to record the accesses for each file. HAPPI provides an interface to the Linux kernel that simplifies policy implementation [18]. We implement a kernel module that accepts disk requests from HAPPI and computes the interarrival time for each file.

To reduce computational overhead, we sample incoming disk requests by recording every 128th request. We notice little difference in the relative popularity of files using this mechanism. We select every 128th request because it is the highest sampling rate where we do not encounter lock contention within our kernel module. To handle bursty requests, we treat requests that occur closely in time as a single request. If a request occurs less than 20 seconds after the previous request, the idle interval between the two requests is ignored. The 20 second length is determined by the application program. For our streaming videos, the buffer refill interval is usually between 5 and 15 seconds. We use 20 seconds to provide a filter against unusually long interarrival times. The interarrival time for each file is exported to user-space using the `debugfs` virtual file system.

5.3. Migrating Files Between Disks

We implement migration in user space because it (a) provides a simple interface and (b) is independent of file system formats. Our file system is represented by a directory structure on the system disk consisting of symbolic links to data files located on other physical disks. By using symbolic links, applications do not need to know the current physical location of files to continue serving data.

We implement our file system policies in a user-space daemon. The daemon extracts the interarrival time for each file from the `debugfs` file system and computes bandwidth requirements once per hour. In this paper, we recompute groupings and assignments hourly to reduce the number of disk speed transitions. After computing groupings and assignments, the daemon migrates files by performing the following process:

1) Copy the source file to the destination.
2) Remove the symbolic link to the source file.
3) Restore the symbolic link to the destination file.
4) Delete the source file.

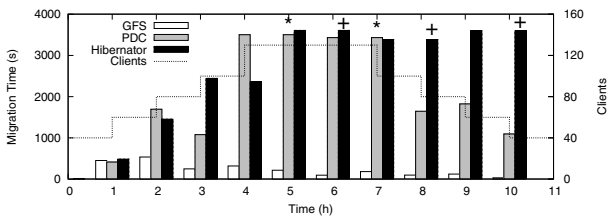
When migrating files, we assign a low priority to our migration daemon to reduce the migration's impact on disk bandwidth. We also migrate files in order of decreasing interarrival time to leave the most likely accessed data in the operating system's page cache.

We observe that a small window of vulnerability exists between Step 2 and Step 3 where a new request will not locate a file on the file system. Profiling indicates the window lasts an average of 3.5 ms and is independent of file size. To determine the likelihood of accessing a file during this window, we read a set of files for an hour and compute the number of video frames dropped due to this window of vulnerability when performing Step 2 and Step 3 once per second. Our results indicate all 142,051 frames are played successfully without dropping a frame. If other applications are unable to risk a loss of quality-of-service, the window may be closed by introducing a system call that performs Step 2 and Step 3 atomically.

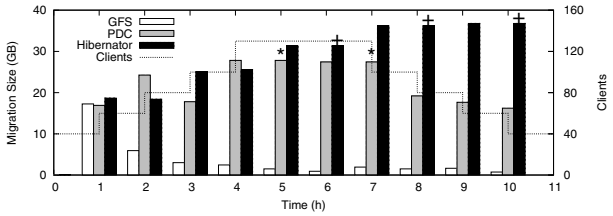
6. Policy Comparison

In this section, we compare GFS with the existing PDC policy [5] and a policy similar to Hibernator [6] that randomly distributes files across disks of similar speed to improve performance. We use a variation of Hibernator because the exact implementation requires a RAID-5 configuration and migrates RAID blocks rather than entire files.

