The Pennsylvania State University

The Graduate School

College of Engineering

OPTIMIZING STORAGE SYSTEM POWER AND PERFORMANCE

A Thesis in
Computer Science and Engineering
by
Rajat Garg

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2011

Abstract

The quest of building bigger and better computing systems has resulted in tremendous growth in the size of the storage systems. Not only have they grown in their size, they play a significant role in determining the overall performance of the applications and success of the entire computing system. While the industry is concerned about reducing the huge costs involved in running/maintaining these storage systems, the scientific community has been pushing the limit to achieve maximum performance. Apart from the contrasting demands of saving power vs. maximum performance, there exist scenarios where a balance of power consumption and performance is expected. In this body of work, we propose/study software based techniques that will help achieve some or all of the above requirements.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude towards my advisor Dr. Mahmut Kandemir for guiding me all the way through and correcting me for my mistakes. I would also like to thank Dr. Mary Jane Irwin, Dr. Padma Raghavan, and Dr. Piotr Berman for their support and valuable suggestions. My family and friends have provided me emotional support and were my source of inspiration. Lastly, special thanks to Karen and Cindy for making things easier within my department.

Chapter 1

Introduction

In order to meet the increasing demands of present and upcoming data-intensive computer applications, there has been a major shift in the disk subsystem, which now consists of more disks with higher storage capacities and higher rotational speeds. These have made the disk subsystem a major consumer of power, making disk power management an important issue. People have considered the option of spinning down the disk during periods of idleness or serving the requests at lower rotational speeds when performance is not an issue. Accurately predicting future disk idle periods is crucial to such schemes. An important characteristic of many high performance applications is disk intensiveness. Many scientific simulation codes for example have frequent disk accesses. In fact, large-scale simulations have become an integral, important, and in many cases, primary approach to solving complex science and engineering problems and automating scientific knowledge discovery. These simulations tend to generate huge amounts of data that must be stored on disks, mined, analyzed, evaluated, check-pointed (on disks), and steered, in most cases dynamically during the course of execution. Similarly, many data base/mining applications frequently exercise disk systems of parallel architectures. In a disk storage system shared by multiple applications, one of the critical problems is resource (disk) allocation across these applications. The main characteristic that makes this problem challenging is the fact that different applications demand different amounts of disk space (capacity) and can tolerate different disk latencies. As a result, allocating disks from a heterogeneous disk pool to satisfy the needs of all applications using the same storage system is non-trivial. Existence of additional constraints such as power and reliability makes this disk allocation problem even harder. In many other scenarios, applications may demand certain quality of service (QoS). This may be measured in terms of latency of a single i/o request or the overall bandwidth. Ensuring QoS becomes non-trivial in scenarios where multiple applications are contending for the same resource. As an example, consider two applications sharing main memory of a compute node. All the above requirements expected from the storage system make it an important topic of research.

Chapter 2

Markov Model Based Disk Power Management for Data Intensive Workloads

2.1 Introduction

It is well understood that reducing the energy requirements of portable devices is important to prolong battery life. But when it comes to large storage systems, making them bigger and increasingly powerful has been the priority, in order to attain the demanded availability and performance. Processors have become extremely powerful, making them more data hungry, and so have the data storage needs, leading to a tremendous growth in the energy consumption of present data centers [6]. In a typical data center, storage system contributes to more than 25% of total power consumption [62]. Apart from the energy consumed for disk operations, cooling costs are also a major concern for this high-density equipment [23]. In fact, the costs have already become the second largest contributor to data center total cost of ownership (TCO) [24]. High density racks and blade servers help reduce total power consumption, but their power density levels exceed the limits of many facilities.

Increasing the number of disks, apart from increasing the total storage space, also helps improve the performance, as data distributed across the disks can now be accessed simultaneously [9]. The reason for the rise in energy consumption is the way disks operate. Disks are made to service the requests at their maximum speeds. Normally, they continue spinning at their maximum rotational speed even if they are *not* servicing any request and hence contribute to the wastage of energy. A direct approach to reducing this energy wastage is to shut down all those components that are not doing any useful work at the moment. Much research has been done to obtain gains from this approach. Two important issues arise in this context:

- How accurately can we predict the occurrence of idle times?
- What would be the energy/performance tradeoffs if we decide to shut down (spin down in the context of disks)?

Recently, techniques that employ multi-speed disks [37] have also proposed and evaluated. With such techniques, when there is a slack (allowable increase in latency), the disk is rotated at a lower speed (compared to the maximum speed available), instead of being completely spun down. The choice of speed is based on the length of the available slack. This approach has been shown to be more applicable to high-performance scientific and data-intensive workloads where disk idle periods are typically small but numerous [17, 55]. While the main problem with spinning-down techniques is that they may not be applicable to short idle times; the problem with multi-speed disks is the large performance penalty incurred if disk idle and active periods are not predicted accurately. Focusing on a three-speed disk, in this paper we propose and experimentally evaluate a novel Markov chain [46] based disk power reduction scheme. Our main contributions can be summarized as follows:

- A Markov model to help disk power management. The rationale behind using a Markov model is that disk access patterns exhibit a repetitive behavior and can therefore be captured by using such a model. First, building a Markov model for a given disk system is presented, followed by the mechanism for making use of this model.
- A three-speed disk model. The need to have such a disk is discussed in detail and its benefits are assessed.
- A prediction scheme. We introduce a scheme that uses the information from the Markov model of the disk system to predict future states of the system (in terms of active and idle periods of disks).

- A runtime approach. This approach uses the Markov model, the three-speed disk model, and the prediction scheme for achieving disk energy savings. The approach decides what needs to be done and when.

Our experiments with various workloads, which include both synthetic traces and traces extracted from real applications, indicate that the Markov model effectively captures the behavior of the disk system. The success of our proposed scheme can be attributed to being able to predict the future states of the system. Since, our approach is *proactive*, meaning the idle periods are predicted in advance, the opportunities to save power are rarely missed (on mispredictions) and also are fully utilized, spinning down to the lowest power mode with little impact on performance. The use of the three-speed disk helps make the most of long idle times by entering the standby mode, additionally giving the flexibility to save energy even when idle times (spin-down to a lower speed) are short.

2.2 Markov Model for Disk Idleness Prediction

We model the disk state transitions using Markov modeling. A Markov model for a system can be completely specified by the total number of states n and the transition probability matrix P [29]. The number of potential states for a N-disk system is given by 2^N (here n=2^N). This is because a disk is either busy (*ON*, represented by 1) servicing a request, or idle (*OFF*, represented by 0). Given the present state and all past states, if the future state of the system depends only on the present state, the system is said to have the *Markov property*. The transition probability matrix is a square matrix of size $n \times n$, where n is the number of states in the system. Values contained in the matrix are probabilities, where P_{ij} (located in i^{th} row and j^{th} column) is the probability of transitioning from state i to state j.

In the context of disk power minimization, one can build a transition probability matrix by *sampling* the state of the disk system at regular intervals (states representing disk being accessed or not accessed). We sample all the disks at runtime, noting whether the disk was accessed during the last sampling period. If it was, then the bit is set for the corresponding disk; otherwise it is reset. Even if a disk access starts toward the end of the sampling period (thus leaving the system in a state of transition at the sampling point), we conservatively assume that the disk was *ON* during the whole sampling period. However, this assumption will not be made while calculating the energy for the base case. We represent the state of the system as a *bit vector*. For an eight-disk system, it will be an eight-bit vector represented as $D_1D_2D_3D_4D_5D_6D_7D_8$ (D_i stands for the i^{th} disk in the disk subsystem) and an example state would be 11001111, which indicates that except disks D_3 and D_4, all others were accessed. The transition probability matrix is built and updated during runtime with the help of these samples. There is a *warm-up period* (explained in Section 4.5), during which the workload characteristics are monitored to help mature the matrix, making it suitable for making predictions on future states (*ON/OFF*) of disks in the system. The probability matrix is updated at regular intervals by including the most recent set of samples. Because of this regular update on the probability matrix, our scheme is able to keep the up-to-date state of changing or mixed workloads. Note that both sampling frequency for the disk subsystem and the update frequency for the probability matrix have to be chosen carefully. We later study in Section 4.5 how crucial is the value of the sampling period.

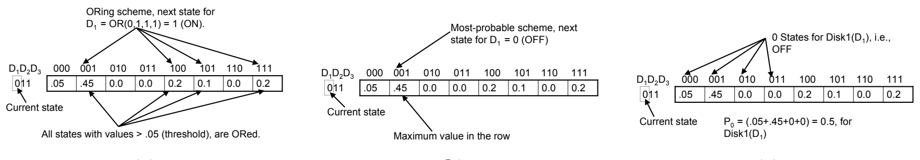


Figure 1: Example showing the outcome of predictions with different schemes specific to D_1: (a) ORing, (b) Most-probable, and (c) Summing. Note that our defended scheme (Summing) is different from the ORing and Most-probable schemes, and might as well transition the system to a state which has zero probability in the probability matrix.

2.3 Prediction Schemes

Transition probability matrix by itself is of no use as far as power reduction is concerned. There is need for a prediction algorithm that predicts the next state for the system by using the information maintained by the probability matrix. We can evaluate the accuracy of a given prediction algorithm by comparing the percentage of matches between the *actual* and *predicted states*. Below, we describe four prediction schemes evaluated in this work. These schemes are *1-step lookahead schemes*, meaning that only the state that directly follows the present state is predicted and none that may happen after this predicted state. We note that predicting the next state from the current state requires indexing into an appropriate row of the probability matrix. This row is determined by the current actual state of the system. Remember that the row and column number of the matrix represent the states and the matrix itself consists of transition probabilities.

- *ORing* (conservative): After indexing into the correct row, OR all the states (recall that state is represented as a bit vector) with transition probabilities greater than a certain probability (0.05 for our case) to get the next state prediction. The rationale behind this scheme is to never predict an idleness if the probability of the disk being *ON* in the next state is greater than some minimum. This scheme tends to produce an *ON* prediction most of the time, not usually giving a performance penalty but providing little power saving opportunities.

- *Most-probable* (aggressive): After indexing into the correct row, predict the next state based on the highest transition probability from the current state. Since we are just selecting the maximum value in the row, it does not necessarily have to be a large value. For example, it may be 0.05 and still be the maximum if other values in the row are all individually less than 0.05 (but they all add up to 0.95). As a result, this scheme might predict an *OFF* even on a value of 0.05. This scheme does produce good energy savings, but it may also lead to a performance penalty, resulting in spin-downs even when not desirable.

- *Last-state* (does not use the probability matrix): The next predicted state will be the last known state of the system. This is the value we used in all other schemes for indexing into the appropriate row (the current actual state). The success of this scheme is based on the assumption that the system possesses some inertia and hence will continue to remain in its present state for some time. The duration of this period is the crucial factor in the success or failure of the scheme. When the sampling period is kept small, the scheme is bound to give good results. We included this scheme in our evaluations to provide us with a baseline. We note that this scheme also inherently makes use of the Markov property by considering only the last state for future predictions.

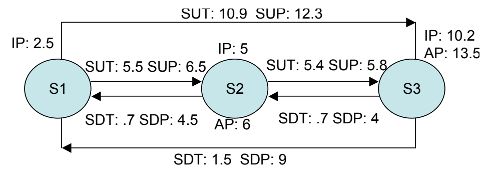
- *Summing* (the scheme defended in this paper): In this scheme, after indexing into the correct row, we sum all probabilities leading to a 0 (*OFF* state). This is done for each disk separately to obtain its next state. If probability of transitioning to 0 (denoted by P_0) is greater than certain threshold, then we decide to turn the disk *OFF* else it is kept *ON*. Note that this scheme is slightly modified when used for disks with more than three levels, such that the threshold value changes to a range defined for each speed level.

An arbitrary row chosen from the transition probability matrix of our system is shown in Figure 14. This row contains eight entries (for a three-disk system, the number of possible states is 2^3), each entry being a probability for a three-disk system ($D_1D_2D_3$). States are represented as a bit vector with the leftmost bit for the first disk (D_1). Figures 14(a) and 14(b) show the results obtained using the ORing scheme and the Most-probable scheme, respectively. Figure 14(c), on the other hand shows how the Last-state scheme predicts the next state for D_1. Figure 14(d) depicts the computation of P_0 (probability of transitioning to 0), which if greater than, for example, 0.7, will give an *OFF* prediction. How we decide this threshold value is discussed later in Section 4.5. This scheme (Summing) is expected to give good energy savings without hurting the performance. Results of prediction accuracies obtained with these schemes are discussed in Section 2.6.2.

2.4 Three-Speed Disk

In this section, we describe *three-speed disk* that will be used for evaluating our power management scheme. The conventional two-speed disk either runs at the maximum speed or stays in the standby (spin-down) mode where it does not spin at all. The constraint of operating in one of these two modes does not give the flexibility of transitioning to a lower-power mode when the duration of the idle time is less than the break-even time.[1] Since we sample the disk system without looking at the actual start times of idle periods, we might miss some of these idle time opportunities. We also use a prediction scheme to guess the idle times that were captured during our sampling. Therefore, using a conventional (two-speed) disk would not give us much opportunity to save disk energy most of the time. Thus, the motivation for using a three-speed disk is to have the ability to capitalize on all the idle time opportunities that we are able to predict and to have enough flexibility to save energy even when the disk idle times are not long enough for the two-speed disk. We note that, when we refer to saving energy, minimizing the performance penalty automatically goes along with

[1]Break-even time is the minimum amount of idle time for which spinning down a disk brings some energy benefits without increasing original execution latency [36].



(a)

IBM Ultrastar 36Z15	
Individual Disk Capacity	18.4 GB
Maximum Disk Rotation Speed (S3)	15000 RPM
Intermediate Speed Level (S2)	7000 RPM
Minimum Disk Speed (S1)	0 RPM

(b)

Figure 2: Three-speed disk. (a) State model. SUT: Spin-up Time; SDT: Spin-down Time; SUP: Spin-up Power; SDP: Spin-down Power; AP: Active Power; and IP: Idle Power. Time is in seconds, and power is in watts. (b) Specification for the three-speed disk model.

it. The flexibility with the three-speed disk comes from the intermediate level of operation, where we spin the disk at half of its maximum speed. A request when serviced at the intermediate speed almost doubles the service time but reduces the energy consumption by a factor of four [17]. The specifications of the three-speed disk along with the disk model we employ are provided in Figure 2. State transition times and energies are based on the linear power model given in [17], and the disk specifications have been extended for an IBM hard-disk [21]. We also note that disks with such multi-speed capabilities, such as Western Digital Caviar GP [37] and Sony multi-mode disk [39], are now commercially available in the market, though they are not server-class disks.

2.5 Algorithm

With the Markov model representing the disk state transitions and accompanied by a prediction scheme that helps predict the next state of the disk, there is a need to have an overall *control strategy* that can make high-level decisions for power management of an I/O subsystem consisting of the proposed three-speed disks. This requires making two important decisions:

- When can a disk be spun down (should try to maximize the energy savings but it does not matter if we miss some opportunities)?
- When should a disk be spun up (should not miss to spin-up when required)?

Depending on how aggressively one makes these decisions, it can result in different energy savings and performance degradations. To make a decision for the next state, we look at the probability P_0 (probability of transitioning to a 0 state) for each disk. In order for this algorithm to work for an

Table 1: P_0 corresponding to disk speed levels in a five-speed disk.

Speed (RPM)	P_0 Range
15000	$0.00 < P_0 \leq 0.30$
11000	$0.30 < P_0 \leq 0.50$
7000	$0.50 < P_0 \leq 0.70$
3000	$0.70 < P_0 \leq 0.85$
0	$0.85 < P_0 \leq 1.0$

n-speed disk, one can set a threshold for each speed level. Essentially, as the value of P_0 decreases, the disk's operating speed should increase. We choose a threshold value of 0.7 for P_0 in our three-speed disk. In a multi-speed disk scenario, on the other hand, this threshold will be a range and not a value. But, the way we use our three-speed disk makes this modification feasible. As an example, Table 1 lists sample threshold values (as a range) for P_0 corresponding to each speed level in a five-speed disk.

We emphasize that increasing the number of operating speeds (e.g., moving from a three-speed disk to a five-speed disk) does not necessarily mean that we can save more energy. This can be seen in a manner similar to when TPM (traditional power management, which spins down the disk after a certain period of idleness) saves more energy than DRPM in the case of very long idle periods, since it can turn off the disk completely, whereas the DRPM scheme will spin down only to a nominal speed. Similarly, the three-speed disk provides enough flexibility to exploit small idle periods and also the ability to save maximum energy when possible. We note that as the idle periods grow smaller, opportunities to save power become meagre and risky. For a three-speed disk, one should not decide to spin up if spin-up time plus the request service time is more than the service time at the current disk speed. Also, One should spin down only if the idle energy consumed in the current state is more than the sum of the energy spent in spinning down and the idle energy in the lower speed state.

All the disks start off from a *normal state* where the three-speed disk is in its intermediate speed level. Once the transition probability matrix matures, we start making predictions about the future disk states. A prediction *ON* will spin up the disk by one level from its current state. A prediction *OFF* will necessarily spin down the disk to its lowest speed. The disk does not wait for a prediction to transition to the *normal state* if no disk requests were waiting. All these decisions help bring down the power consumption while minimizing the performance degradation. In the following paragraphs, we discuss how the values of various parameters employed in our approach affect the behavior of our proposed scheme.

- *Warm-up Period*: This is the period of time spent before building the initial transition probability matrix. This is an important step in getting started with making good predictions about disk accesses. The transition probability matrix built during the warm-up period will represent more of the transient behavior of disk accesses, but it eventually adapts itself to the changing workload during execution because of the regular updating of the matrix. Note that while updating the matrix, we give lower weight to the older values of the probability matrix. Deciding the right value for the warm-up period is a tradeoff between the accuracy of prediction (large value) vs the time of wait (small value) before the predictions begin. Instead of operating in either of these extremes, we can keep the warm-up period moderately short to obtain the best of energy savings and prediction accuracy. In our baseline implementation, we set it to the time taken to gather 50 samples, a value determined based on some preliminary experiments.

- *Threshold Probability*: This threshold value is used to decide which state our disk can transition to by comparing P_0 with this value. If we want to be aggressive and save more energy without caring much about the performance, then we can set it to a low value (e.g., 0.4). On the other hand, if we want to be conservative, then, say, 0.9 will be a good choice. It affects directly the prediction accuracy, which in turn can hurt both energy savings and performance. For our three-speed disk implementation, we chose this threshold to be 0.7, again based on some preliminary experiments.

- *Sampling Period*: The value of this parameter is crucial to the success of our prediction based scheme. It affects the overhead involved in the scheme, the closeness with which the transition matrix represents the workload, and the energy savings achieved. If it is chosen to be very small, the frequency of state predictions and matrix updates increases. Depending on the disk state transition times and the energy consumed during transitions, a small sampling period may or may not be beneficial. On other hand, making this period too large can lead to missing some energy saving opportunities, specifically, when the idle time is greater than the break-even time but smaller than the sampling period (there was a short duration of disk access). The length of sampling-period used in our default implementation is 12.5 seconds for a simple (two-speed) disk, 7 seconds for a three-speed disk, and 4 seconds for a five-speed disk.

Two overheads are associated with our scheme: updating probability matrix and prediction. Since each prediction scheme uses a simple operation (e.g., bitwise-OR or summation), the prediction overhead is negligible. Updating the probability matrix might have some overheads depending on the size of matrix size. In an eight-disk system, the matrix size will be 256 (2^8) by 256. Since this operation can also be done by using simple loop and the update frequency is at least tens of seconds, we believe that the overhead associated with updating matrix is also negligible.

In our experiments, we also vary the default warm-up period, threshold, and sampling period values and conduct a sensitivity analysis.

2.6 Experimental Evaluation

In this section, we first introduce our experimental setup (Section 2.6.1) and then present the results from our experiments (Section 2.6.2).

2.6.1 Setup

DiskSim [15] was used to simulate the behavior of our disk subsystem and to measure the benefits brought by our scheme. DiskSim is an accurate, highly configurable disk system simulator to support research into various aspects of storage systems. DiskSim is a trace-driven simulator, and we performed one simulation per each workload. Our simulated system has 8 disks; the specifications for the disk were provided earlier in Figure 2. We augmented DiskSim to help us carry out the experiments for various prediction algorithms discussed above to analyze how good they work in saving energy. As the simulation runs, this augmented version of DiskSim checks the state of the disk system at regular intervals. This is referred to as *sampling the system*.

DiskSim provides a synthetic workload generator used to generate the workloads with desired characteristics. Some characteristics common to all workloads are given in Table 2. We concentrated mainly on workloads with small inter-arrival times where TPM and other older techniques

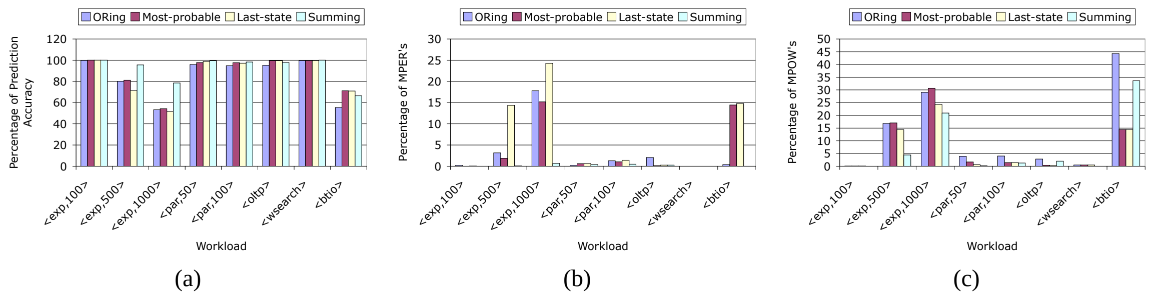


Figure 3: (a) Prediction accuracies with different schemes. (b) Contribution of mispredictions leading to performance loss (MPER). (c) Contribution of mispredictions leading to power loss (MPOW).

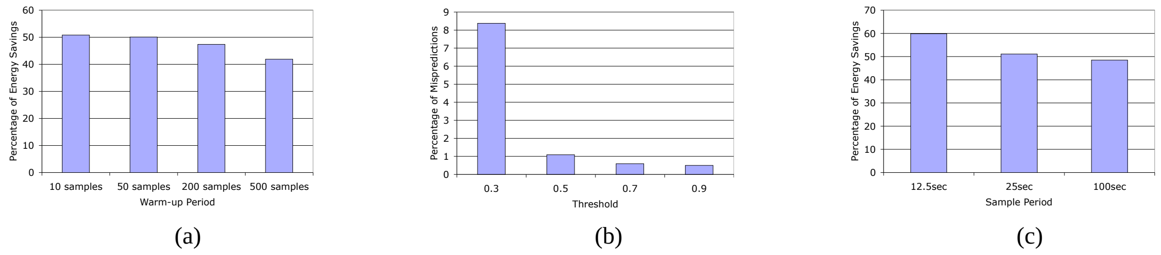


Figure 4: Effect of changing the important parameters: (a) warm-up period, (b) threshold, and (c) sample period.

have failed to save energy efficiently (that is, the high-performance workloads that exhibit short disk idle periods) and results for DRPM [17] could be compared. For the synthetic disk traces, we used two types of workloads:

- Type 1: Inter-arrival times were exponentially distributed, and
- Type 2: Inter-arrival times followed the Pareto distribution.

Type 1 workload is represented as $< exp, t >$, where t is the mean inter-arrival time in milliseconds. This type of workload models a purely Poisson process, with arrival traffic showing some kind of regularity. Type 2 workload is represented in a similar fashion as $< par, t >$, with t having the same meaning as before. This workload offers more burstiness in the traffic behavior, meaning that there exists a group of requests clustered close to each other at some places. We used synthetic workloads to show that our scheme is well adapted to different type of inter-arrival times. As far as disk power management is concerned, inter-arrival times matter most because they will eventually affect the length of disk idle periods. Thus, these two types of workloads do offer a good experimental testspace. The original version of DiskSim does not support generation of Pareto workloads.