Figure 5(a) illustrates the time required to complete file migration for each of the three policies. On this figure, a smaller value is better. This time is different



(a)



(b)

Figure 5. (a) Migration time for GFS, PDC, and Hibernator. (b) Migration size for GFS, PDC, and Hibernator. '*' indicates previous PDC migration exceeded one hour. '+' indicates previous Hibernator migration exceeded one hour.

from t_m in Section 3 because server requests (t_a) are interleaved with migration requests during this time. For reference, the number of clients is overlaid on the figure. PDC's migration time follows the number of clients very closely. Figure 5(b) shows the size of each migration in gigabytes. We observe that PDC's migration sizes follow a similar pattern. Large migrations near the beginning of the workload require less time than similarly-sized migrations in the middle of the workload because fewer server requests are interleaved with the migration requests. At five hours and seven hours, PDC performs large migrations that require more than one hour to complete causing the next migration to be delayed by an hour. These points are indicated by a '*' on Figure 5(a) and Figure 5(b). PDC performs such large migrations under heavy load that migration is unable to complete. We observe two important phenomena with PDC. First, the majority of files are migrated between high-speed disks. This churn is unnecessary unless the high-speed disks exceed their bandwidth constraints. Second, the low-speed disks' utilizations become very low because their files are migrated after being accessed.

Figure 5(a) indicates that Hibernator's migration time steadily increases because Hibernator randomly shuffles files across all disks of a similar speed. As more files are accessed, the migration time increases. Similar to PDC, Hibernator performs large migrations (indicated by '+') under heavy load that do not complete during the migration interval. The migration sizes in Figure 5(b) begin to level off near the end of the workload after all videos have been accessed. Regardless of activity level, the number of files migrated per period will not decrease during our experiment.

	PDC	Hibernator	GFS	Δ
Migration size	20.9 GB	27.4 GB	3.7 GB	82%
Migration time	1834 s	2474 s	228 s	88%
Average power	36.6 W	36.6 W	35.6 W	3%
Dropped Frames	0.20%	0.25%	0.075%	63%

Table 3. Average migration size, migration time, average power, and dropped frames for policies. Δ is percent improvement compared with the better existing policy. Files are migrated every hour.

In contrast to PDC and Hibernator, GFS generally exhibits a low migration time. Since GFS jointly minimizes steady-state power consumption and migration energy, many unnecessary migrations are removed. GFS does not migrate files between disks of the same speed because steady-state energy cannot be reduced as a result of the migration. Figure 5(b) illustrates that GFS performs few migrations after the first hour because most popular files are already located on high-speed disks. Furthermore, GFS tends to utilize low-speed disks more effectively by avoiding the migration of infrequently accessed files that gain little energy savings by being stored on high-speed disks.

Table 3 summarizes the average migration size, migration time, power, and dropped frames for the three policies. PDC moves an average of 20.9 GB of data during each migration, requiring in excess of 30 minutes on average. Since many files are migrated between highly utilized disks, 0.20% of frames are dropped during the experiment. Hibernator improves steady-state bandwidth by balancing the load across disks of a similar speed. However, Hibernator drops 0.25% of all frames because the policy migrates too many files between disks to achieve steady-state improvements. GFS significantly reduces the migration time and size because GFS only migrates files when the steady-state energy savings exceeds the migration energy. On average, GFS migrates 3.7 GB of files between disks, requiring less than four minutes. Since disks incur less migration traffic, quality-of-service violations are reduced to 0.08% dropped frames, an improvement of 63%. We also observe that GFS achieves comparable power savings, reducing power slightly by 3%.

7. Discussion and Future Work

The experiments in Section 6 indicate that quality-of-service for a streaming video server may be signifi-

cantly improved while maintaining comparable energy consumption by using GFS. An optimization to GFS is to migrate files as they are read by the application, reducing migration bandwidth. GFS migrates files periodically to avoid a performance bottleneck if a large number of files must be migrated simultaneously. Future work may implement a combination of periodic migration and event-driven migration to reduce migration bandwidth while avoiding migration penalties.

We utilize observations from our problem formulation in Section 3 to narrow the massive search space. Other heuristics to limit the search space may produce additional quality-of-service improvements and energy savings. In our future work, we intend to apply GFS to different types of workloads and evaluate GFS's effectiveness in low-latency applications, such as databases and web search. Furthermore, we intend to explore the inclusion of an analytical trade-off between quality-of-service and energy savings within our equations.