Thus, as a part of our work, it was also augmented to generate such a workload. With these two types of workloads, we vary the mean arrival times of requests, which affects the length of the idle periods (the higher the value of t, the greater the idleness). Table 2 summarizes some default characteristics of the synthetic workloads for the request distribution across the disks.

In addition to our experiments with these synthetic traces, we performed experiments with traces extracted from real applications. These applications are *parallel* in that the number of clients issuing the requests for our 8-disk system are more than one. More specifically, the number of clients range from one to sixteen. One of the workloads is a trace from an online transaction processing application (OLTP); the other trace is gathered from a popular Web search engine. OLTP traces [57] are characterized by frequent insert/updates. The web search trace we use [57] captures the I/O traces of a system that processes web search queries. Both of these traces are obtained from a publicly available repository [57]. The I/O accesses exhibited by these applications are small, numerous, and concurrent. The results with the OLTP trace are indicated with $< oltp >$, whereas those with the search engine trace are represented by using $< wsearch >$. We also tested our scheme with a trace from a scientific application called BTIO, which is a disk-based version of a flow-solver program from the NAS Parallel Benchmarks [64]. The main operation in the code is periodic writes performed by all processors to a multidimensional array stored in a file. This trace is represented as $< btio >$. The number of clients for this type of workload was kept as sixteen. Note that the energy-saving opportunities in all these traces depend on the length of idle periods between various accesses. Specifically, the workload from the search engine was found to contain less than 2 percent overall I/O system idle time. Our experiments were carried out with these diverse (synthetic plus real) workloads to obtain statistics for the following.

- Total energy consumed by disk system when no optimization is performed (E_{tot})
- Percentage of energy savings with different power management schemes (Sav)
- Performance penalty
- Accuracy of various prediction schemes
- Effect of changing the important parameters employed in our scheme

Table 2: Default system parameters.

Parameters	Values
Request Number	100000
Number of Disks	8
Disk Size	18 GB
Sequential Access Probability	0.1
Local Access Probability	0.2
Read Access Probability	0.6
Maximum Local Distance	100 blocks

We also conducted experiments with a five-speed disk based execution scenario in order to evaluate the effect of increasing the number of speed levels in a disk. The energy savings produced with the five-speed disk are compared against those achieved with the three-speed disk and TPM. Note that all the energy saving results presented here consider the savings across all disks in our 8-disk system. The energy spent in transitioning the disk to a different state was considered in all our calculations. In the context of this work, the performance penalty of a disk system is defined as the percentage increase in the execution time for the given workload. More specifically, if the last request of the workload was serviced at time T when no energy optimization was applied and now with the optimizations it gets serviced at time $(T + x)$, the percentage performance penalty is calculated as $(x/T) * 100$. The results presented below include *all the overheads* incurred by our scheme.

2.6.2 Results

We conducted experiments to test and validate the three-speed disk model along with the prediction schemes and verify the usefulness of Markov modeling. First, the prediction algorithms described earlier were evaluated for their prediction accuracies. Specifically, we tested each prediction scheme on all the workload types we have. Figure 10(a) shows the prediction accuracies of the four schemes discussed earlier. The average prediction accuracies (when all workloads are considered) are 86.0%, 84.2%, 87.6%, and 92.0% for the Last-state, ORing, Most-probable, and Summing schemes, respectively. Since inaccurate prediction of disk idleness can be determined from a performance perspective, we consider 90% or higher as a good accuracy, and our prediction accuracies are in this range. The total mispredictions (TMPs) can be broken down into two types:

- Mispredictions leading to performance loss (MPER), and

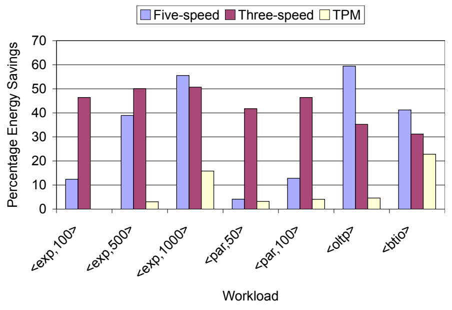


Figure 5: Comparison of energy savings achieved by five- and three-speed disk based systems relative to the base case, namely, TPM.

- Mispredictions leading to energy loss (MPOW).

MPER happens when one predicts a spin-down for the disk but the disk was actually accessed and hence we incur spin-up delays. In comparison, *MPOW* happens when the disk is predicted *ON* but it was never accessed during that period, and consequently, an opportunity to spin-down was missed. We see from Figures 10(b) and 10(c) that the ORing technique gives more mispredictions leading to energy loss, whereas the Last-state technique gives more mispredictions leading to performance loss. Overall, our defended prediction scheme (Summing) performs better in all respects. Although all these schemes do provide a good percentage of correct predictions, the Summing scheme has significantly lower *MPER* value. It is also clear from these results that all the prediction schemes tend to become less accurate as the sampling period increases, specifically the Last-state scheme. In cases where even a slight performance degradation is intolerable, one should try to minimize the percentage of the *MPER* even if, in doing so, we increase the contribution of *MPOW*. Note that a higher *MPOW* value only means that we missed some energy saving opportunities, but a higher $MPER$ value may be intolerable in a high-performance computing environment.

There should be enough samples to build the transition probability matrix initially so it really does reflect the workload characteristics with a reasonable accuracy. Hence we decided to take at least 50 samples to capture the workload behavior. Obviously, the more samples we take, the better our knowledge of the workload. However, this also means we start the energy optimizations late. Figure 4(a) shows that the energy savings decrease when the warm-up period is increased. Figure 4(b) shows the effect of varying the threshold value on the percentage of mispredictions leading to performance loss (*MPER*). Decreasing the threshold value means that we aggressively turn disks

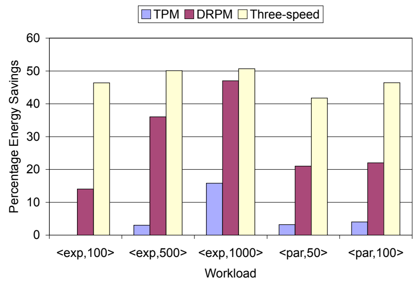


Figure 6: Comparison of energy savings with different schemes including results with DRPM. Results for DRPM were obtained from [12], where no tests were performed with real traces.

Table 3: Percentage of performance penalty.

Workload	Penalty (TPM)	Penalty (Three-Speed Disk)
$< exp, 100 >$	0.0	0.0
$< exp, 500 >$	0.03	0.0
$< exp, 1000 >$	0.015	0.0
$< par, 50 >$	0.0	0.0
$< par, 100 >$	0.0	0.0
$< oltp >$	1.42	1.76
$< btio >$	0.0	0.0

OFF and therefore increase the chances of mispredictions, which is reflected in Figure 4(b). In Figure 4(c), on the other hand, the effect of increasing the length of the sampling period is shown. The energy savings decrease because we miss some idle time opportunities. We also tested the effectiveness of these prediction schemes using a five-speed disk. The results given in Figure 5 indicate that three-speed disk provides better energy savings in most cases. The response of a five-speed disk to a disk state prediction is more gradual than that of the three-speed disk. The reason is that the five-speed disk slows down the disk speed one step at a time unless a disk experiences big slowdown in the response time. Consequently, it takes more time to transition to a lowest power mode, in turn producing less savings. This one step approach is a bit less aggressive in lowering the disk speed, but it enables us to identify the system state at all times and ensures easy recovery on misprediction (there are forced spin-ups and spin-downs when the actual state is not equal to the current state of the system). Note that one has more flexibility with a five-speed disk when it comes to selecting a speed level, which can be helpful, as is the case for the savings on OLTP and BTIO workloads in Figure 5. Although we achieve better energy savings with a five-speed disk, it also leads to more performance penalty (not shown in results because of lack of space). This can be attributed to the increased overhead of transitioning across different speed levels.

In Figure 6, the energy savings obtained with the three-speed disk supported by our scheme are compared with TPM and DRPM savings. All energy savings are normalized with respect to the base case, where no power saving scheme is employed. The energy consumption evaluated and the power saving results consider the entire disk system. We regenerate the energy savings with DRPM (denoted as $DRPM_{perf}$ in [17]) where it can predict the idle times with full accuracy; consequently, there is no performance loss. Since we are using the same simulation tool for generating the same workload types, it makes sense to compare the results. We see from these results that our scheme provides more energy savings compared to TPM. It also does better than DRPM. The reason can be attributed to the ability of the disk to totally spin down (standby mode) whenever possible, and save energy even when the duration of idle periods is not sufficiently long (spin at an intermediate speed level). Although the opportunity to save energy with these workloads may look meagre, it is the result of using the predictive scheme along with the concept of a multi-speed disk that helps save energy. There is not much performance penalty from TPM as this scheme triggers a shutdown only when the disk has been idle for a long period of time. However, when we use prediction algorithms and perform spin-ups and spin-downs proactively, there is a chance of significant performance penalty. This can be a result of a mispredicted spin-down (MPER) when the disk is being actually accessed. Table 3 shows that, with our scheme, there is very small or no performance penalty with the used traces. Gurumurthi et al. [17] give performance degradation in terms of response times, but does not show the net effect on the total execution time.

2.7 Concluding Remarks

The main contribution of this paper is a novel Markov model based disk idleness prediction scheme that can be used for reducing disk power consumption when used with a three-speed disk. The paper explains in detail why the defended prediction mechanism is better than others and why it saves disk power. To evaluate the effectiveness of our approach, we implemented it using DiskSim and performed experiments with both synthetic traces and real application traces.

Our experimental results show that (i) the prediction accuracies of the proposed scheme are very good (87.5% on average); (ii) it generates significant energy savings over the traditional power saving method of spinning down the disk when idle (35.5% on average); (iii) it performs better than a previously proposed multi-speed disk management scheme (19% on average); and (iv) the performance penalty it brings is negligible (less than 1% on average). Overall, our implementation and experimental evaluation demonstrate the feasibility of a Markov model based approach to saving disk power. Our ongoing work involves integrating this scheme with existing disk power saving strategies and testing them under different workloads. We are also investigating whether high-level (application level) information supplied by programmers can be used for improving our power savings.

Chapter 3

Power Aware Disk Allocation

Disk power consumption is one of the major concerns in adopting applications with large scale I/O in both mobile and scientific computing domains. Virtualization and the resulting abstraction of large scale storage systems and variety in the I/O demands of applications call for a power efficient disk allocation strategy across simultaneously executing applications that provides necessary performance guarantees. In order to abstract the underlying diversity in capacities and rotation speeds of disks and attain a performance and power efficient allocation, a disk allocation algorithm has to often choose from a set of conflicting optimization criteria. This paper presents the trade-offs associated with power and performance across different disk allocation schemes, targeting a scenario where multiple applications exercise the same disk storage system at the same time. We also present a novel disk allocation scheme that reduces overall power consumption of a disk system while satisfying the performance and storage capacity constraints set by applications. Extensive analysis of our proposed disk allocation scheme shows that it reduces disk power consumption compared to other alternate disk allocation schemes, while providing similar or better performance guarantees.

3.1 Introduction

An important characteristic of many high performance applications is disk intensiveness. Many scientific simulation codes for example have frequent disk accesses. In fact, large-scale simulations have become an integral, important, and in many cases, primary approach to solving complex science and engineering problems and automating scientific knowledge discovery. These simulations tend to generate huge amounts of data that must be stored on disks, mined, analyzed, evaluated, check-pointed (on disks), and steered, in most cases dynamically during the course of execution. Similarly, many data base/mining applications frequently exercise disk systems of parallel architectures. In a disk storage system shared by multiple applications, one of the critical problems is resource (disk) allocation across these applications. The main characteristic that makes this problem challenging is the fact that different applications demand different amounts of disk space (capacity) and can tolerate different disk latencies. As a result, allocating disks from a heterogeneous disk pool to satisfy the needs of all applications using the same storage system is non trivial. Existence of additional constraints such as power and reliability makes this disk allocation problem even harder.

Disk power consumption poses severe challenges in terms of electricity costs, overall system design and reliability. Power consumption can put a limit on how much designers can push performance, as power dissipation generates heat that affects component stability and reliability, especially for large server systems [17]. While recent research has focused on hiding most performance bottlenecks by overlapping computation with disk I/O [38, 43], power bottlenecks cannot be hidden. Note that both cost and reliability are very important for large scale server systems in general and disk storage systems in particular. Disk power consumption can be high in systems that execute large data-intensive scientific applications, e.g., those from the domain of astrophysics, genome research, computational chemistry, and nuclear simulation. In fact, a recent research [22] shows that disk storage can be responsible from up to 27% of total system power consumed by data centers.

Motivated by these observations, this paper presents several disk allocation schemes and experimentally evaluates them using multiple metrics. One of these schemes is defended in this paper and performs disk allocation across multiple, concurrently running applications such that overall disk power consumption (of the underlying storage system) is reduced and performance and storage constraints specified by applications are satisfied. At a high level, one can see our disk allocator as a *virtualization layer* that maps virtual disk requests to physical disks in the storage system such that the application needs are satisfied.

We implemented our disk allocation schemes and tested their effectiveness under a wide range of execution scenarios, which involve different resource (disk) pool sizes, different sets of disks (with different disk speeds), and different disk request arrival patterns. We can summarize our results as follows. Our defended scheme performs consistently better than other schemes tested when multiple metrics of interest are consolidated. Specifically, it reduces power consumption over a performance-only scheme by 20%, while improving performance over a power-only scheme by almost 23%. This paper also discusses the results from our sensitivity study in which we change several storage system characteristics. Our experimental data for this sensitivity study suggests that proposed power-aware disk allocation scheme is robust under varying values of major experimental parameters.

The rest of this paper is organized as follows. Section 3.2 presents a description of relevant efforts in solving similar or related problems followed by the precise problem addressed in this paper in Section 3.3. A comprehensive description of design parameters considered in this study including types of resource pools, metrics of interest, format of application request and various disk allocation schemes are presented in Section 3.4. Section 4.6 describes the experimental setup, results and the sensitivity of our scheme to various design parameters, and finally, our conclusions are presented in Section **??**.

3.2 Related Work

Significant previous work exists on power aware disk management in both mobile and server domains. The prior studies have ranged over various layers of the system including the hardware and software managed schemes. Spinning down disks [13, 14, 20] or reducing the speeds of the disk when it is used sparsely is a popular technique for disk power management. Simunic et al. [59] proposed to use a semi-Markov decision process to obtain optimal power management policy for laptop hard disks with a system model that can handle non-exponential inter-arrival times in the idle and the sleep states. An alternative hardware based strategy is DRPM [17, 7] which involves dynamic modulation of disk rotation speeds. Zedlewski et al. [69] extended a disk simulator to examine the energy consumption behavior of the hard disk drive.

The interval between the successive disk I/O requests is crucial for effective power management of disks. Therefore, several studies attempted to increase these intervals via postponing dispatch of I/O requests [63, 41]. Son et al. [54] proposed an energy-aware data prefetching scheme for multi-speed disks, which is compiler driven. In comparison, Okada et al. [39] developed a hard disk drive which can operate at different speeds. They used two RPM modes: file download/upload and audio/video application. Rao et al. [47] found the optimal revolution speed and length of standby period for a certain playback rate.

The power management issue has also drawn significant attention in the domain of server class storage systems. [8, 7] exploited the multi-mode disk drive technology in massive scale storage systems These studies proposed to monitor the I/O queue length and to adjust the disk revolution speed subject to I/O queue length. Pinheiro et al. [45] suggested to place the frequently used files into a small number of disks so that the hot spot disks are in active mode while other disks are maintained at low power mode. A number of studies exploited the memory hierarchy of the I/O subsystem, i.e., via intelligent caching, for reducing energy consumption of hard disk based storage subsystem [31].

Application-level optimizations for reduced power consumption in the disk have also been studied. The approach proposed in [26] restructures a given application code considering the disk layouts of the datasets it manipulates. Son et al [55] presented compiler analysis to extract disk access patterns and use this information to insert explicit disk power management calls at appropriate places in the program code. Use of feedback directed adaptive resource control has been studied in the literature for disk space allocation [40], throttling the storage access requests to ensure system throughput is shared fairly [28].

Providing performance guarantees in distributed storage systems is more complex because clients may have different data layouts and access their data through different coordinators (access nodes), yet the performance guarantees required are global. In this context, Wang and Merchant [61] presented an adaptation of fair queuing algorithms for distributed servers that enforces an extra delay (possibly zero) that corresponds to the amount of service the client gets on other servers. Several recent efforts [4, 11, 10, 27] have also considered the use of feedback control theory for handling performance specifications.

Many of these prior efforts have concentrated on manipulating the disk's rotational speed or modifying the occurrence and duration of disk idle periods for power management. We attack the power management problem from a different angle by considering it as a *resource allocation prob-*

lem, where the aim is to reduce overall power needs of the I/O system in presence of performance constraints coming from different applications concurrently exercising the same disk storage system.

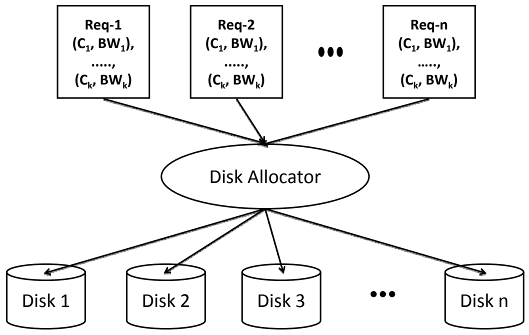
3.3 Problem Statement

This paper presents a novel approach of allocating disks from a resource pool of disks with different rotational speeds (RPMs). The proposed disk allocation scheme strives to reduce disk power consumption of an I/O system, while providing the necessary performance guarantees as requested by the applications. The incoming applications issue disk requests specifying their needs from the I/O subsystem in terms of storage capacity (disk space needed) and performance (minimum bandwidth).

Given a specific resource pool with a large number of disks with different RPMs, and an application workload's consolidated set of requirements (a set of disk requests), our approach attempts to reduce the power consumption of disks while providing the performance guarantees. Applications state their I/O requests in terms of disk requirements which are fed to the *Disk Allocator*, that helps achieving the performance goals expected by the application user while making the best use of the available storage resource(minimizing power). The general architecture of our approach is presented in Figure 13(a). At any given instant in time, an arbitrary number of disk requests may demand allocation from the available set of disks with a storage constraint in terms of disk capacity needed (C) and a performance constraint in terms of bandwidth (BW). In this paper, we use the term 'demand' to indicate the set of disk requests made by an application that uses our I/O system.

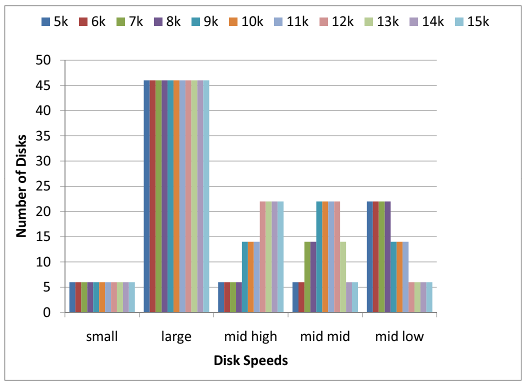
3.4 Our Approach

While building a large storage system, one can only estimate the amount of storage required, based on which the type (various speeds) and number of disks are decided. This means that there can be no storage system which will be optimal at all times, once it is built. Here, optimality can be associated with many factors: power usage, performance, etc. Our goal, given a storage system, allocate the available set of disks to the applications in a way that guarantees performance and reduces the power consumption. In the following subsections, we discuss the major components of our approach.



Req-1 (C1, BW1), ..., (Ck, BWk)
Req-2 (C1, BW1), ..., (Ck, BWk)
Req-n (C1, BW1), ..., (Ck, BWk)
Disk Allocator
Disk 1
Disk 2
Disk 3
Disk n

(a)



(b)

Figure 7: (a) Architecture of our power aware disk allocation scheme with each request specifying the requirement in terms of capacity (C) and bandwidth (BW) needed from the disk. (b) Composition of different resource pools. This graph shows the number of disks of different speeds in each type of resource (disk) pool.

3.4.1 Resource Pool

A resource pool in case of our storage system is a collection of disks. Since there is no fixed pool which is good for all situations, we try to look at various types of resource pools, when the different allocation schemes are used. For our experiments, we consider five types of pools, which are different from each other in terms of the total number of disks or the number of disks for a particular speed. All the disks considered for building the resource pool have a fixed capacity of 10GB, while the speeds (RPM) range from 5k to 15k. Note that the speed range is not for a single disk, but these are ranges of speeds available in the resource pool. Composition of our disk pools in terms of the number of disks with different speeds is presented in Figure 13(b).

● *Small Pool*: The number of available disks in the resource pool gives a good measure of its power costs, availability, ease of administration, etc. Our intuition behind deciding the pool size as small is to use the number of demands[1] that will be using this resource pool as a metric for defining its size. For example, if the number of demands is X and the maximum number of disks each demand may request is Y, then a small pool can be one in which the number of disks is a fraction of X*Y. Note that in all types of resource pools that we will be considering, all the disks (with different speeds) will have the same fixed capacity. The distribution of disk speeds is uniform in this type of

[1]Recall that we use the term ’demand’ to indicate the set of disk requests made by an application.

pool, meaning that the number of disks for each speed will be same.

• *Large Pool*: This pool is similar to the small pool described above in organization and characteristics, except that the number of disks is greater. This pool is considered to observe the effect when the resources available are almost infinite, meaning that we have more than the required resources. Again, if X*Y is the total number of requests predicted, then having a multiple of X*Y disks in the resource pool can be described as a large pool.

• *Pool Skewed to High Speeds*: This and the following pools have reasonable number of disks (mid-size) which is neither small nor large. Our goal here is to measure the effect when the size of the pool is moderate, but the disk speed distribution is non-uniform. Specifically, the number of the disks of high speeds is more than the number of the disks of moderate or low speeds. Since the number of disks in the system remains the same, we reduce the number of disks with low speeds when increasing the number of high speed disks and keep the number of disks with moderate speed to be the same.

• *Pool Skewed to Mid Speeds*: This is again a mid-sized pool in the number of disks it contains, as compared to the number of disk requests that might be generated, difference being the disks with moderate speeds are more in number as compared to other disk speeds in the pool.

• *Pool Skewed to Low Speeds*: Similarly, this pool is skewed to the low speed disks with them more in number as compared to other disk speeds in the pool.

These different types of disk pools should give us a good picture of what happens when various disk allocation schemes are applied to service competing application's storage requests.

3.4.2 Request Format

The disk allocation schemes presented in the paper are all runtime schemes, that is they allocate/deallocate the disks as the applications come in and leave the system. Each application has a start time and an end time. As the applications come in, they provide information to the resource pool regarding their disk requirements in terms of the capacity (C) and the bandwidth (BW). Each application might ask for more than one type of disk. It is possible that at the time when the application leaves the system, the disks that were allocated for that application may be reclaimed for further use. For our basic experiments, we will consider that the data used/generated by the application was persistent and hence those disks cannot be reclaimed. However, in our sensitivity analysis, we will also consider reclaiming of the disks. For the purpose of our experiments, we generate random start and end times for different applications. We will be assuming that the maximum number of requests generated by a particular application does not exceed five (though this parameter can assume any other value). For the disk allocation schemes evaluated in this work, it is not mandatory to assign a new disk for every request of the application. This means that, if there is a disk that can service more than one request, one is allowed to use the disk in such a manner. It should be clear that, under this execution model, an application demands certain performance level and storage capacity from the storage system. This demand is an aggregation of requests, which in turn are not actual disk I/O requests, but specifications of a disk that might be used to service that application. In our setting, each application has a demand which consists of at most five such disk specifications (which might be stated as disk requests from this point on). This again is a parameter that can be changed if desired.

Since the disk requirement is specified in terms of space and bandwidth, it gives us an option of using more than one disk for fulfilling such a request. This means in practice that we can derive the bandwidth requirement from multiple disks, making the disk allocation across applications an even more interesting problem. For all our experiments, we will keep a bound on the number of disks that might be used to share the load (at most five). When splitting bandwidth, the ratio in which the capacity is split will be proportional to the amount of bandwidth and vice versa.