8. Conclusion

File system policies induce idleness in high-performance servers by grouping frequently accessed files together. This idleness may be exploited to reduce power consumption. In this work, we present a file system policy called Genetic File System (GFS) that significantly reduces the number of files migrated between disks while maintaining bandwidth constraints. GFS only migrates files when the steady-state energy savings exceed the migration energy. Our experiments with a streaming video server indicate that GFS may reduce quality-of-service violations by 63% over existing approaches while maintaining similar power savings by reducing the number of migrated files.

Acknowledgment

The authors would like to thank David Canada and the anonymous reviewers for their suggestions. Nathaniel Pettis is supported by the Bilsland Fellowship. This work is supported in part by NSF CAREER CNS-0347466 and CCF-0541267. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] R. Joseph and M. Martonosi, "Run-time Power Estimation in High Performance Microprocessors," in *International Symposium on Low Power Electronics and Design*, 2001, pp. 135–140.

[2] Transaction Processing Performance Council, "PowerEdge 2900/1/2.33 GHz/2x4M Executive Summary," http://www.tpc.org/results/individual_results/Dell/Dell_Inc_91_070423_ES.pdf, April 2007.

[3] Dell, "Dell PowerEdge 2900 Server," http://www.dell.com/downloads/global/products/pedge/en/2900_specs.pdf, June 2006.

[4] ——, "Dell PowerVault MD1000 Modular Disk Storage Expansion Enclosure," http://www.dell.com/downloads/global/products/pvaul/en/pvaul_md1000_specs.pdf, June 2006.

[5] E. Pinheiro and R. Bianchini, "Energy Conservation Techniques for Disk Array-based Servers," in *International Conference on Supercomputing*, 2004, pp. 68–78.

[6] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: Helping Disk Arrays Sleep through the Winter," in *ACM Symposium on Operating Systems Principles*, 2005, pp. 177–190.

[7] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "Reducing Disk Power Consumption in Servers With DRPM," *IEEE Computer*, vol. 36, no. 12, pp. 59–66, December 2003.

[8] D. Colarelli and D. Grunwald, "Massive Arrays of Idle Disks for Storage Archives," in *ACM/IEEE Conference on Supercomputing*, 2002, pp. 1–11.

[9] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level Power Management for Dense Blade Servers," in *International Symposium on Computer Architecture*, 2006, pp. 66–77.

[10] A. E. Papathanasiou and M. L. Scott, "Energy Efficient Prefetching and Caching," in *USENIX Annual Technical Conference*, 2004, pp. 255–268.

[11] L. Cai and Y.-H. Lu, "Power Reduction of Multiple Disks Using Dynamic Cache Resizing and SpeedControl," in *International Symposium on Low Power Electronics and Design*, 2006, pp. 186–190.

[12] Q. Zhu, F. M. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," in *International Symposium on High-Performance Computer Architecture*, 2004, pp. 118–129.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, 1979.

[14] A. Homaifar, S. Guan, and G. E. Liepins, "A New Approach on the Traveling Salesman Problem by Genetic Algorithms," in *International Conference on Genetic Algorithms*, 1993, pp. 460–466.

[15] S. Khuri, T. Bäck, and J. Heitkötter, "The Zero/One Multiple Knapsack Problem and Genetic Algorithms," in *ACM Symposium on Applied Computing*, 1994, pp. 188–193.

[16] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*, 2nd ed. Wiley and Sons, 2001.

[17] L. Gao, Z.-L. Zhang, and D. Towsley, "Proxy-assisted Techniques for Delivering Continuous Multimedia Streams," in *IEEE/ACM Transactions on Networking*, December 2003, pp. 884–894.

[18] N. Pettis, J. Ridenour, and Y.-H. Lu, "Automatic Run-Time Selection of Power Policies for Operating Systems," in *Design Automation and Test in Europe*, 2006, pp. 508–513.