3.4.3 Metrics of Interest

Disk allocation needs to be done in a way such that it helps reduce the power consumption of the disk storage system, improves availability of resources, and ensure certain performance guarantees to the applications. All the schemes that we present here guarantee some level of performance to the applications (if the allocation is granted) in the sense that if the application requests some bandwidth, then the disks allocated will be able to provide at least that amount. This way, we do guarantee the performance, but it can be easily seen that, there are a lot of strategies that can give different disk assignments, hence producing different set of performance or power values. We will look at all these metrics and schemes, targeting specifically each one of them taken individually. It can be easily seen that when all of these metrics are considered simultaneously for generating a disk allocation, one decision might not confirm with the other (e.g., a performance aware allocation may not be power aware or vice versa). Also, these metrics are usually not very clearly defined and their measurement varies depending on how one defines them. Let us try to define these metrics and the way we calculate their values for our experiments.

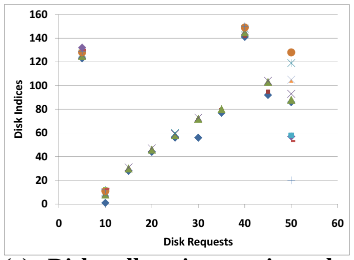
• *Performance*: One thing we ensure in all allocations is that a request will always be allocated a disk with higher than or equal to requested bandwidth (BW). In order to provide a relative measure of performance for the various disk allocation schemes, we consider the extra bandwidth available for the allocated requests. As mentioned earlier, a single disk might be servicing more than one request. In such scenarios, merely providing the bandwidth demanded by the request, might not ensure the desired performance. This can be attributed to the degradation offered by the other workloads present on that disk. Keeping this in mind, there can be a possible degradation in the overall performance of applications, if such scenarios are not handled carefully. We measure these degradations or enhancements by using a parameter from our power aware scheme (explained shortly) called ”buffer”. To capture the effect of more than one workload on a disk, we say that some ”buffer” amount of bandwidth is wasted over the bandwidth already used by the workload. As the number of workloads increases, so does the wasted bandwidth. Hence, if the number of workloads present on a disk is N, then the wasted bandwidth is buffer $\times$ N. The spare bandwidth's on active disks are summed up over all the serviced requests and then the effect of wasted bandwidth is evened out.

• *Power*: There are two mechanical components to disk power, namely the spinning of the platters and the head movement (seeks). As pointed out previously in [17], spindle motor is responsible for nearly 50% of overall idle power and the number is close to 82% in server class disks with ten platters. Power consumed by a disk is proportional to the square of the speed of its rotation (based on the model suggested by [17]). We use this model to compute the total power consumption of the storage system when various disk allocation schemes are used. In order to compute the total power, all the active disks are identified, their speeds are squared and summed. The goal behind our power–aware scheme is to reduce the value of this metric as much as possible.

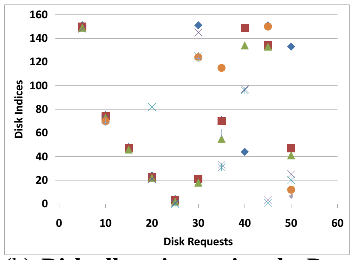
• *Availability*: Clearly, in order to satisfy a request, we must have enough capacity and the required bandwidth left on disks. The availability criteria shows how good an allocation scheme is in maximizing the number of serviced requests. It is not clear if the quality of service may be ignored when marking the request as serviced or unserviced. For our experiments, we do not look at the quality of service to qualify the allocation as serviced/unseviced request. The higher the number of unserviced requests, the weaker the scheme is in terms of availability. We simply sum up all the unserviced requests at various phases during the execution to build our availability graphs.

3.4.4 Evaluated Schemes

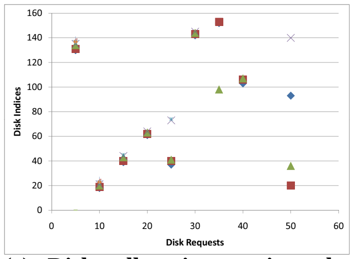
We now describe the disk allocation strategies evaluated in this work. We assume that all the disks in the resource pool are initially idle and become active only when allocated to a particular application. Disks can also become idle once all the workloads they were servicing have left/finished.



(a) **Disk allocations using the Power Only scheme.**



(b) **Disk allocations using the Performance Only scheme.**



(c) **Disk allocations using the Power Aware scheme.**

Figure 8: Variations in disk allocations from the resource pool to various disk requests by different schemes. Disks are numbered from 0 to N in the resource pool. Y-axis depicts the disk number selected from the resource pool for servicing a disk request whose number is given by X-axis.

● *Power Only*: The sole aim of this disk allocation scheme is to reduce the power consumption of the storage system, without considering the effect it might have on other metrics of interest. The intuition will be to activate the least number of disks such that their rotational speeds are also the lowest. This will at least ensure reducing the power consumption, a parameter dependent on square of the disk's speed of rotation [17]. The conversion from RPM to bandwidth they can offer is straightforward [44][2]. For the power only scheme, the minimum speed disk from the resource pool is selected. Note here that the selected disk might be already active or it may be idle at the time of

[2]User data transfer rate in MB/s = RPM/60*sectors per track*512*8/1,000,000

allocation. Changing the state from idle to active is a one time cost, and its effect on performance is typically insignificant. Allocation of more than one request on a disk might hinder the performance, but the primary goal of this scheme is to minimize the power consumption.

• *Performance Only*: An application when given bandwidth more than its minimum requirement, might finish early, hence creating opportunity to reclaim some of the disks it was using. In this scheme, the highest speed available is allocated first. In doing so, we hope that each request will have the most spare bandwidth available, helping it to improve its performance.

• *Power Aware (Our Defended Scheme)* : In this scheme, we try to maintain a balance amongst the various metrics while providing the performance guarantees, saving power being the priority. The disk allocation returned from this scheme tries to bring the best from the previously-described schemes. This means that it tries to improve performance, be good in terms of availability, and reduces the power demand. One aspect that we ignored in previous two schemes is the effect on achieved performance in presence of more than one workload per disk. The result is lack of reliability in terms of disk's performance. In order to ensure the application with the demanded performance, we introduce a degradation term ”buffer”. *Buffer* is the bandwidth wasted on a disk due to the presence of a workload. The value of buffer will be constant for all workloads. Hence, as the number of workloads increases, the wasted bandwidth increases. The net result is, when allocating disk to a new workload, the available bandwidth is computed taking into consideration the degradation due to already present workloads on the disk. By doing so, we ensure that even in the presence of degradation, the application's performance will not suffer. In order to keep the power low, we use the same strategy used by the power only scheme, and select minimum speed disks as much as possible. Interestingly, by doing so, we indirectly make the availability good too. Since we are consuming the low speed disks earlier on, servicing the lowest bandwidth requirement first, it helps in keeping the disk fragmentation low (we measured the total unused bandwidth on all the active disks), while reserving the high speed disks for requests with high bandwidth requirements. Figure 9 gives the pseudo code for this allocation scheme. The complexity of this scheme is O(n log n), where n is the total number of I/O requests generated by all the applications.

```
main () {
 do {
  for each application
   sort disk requests in ascending order of speed requirement;
   for each request
     allocate a minimum speed disk with the required capacity;
     update disk capacity and available bandwidth;
}}
```

Figure 9: Outline of our algorithm for power aware disk allocation.

Table 4: Major experimental parameters.

Parameter	Value
Number of requests	50
Disk requests per request	2-5
Capacity requested	1-10GB
Bandwidth requested	15-65Mb/s
Resource Pool size	(Small, 66 disks) (Mid-size, 154 disks) (Large, 242 disks)
Type of disks	Single Speed
Capacity of each disk	10GB
Disk Speeds	5-15K

Table 5: Disk indices and their speeds for mid-size pool skewed for mid-speed disks.

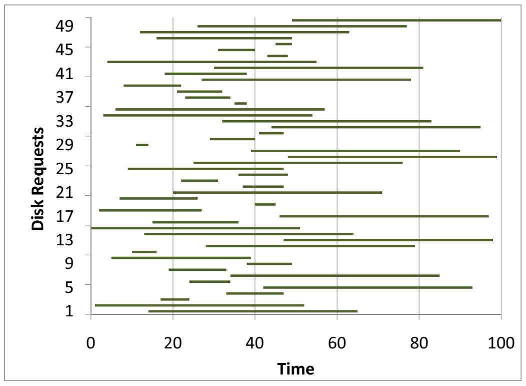
Disk Indices	Speeds
D1-D22	9K
D23-D44	10K
D45-D66	11K
D67-D88	12K
D89-D94	5K
D95-D100	6K
D101-D106	14K
D107-D112	15K
D113-D126	7K
D127-D140	8K
D141-D154	13K

3.5 Experimental Evaluation

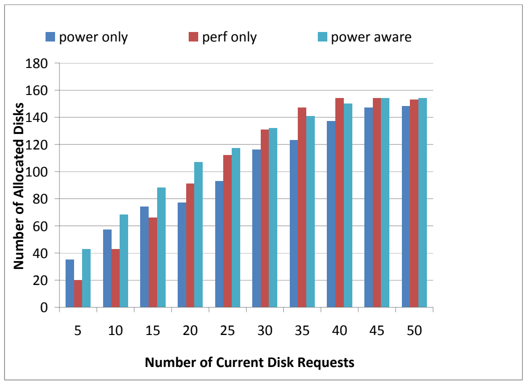
The experiments were conducted using a simulation infrastructure we developed in C++ which could perform disk allocations as the applications enter and exit the system providing their I/O requests. This simulator can be thought of as a part of the operating system which can communicate with the block I/O device controller to get the required information about the composition of the storage system and the I/O requests generated by the applications. It can also be employed as a part of a storage virtualization layer such as [1] and [2]. The virtualization software has all the necessary information about the physical storage lying underneath and the allocation decisions can easily be guided by our power aware scheme.

3.5.1 Setup

The main components of our experimental setup consists of the resource pool and the application request stream. The applications running on different clients come in and exit the system which includes the client machines and the storage system. The storage system (referred to as resource pool) consists of a variety of disks which get allocated to different applications' demands as per the different disk allocation schemes. The range of values for all our experimental parameters are described in Table 6. Table 7 provides the correspondence among the disk indices and their respective speeds for mid-size pool skewed for mid-speed disks. Note that, this correspondence among the disk indices and their speeds is different for other types of pool (see Figure 13(b) for a general description of composition of each of different resource pools in terms of the disk speeds). Specifically, the mid-sized pool with disk speeds skewed for mid-speeds depicted in Table 7 is used for our main set of experiments comparing the benefits from different schemes. The rest of the pool types will be used as part of our sensitivity analysis.



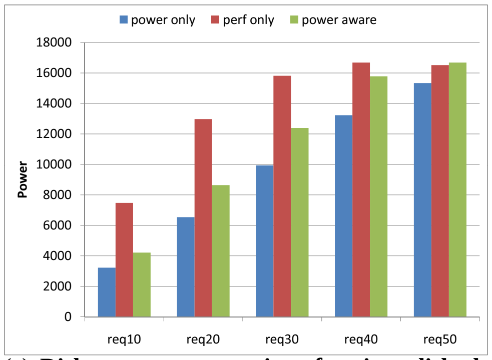
(a)



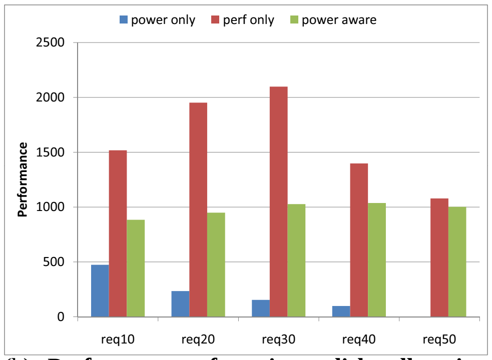
(b)

Figure 10: (a) A possible disk request arrival distribution and their execution times in our default setting. (b) Cumulative number of disks assigned by various disk allocation schemes. Values on horizontal axis represent the number of requests made so far.

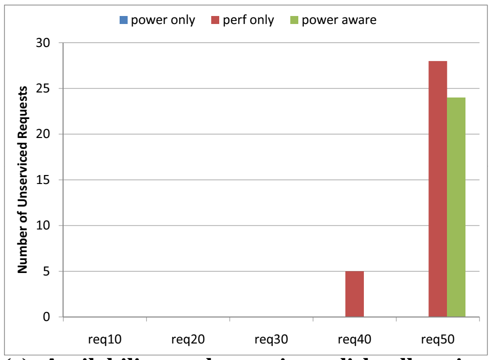
The application request stream is not fixed and Figure 10(a) provides a possible sequence of applications' arrival order and exiting times, usable as input to the experiments. The relative arrival order is important for the disk assignment, as the disks get allocated in terms of the arrival order. On the other hand, the application exiting time is important to free/reuse the disks assigned to the applications for further use (if the data on those disks was impersistent). Note that, if all the applications using a particular disk exit the system, that disk is spun-down totally until further assignment.



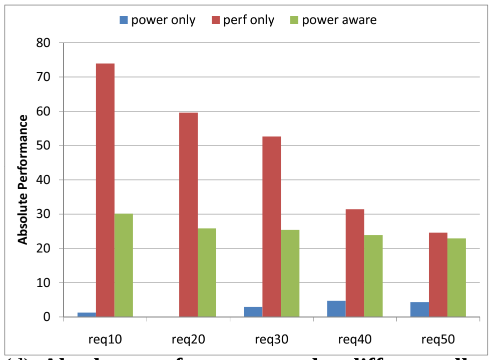
(a) **Disk power consumption of various disk allocation schemes.**



(b) **Performance of various disk allocation schemes.**



(c) **Availability under various disk allocation schemes.**



(d) **Absolute performance under different allocation schemes.**

Figure 11: Study of different metrics with respect to various disk allocation schemes.

3.5.2 Results

We performed experiments with all the disk allocation schemes explained above in order to evaluate each of their potentials. The input to the experiments was a resource pool consisting of disks with different speeds and a set of application requests. The format of the application requests has been described earlier. We varied the size of the resource pool (keeping the number of disks for each speed and the disk speeds available to be uniform) and also the type of disks within the pool. The output of the schemes produces a disk allocation for the demands laid by applications, and is evaluated on the basis of three parameters explained earlier: performance, power, and availability.

We now show the disk allocations by various schemes proposed individually over a set of fifty demands. The timing characteristics, namely, arrival order and processing time of the requests captured by these demands in our default setting are as shown in Figure 10(a). The indices of disks allocated from the resource pool by each of the schemes, namely, Power Only, Performance Only and Power Aware schemes, are depicted in Figures 8 (a), (b), and (c) respectively. The cumulative number of disks allocated by the individual schemes are shown in Figure 10(b). Although the cumulative number of disks used by each scheme look similar, it does not provide an exact picture of the power consumption or the performance offered.

The number of unserviced requests in each of the schemes (the inverse of availability) is depicted in Figure 11 (c). As seen from this figure, the performance only scheme has the highest number of unserviced requests. Note also that the power aware scheme performs better in terms of availability with lower number of unserviced requests, at the same time keeping the performance at par with the performance only scheme. This is due to allocation of the lowest speed disks for each request that is performed by the power aware scheme which retains the high speed disks until they are necessary. In terms of power, the power aware scheme lies mid way between the extremes of power only and performance only. Towards the end, the power aware scheme has the highest power consumption because of the higher speed disks chosen and the large number of disks used, as a consequence of servicing of more requests and taking care of the performance degradation.

It can be seen from these results (Figures 11 (a) and (b)), power only and performance only schemes perform optimally (as can be expected) with respect to power and performance metric, respectively. We also see that our power aware scheme performs very well with respect to power and fares well with respect to performance, and it also has higher availability (lesser number of un-serviced requests). Specifically, the power aware scheme consumes approximately 9% more power than power only scheme when the number of disks requested is fifty. On the other hand, power aware scheme performs quite similarly to the performance only scheme with a slight depreciation for the same case.

Figure 11 (d) gives the best description in terms of how the allocations schemes fare in terms of providing performance guarantees to the application demands. As is evident from the figure, the performance only scheme does very well when the number of requests in the system is low, as each request goes to a new disk, thus leaving a lot of spare bandwidth as compared to the bandwidth utilized. In order to compute this, we used the value of spare bandwidth computed to measure performance values in Figure 11 (a), and divided it by the total bandwidth allocated to all the serviced requests. The value generated gives a measure of the change in performance due to the spare bandwidth. Note that all the performance values were normalized by referencing with the lowest performance value. This was necessary because for the power only scheme, there was in fact degradation in performance which resulted in negative values of spare bandwidth. This means that for the power only scheme, the workloads suffer in terms of performance because of the presence of more than 1 workload per disk. The power values in all graphs are also normalized to the power consumption of a disk at unit speed.

3.5.3 Sensitivity Analysis

We performed various sensitivity experiments using our power aware disk allocation scheme. Due to the space constraints, we are not presenting the related graphs for these experiments; instead, we only summarize our major observations.

Firstly, we varied the way applications data is handled (in terms of persistency) during the course of their execution and once they finish. Data which is non-persistent (not required once the application finishes) uses up the disk space which can be reclaimed. Identification and reclaiming of such disks can help provide more allocation options for the incoming applications. We experimented with partial reclaim strategy which reclaims about 50% of this volatile disk capacity to be allocated to the fresh requests. The results obtained indicate that the relative behavior of our power aware disk allocation scheme is not highly sensitive to the reclaim strategy.

We then varied the constitution of the resource pool mentioned in Section 3.4. The results indicated that, when the number of applications in the system was not high (approximately 10 in our experiments), the power was less for large pool as compared to the small one. This is because, in a large pool there are more options (in terms of availability of low speed disks) to choose from. But, as the number of the applications increase, the power usage for large pool exceeds that of the small pool. For the three variants of mid-size pool, the pool that was skewed towards low speeds had the least power consumption, whereas the one skewed for high speeds had the highest. This is intuitively due to the dependency of power consumption on disk speeds.

Thirdly, we varied the application request pattern that is fed as input to the power aware scheme. Four different access patterns were randomly generated with four distinct randomly generated seeds.

It was found that the scheme is quite robust. For a fixed resource pool, an application request pattern with more requests consumed more power and performed relatively poor as compared to an a request pattern with less number of requests.

Finally, we varied the the amount of degradation offered by the presence of multiple workloads on a single disk. We tested for three values, when buffer value was 1,3, or 5 Mb/s. As one can expect, the higher the value of the buffer used, the higher performance one achieves while at the same time sacrificing on power and availability. This was backed up by the results we obtained where the performance was highest for buffer equal to 5. Deciding the right value of buffer is a tricky question and may depend on the metric one is looking to improve.

3.6 Conclusion

One of the critical problems in managing a complex storage system is the allocation of available disks across competing applications. The main characteristic that makes this problem challenging is the fact that different applications demand different amount of disk space and can tolerate different disk latencies. As a result, allocating disks for a disk pool that contains multiple heterogenous disks is not trivial. Providing performance guarantees considering additional constraints such as power and reliability makes the problem even harder. In order to abstract the underlying variety in capacities and rotation speeds of disks and provide performance and power efficient allocation, a disk allocation algorithm was proposed in this paper that considers various trade-offs associated with power, performance and availability in disk allocation schemes. A novel disk allocation scheme that reduces overall power consumption of a disk system while satisfying performance constraints of individual requests was also presented. Extensive analysis of our disk allocation scheme with a large number of applications shows that it reduces disk power consumption compared to other traditional disk allocation schemes, while providing similar performance guarantees. Our scheme performs consistently better than multiple schemes when multiple metrics of interest are consolidated. Specifically, it reduces power consumption over performance only scheme by 33.4% while improves performance over power only scheme by almost 67.1%.

Chapter 4

Dynamic Storage Cache Partitioning Using Feedback Control Theory

In this paper, we propose a new quality-of-service (QoS) aware storage cache partitioning scheme that dynamically partitions cache space amongst simultaneously running I/O-intensive applications. The QoS specification is given in terms of latency of data access. Apart from data access latency which is perceived by the user, there can be more constraints put forth from the system administrators point of view. One of the more important constraints is the disk utilization level in a storage system. Normally, higher disk utilization levels are preferred by the system administrators for better resource consolidation. In this paper, we focus on these two constraints, namely, data access latency and disk utilization, which are translated into an overall storage cache hit rate requirement. We employ feedback control theory to achieve the required hit rate target per application. Our experimental results indicate that the proposed storage cache partitioning scheme is able to meet the required storage cache hit rate targets and improve overall storage system performance.

4.1 Introduction

Present day computing makes use of resource sharing for improving resource utilization. One of the implications of resource sharing for the user is lack of performance guarantees. We focus our attention on the disk sub-system where one disk may serve requests from more than one application. Similarly, the storage cache (implemented in memory) is usually shared by the simultaneously running applications. With the decrease in manufacturing cost, computers today have larger memories and thus have larger storage caches. These storage caches are typically shared by applications simultaneously running on the same system. These applications may interact in several ways, sometimes improving the overall performance of the system, as in the case of, multitasking which can help increase CPU utilization. However, not all shared resources can be efficiently utilized when multiple applications contend for the shared resource. In fact, very often, inter-application effects may lead to destructive interferences [49] (one application kicking out the data of some other application). These effects may become very pronounced in modern operating systems (OSs) as they do not normally allow applications to control resource usage, and they themselves are unable to provide any guarantees (known as differentiated Quality-of-Service (QoS)) to the applications. Our goal in this paper is to make use of I/O access latency and disk utilization as metrics for governing the storage cache partitioning across multiple applications. Our main contributions in this paper include:

- Use feedback control theory to adaptively partition a storage cache at runtime amongst multiple applications such that their QoS (data access latency in our case) can be met.
- Build a storage cache performance model which helps guiding the main controller (used in the defended scheme).
- Experimentally evaluate the proposed cache partitioning scheme using a diverse set of applications. Through extensive experiments using a variety of real application traces, we verified the applicability of feedback control for the storage cache partitioning and the utility of our cache performance model to the success of the approach. Not only were we able to achieve the aggregate hit rate targets per application, but in some cases we improved upon it. Specifically, for a mix of five applications, we saw an improvement in aggregate hit rate by at least 2% and a maximum of 9% per application.

The rest of the paper is organized as follows. We start with building up some background in Section 4.2. In Section 4.3, we motivate the use of feedback control theory and the need to partition the storage cache wisely. We describe the main aspects of our approach in Section 4.4. In Section 4.5, we provide the algorithm used to meet the targets specified to our system. Experimental setup and evaluation of our defended scheme are presented in Section 4.6, followed by a discussion of experimental results in Section 4.7. Finally, section 4.8 presents a body of research that has used ideas related to our work.

4.2 Background

4.2.1 Control Theory

Control theory has been successfully applied to many electro-mechanical systems. Feedback control has found its application to computing systems as well, as demonstrated by recent research efforts [4, 11, 10, 18, 25, 34, 33, 42]. The basic idea behind feedback control is to measure the output of the system which is being controlled and use the error (the difference between output and target input) as a guide to achieve a specified goal. This goal in computing environments can be response time, throughput, resource utilization, etc. Since the output of the system is guiding the control input which in turn affects the output, the approach is called *feedback control*. Though not formally employed, the idea of feedback is inherently present or used in many other techniques employed in computer science to achieve certain objectives. A successful implementation of feedback control requires a good understanding of how the control input affects the measured output. Apart from the control effort, there may be other inputs to the system which are not controllable. These are generally referred to as disturbances and a robust design should minimize the effect of these disturbances. Hence, identifying the disturbances present in a system may be vital to the success of the controller. Depending on the requirements posed, there can be different control objectives. They can be broadly classified as *Regulatory Control, Disturbance Rejection,* and *Optimization*. In this work, we are mainly interested in regulatory control, where the idea is to minimize the difference between the measured output and the target input. A basic control loop is shown in Figure 12 with the controller making the main decisions based on the control error which are fed to the target system. More information on formal control theory can be found in [19, 35, 42, 18].

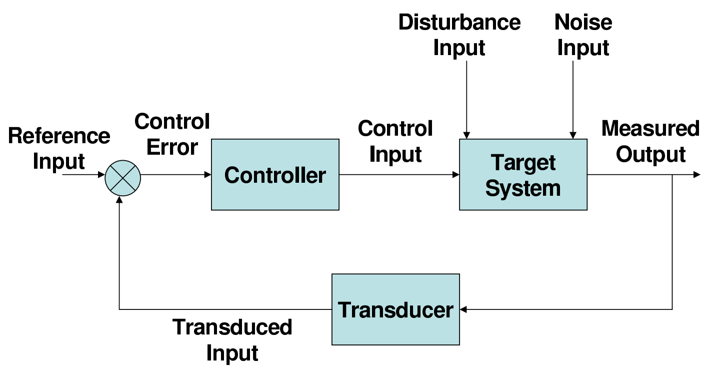


Figure 12: Block diagram of a feedback control system.

4.2.2 System Layout

The basic layout of our target system is shown in Figure 13. In this system, each application specifies its QoS in terms of maximum tolerable I/O latency at the time of instantiation, and the proposed approach partitions the storage cache to satisfy the specified QoS's for all applications. The system model we simulate here is generic and the main idea is to mimic a scenario where multiple applications share a common resource (in our case it is the storage cache). This model is easily extensible (with minor modifications) for various other system implementations used in industry or research. More specifically, it can be looked at as a buffer cache used by Linux kernel. In Linux, there is buffering for the block I/O devices that is done by maintaining lists of buffers for each device. The management of these lists of buffers is what we intend to do in this body of work. The management of the buffers using feedback control and the use of cache performance model brings novelty to our work. This kernel buffer cache is a part of the main memory that performs the function of a storage cache. The default Linux cache management scheme uses LRU [53] as the data replacement policy, which, as we see later, may not be a good choice when considering multiple application scenario.

4.2.3 Disk Utilization

We define *disk utilization* as the percentage of time disk is busy servicing a request. When there is a miss at the storage cache, it leads to a disk access which makes the disk busy. The ratio of busy to idle time for a disk can be approximated as a product of the number of transfers per second and the average access time for the disk. Note that each disk in the storage system may have a different data access time specification (disks can have different RPMs). Also, depending on the disk utilization level, this average access latency value can change for the disk. A higher disk utilization value means the disk is busy for a larger percentage of time which in turn is due to the higher number of I/O requests sent to the disk per unit of time. One may wish to have a high disk utilization value to achieve better resource consolidation. Note however that, higher utilization may affect the performance of the application in an undesirable manner. As a result, typically, storage administrators determine an acceptable disk utilization level, which in turn affects the average latency of a disk access.

4.2.4 Our Goal

In the presence of multiple applications sharing the same storage cache, our aim is to achieve a specified data access latency per application and also to keep the disk utilization levels within a certain range. Our cache partitioning scheme considers these two constraints, one from user's point of view, while the other as seen by the storage system administrator. The QoS used by our control strategy will be an overall (or aggregate, used interchangeably in this work) hit rate target which will factor in these two constraints. We discuss more on this translation (from latency specification

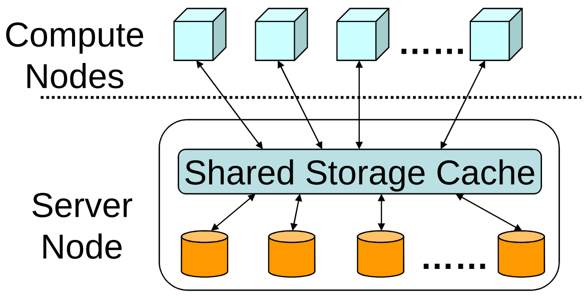


Figure 13: System layout.

Case	Number of requests	Hit rate	Cache usage	Class
1	L	L	L	2
2	L	L	H	1
3	L	H	L	2
4	L	H	H	1
5	H	L	L	2
6	H	L	H	1
7	H	H	L	2
8	H	H	H	1

Figure 14: Table of all possible combinations for different cache performance metrics.

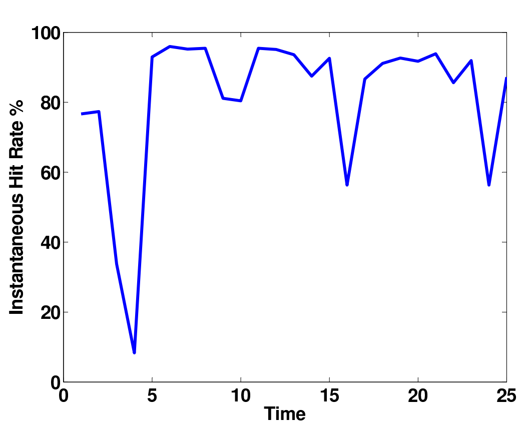
4.3 Empirical Motivation

As pointed out earlier in Section 4.1, when there are multiple applications exercising the same storage cache concurrently, there is a need to partition that cache dynamically. The part that makes this partitioning interesting and nontrivial can be explained better by looking at curves in Figure 15. The curves in this figure plot the characteristics of a benchmark (TPCC) captured over its execution on a fixed size storage cache. The x-axis denotes the execution time units. The curve in Figure 15(b) indicates that the number of accesses (I/O requests generated by the application) made during each time quanta are different. Similarly, the plot in Figure 15(a) indicates that storage cache hit rate is not constant either. Although not presented here, most of the applications we experimented with exhibit a similar behavior. This suggests that the applications' I/O behavior changes over time. The most common requirement for any user of the application is data access latency. The average data access latency of an I/O request is dependent on cache hit rate and the disk access time. The hit rate is affected by the cache space provided to the application (usually increases with more cache space), whereas disk access time is dependent on the disk utilization level (disk access time grows exponentially with increase in disk utilization level). As a result, in order to keep data access latency under tolerable limits, we need to control either or both of these parameters.

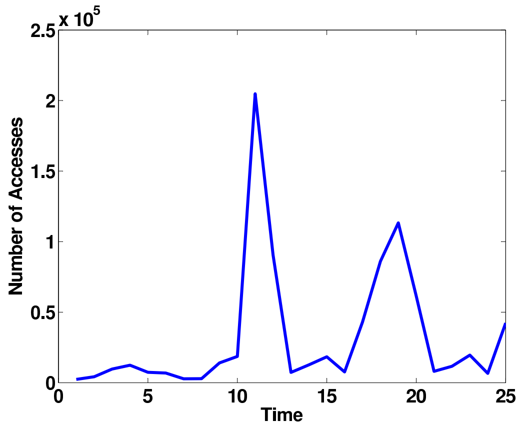
A request generated by an application reaches a disk only when there is a miss at the storage cache. This implies that the disk utilization can be regulated by controlling the number of storage cache misses. The easiest way to keep this check is to allocate cache space for an application carefully. At first glance, one might think of giving more cache space to applications during periods when number of requests is more. There are phases in an applications' execution where the number of requests are more, but the hit rate is high and cache usage being low, indicating low cache requirement. Let us try to look at all such possible combinations. A list of all these combinations is given in Figure 14.

In order to refine the cases of interest, let us try to understand what application behavior to expect from a particular response. Note that when the cache usage is low (L), it means there was some free cache available. Similarly, when the cache usage is high (H), it indicates that the whole available cache is being used. Using the above mentioned inferences, we can identify the following scenarios and make the following conclusions:

- Case 1: Cache usage and hit rate both are L, indicating that during the period of interest, the values obtained are the maximum hit rates achievable and the maximum storage cache requirement of the application. It will not benefit to provide more cache during this period.



(a)



(b)

Figure 15: A look at characteristics of the TPCC benchmark during its execution on a 64MB storage cache. (a) Hit rate computed per sample interval, and (b) Total I/O requests generated per sample interval.

- Case 2: When the cache usage is high and the hit rate is low, it is possible to improve the hit rate if the application is given more cache.
- Case 3: The high hit rate might be due to the good cache locality as the cache usage is low. It indicates that the maximum hit rate is already achieved with the amount of cache used.
- Case 4: This is similar to case 3 but the cache usage is H indicating a possibility for improvement in hit rate by providing more cache.
- Case 5: It is intuitive to see that even though the number of I/O requests is H, the cache usage and hit rate are L, indicating that the application will not use more cache if provided; hence, no improvement in hit rate with more cache. This is similar to case 1.
- Case 6: A low hit rate with a high cache usage hints that the application would have used more cache if available. Hence, there is a possibility of improvement in hit rate with more cache considering that the number of requests is H.
- Case 7: This is again a case where the hit rate is high due to high cache locality, indicated by low cache usage. We infer that the maximum hit rate is already achieved and maximum cache requirement is low.

- Case 8: Possibility of improvement with more cache even though the hit rate is high. This is because the cache usage and the number of requests are also H.

Going through the above interpretations, one can group all the above cases into two main classes: one in which there is a possibility of hit rate improvement (class 1), the other where the maximum hit rate is already achieved (class 2). Class1 cases are of interest to us as they provide opportunity of improvement. From amongst the four cases in Class1, i.e., cases 2, 4, 6, and 8; only cases 2 and 6 should be of concern because of the low hit rates achieved. Both these cases indicate a possibility of improvement if given more storage cache space, but the part that distinguishes them is the number of requests. When we consider case 6 as compared to case 2 from a disk utilization control perspective, it is easy to see that case 6 is of greater concern. This is mainly due to the higher number of I/O requests generated during an interval. Even with a low cache miss rate, the number of requests reaching the disk may be higher as compared to case 2 or some other cases. If the number of I/O requests reaching the disk is not regulated, this might lead to overloading the disk, thereby compromising the desired data access latency.

The above analysis illustrates that it is non-trivial to make cache allocation decisions for an application during the course of execution without having sufficient information at hand. This problem of storage cache allocation becomes even more interesting and complicated when multiple applications share a cache which has to be partitioned in order to achieve a certain goal. The biggest problems that arise are due to unpredictability in application behavior and conflicting allocation decisions.

We must better understand the equation for latency of an I/O request in order to get a better idea on what to control and how to control:

$$Latency = H * T_{hit} + (1 - H) * T_{miss}.$$

In this formulation, H is the hit ratio and gives an idea about the number of accesses that can be serviced by the storage cache from the total number of I/O accesses generated by the application. T_{hit} is usually quite small and constant when compared to T_{miss}, which is the time taken to service a request reaching the disk. When considering this equation, it is tempting to assume that T_{miss} is a constant quantity, which does not change during the execution. As a result, for meeting certain latency targets, this simple assumption leads to controlling the hit rate alone. However, in reality, T_{miss} is affected by the disk utilization level and may change during the course of execution depending on the load at the disk. Rearranging the terms of the above equation and assuming that T_{hit} is much smaller than T_{miss}, one can find the hit rate percentage required to meet a latency target (assuming T_{miss} is known for various disk utilization levels for all type of disks in the storage system) by using the following equation:

$$H\% = (1 - Latency/T_{miss}) * 100.$$

We use the above equation for factoring the latency and disk utilization constraints in to an overall hit rate target specification.

4.4 Our Approach

As discussed in Section 4.3, keeping in mind the equation of latency, our system uses two input constraints to guide the storage cache partitioning scheme. Both these inputs are necessary to make an appropriate hit rate requirement estimation which in turn is used to guide the controller for partitioning the storage cache wisely.

4.4.1 Specification of the QoS

When an application is being run on a shared resource system, the user is mainly concerned about the application meeting the performance goals. One of the most common methods to specify QoS for an application is in terms of data access latency. This requirement for latency can be translated to a hit rate requirement for the I/O requests. The equation required for such a translation is presented earlier in Section 4.3.

In this work, we are not only concerned with meeting the user's need for latency of access but we also try to maintain a certain disk utilization level. As stated earlier, depending on the disk utilization levels in the storage system, the power and performance achieved may vary. A storage administrator may want to maintain high resource utilization to meet some power budget or it could

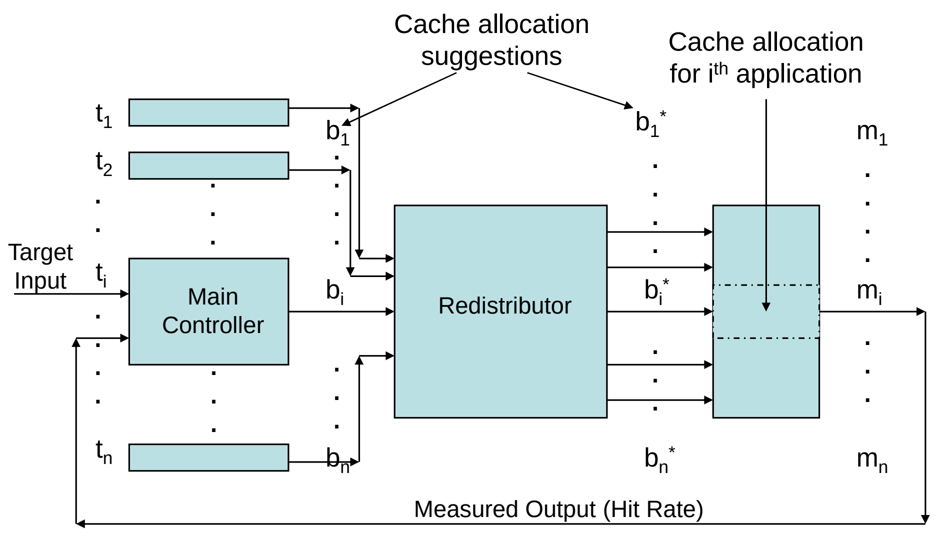


Figure 16: Control architecture for the target storage system.

be for resource consolidation purposes.[1] From the latency equation presented earlier, one can see that as the disk access latency increases (as a result of increased disk utilization level), so does the hit rate required to achieve the desired latency of access. Therefore, we see that there are two factors that affect data access latency: the hit rate and the disk access latency. The inputs for our scheme are the latency targets for each application which are provided by the user and a time of access for each disk in the storage system (provided by the system administrator). We assume that system administrator has enough knowledge about disk access times under various levels of disk utilization. As a result, if she/he wants to keep the utilization under a certain range, the corresponding time of access for the disk is used in the latency equation.

4.4.2 Control Aspect

The control architecture in Figure 16 is inspired from previous work on storage cache partitioning [27]. There are two main components to our design:

1. Main Controller: This is the controller (left portion of Figure 16) which makes suggestions for storage cache space allocation. Each application has its own main controller. The job of a controller in general is to help track the reference input t_i as accurately as possible.

[1]Note that, if the overall storage system utilization is low, the system administrator may choose to power off/discard certain disks.

The reference input in our scenario is a hit rate target. Recall that this target is a translation taking into consideration the original two input constraints, namely, data access latency and disk utilization level. The controller outputs a possible cache allocation b_i for the application of concern, which if provided, may help the application attain the required hit rate target. The response of the system to the current allocation is measured at the end of each time interval and is compared with the reference input. A good controller is one that minimizes this difference, more commonly referred to as the *error signal* in control theory. In this work, we implemented and tested two different types of controllers.

• PID Controller: This is a very popular controller used in many control applications. It has three components which provide proportional, integral and derivative control. Each has its own effect on the system response and error control. For example, a large value of proportional gain K_p, will make the system response quicker but might hurt the system stability. Similarly, the derivative gain K_d is used to reduce the magnitude of the overshoot but it also magnifies the noise signal. The governing equation for this kind of controller can be expressed as:

$$b(t) = b(t-1) + K_p e(t) + K_i \sum_{u=1}^{t} e(u) + K_d(e(t) - e(t-1)).$$

The output of the controller is a storage cache allocation suggestion denoted by $b(t)$. Thus, $e(t)$ represents the error term which is computed as the difference of the measured hit rate m and the target hit rate t_i. $K_p, K_d,$ and K_i are the controller gains which have a constant value. Estimation of these gain values is crucial to the success of the PID controller. For our experiments, we chose the values which gave the best results from amongst a set of tested values. Note that since this is a per application controller, the set of gain values can be different for each, making the design process quite tedious.

• History Based Controller: This is the controller that is defended in this paper. The governing control equation can be stated as:

$$b(t) = b(t-1) + \delta b(t),$$

where $\delta b(t)$ is a correction factor computed as a difference of $pb(t)$ (the predicted cache size using the cache performance model), and $m(t-1)$ (the moving average of the previous values of b) determined using the same control law. The moving average can be computed over the previous intervals of time using the following equation:

$$m(t-1) = \beta m(t-2) + (1-\beta)b(t-1).$$

The older values of average cache size (m) are exponentially attenuated with a factor of β, where $0 < \beta < 1$. A higher value of β will increase the window size over which b is averaged. The motivation for using a history based controller is to make use of the knowledge about the past cache behavior operating under the current workload. Instead of developing a complex control system with components for model prediction of the system, we employ a machine learning based technique that uses curve fitting. Note that, as opposed to the PID control, there is no need to determine the suitable gain values in advance for making the control loop successful. More details on the working of this controller are given in Section 4.5.

2. Redistributor: The job of this component is to handle the scenarios where the amount of cache available is either less than the amount demanded by the running applications or is more than what is needed to meet their QoS demands (this demand is the sum of all cache suggestions made by each of the main controller). In both these cases, there is a need to redistribute the available storage cache space. This is achieved by enforcing some high level policies. For the case where the amount of the storage cache in the system is less than the demanded cache, we take away a certain amount of cache from each cache suggestion. The amount taken away is decided by the incremental loss in hit rate if the same amount is taken away from all the applications. This is decided by looking at the slope of the hit rate curves in the cache performance model. Intuitively, we would like to take away more cache from an application which is going to have the least depreciation in its hit rate. On the other hand, when we are left with some unallocated cache space and none of the applications were flagged, we distribute this space in proportion to the hit rate benefits we might achieve upon giving that extra cache to the application. The intuition is to maximize the overall hit rate achieved by the system. More about the policy employed and its usage in our control system is given later in Section 4.5.

It is hard to control the number of disk accesses for each time interval. The reason is that, one cannot accurately predict the number of requests that are going to be generated for the next time interval. On the other hand, controlling the aggregate hit rate or the instantaneous hit rate are both achievable targets. Aggregate hit rate here means the cumulative hit rate achieved from the start of the applications execution up to the current instant. Long term hit rate target is easier to achieve and will also cause less fluctuations in the cache allocations suggested during the execution period. Long term hit rate target may not be a good idea for applications with high degree of fluctuations in their I/O access patterns. On the other hand, instantaneous hit rate is evaluated for each control interval separately by finding the ratio of the number of hits to number of requests within that interval. This solves the problem of keeping the disk utilization within the required range for each time interval. This is also a better approach in achieving the hit rate target for applications with highly fluctuating I/O access patterns, though it may lead to a similarly fluctuating cache allocation pattern which is not very desirable. The advantage with the scheme being the latency of access target is more closely tracked as the allocation decisions are based on application behavior in the last interval only. As mentioned earlier, our goal will be to achieve an aggregate hit rate target per application. Note that in our defended scheme we will make cache size predictions for every interval based on a cache performance model which accounts for instantaneous hit rate changes. Though the cache prediction is made based in the instantaneous hit rate, the control law used by the main controller minimizes the error in overall hit rate.

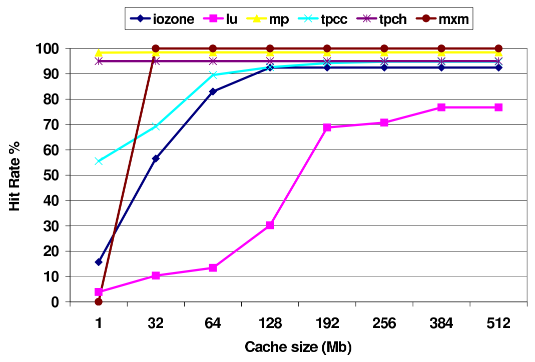


Figure 17: Aggregate hit rate with increasing cache size.

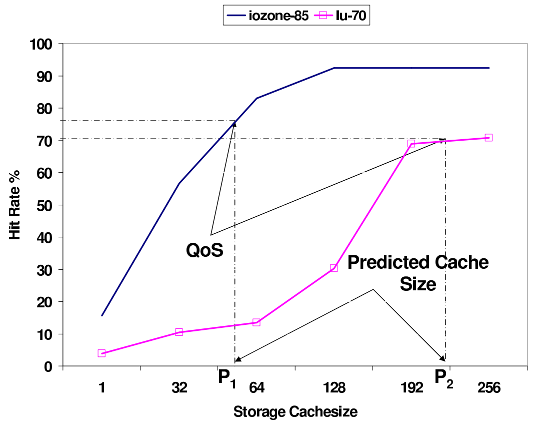


Figure 18: An example snapshot of how the cache performance curve is used and updated in our defended scheme.

4.4.3 Cache Performance Model and the Learning Aspect

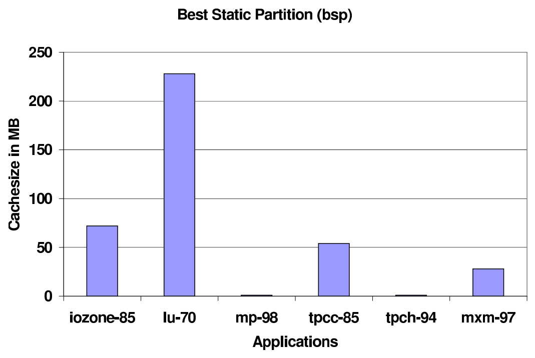


Figure 19: Minimum amount of cache needed by each application to meet the QoS. The values for mp-98 and tpch-94 are 1MB and 1MB, respectively.

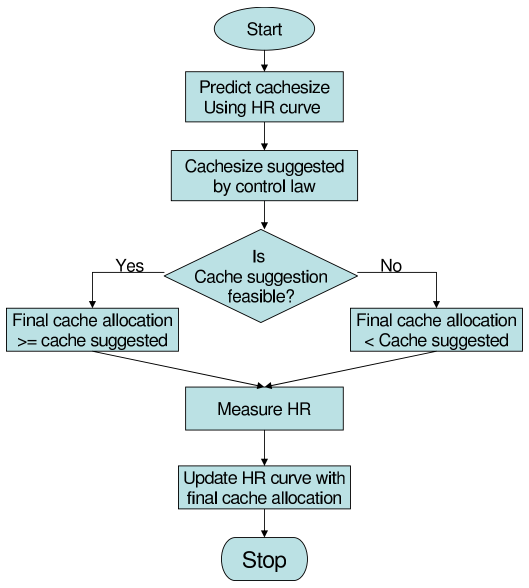

Start
Predict cachesize Using HR curve
Cachesize suggested by control law
Is Cache suggestion feasible?
Yes
No
Final cache allocation >= cache suggested
Final cache allocation < Cache suggested
Measure HR
Update HR curve with final cache allocation
Stop

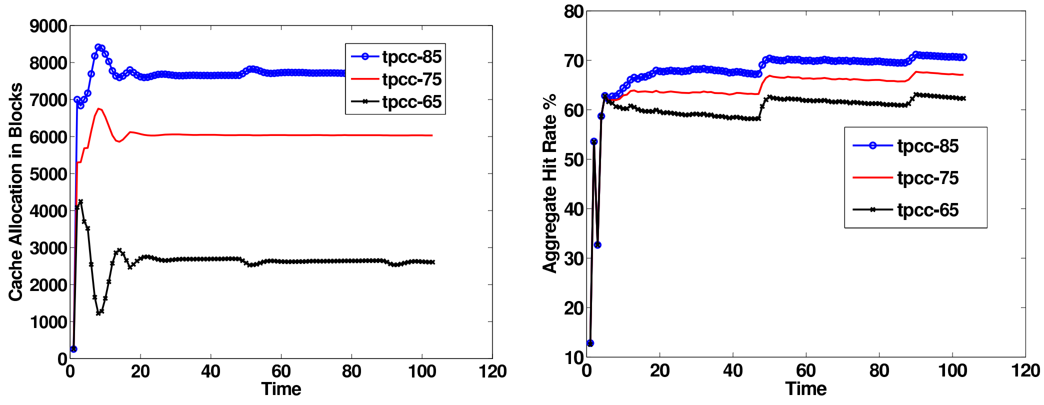
Figure 20: High level view of the dynamic partitioning algorithm per application. Note that this represents only a *single iteration* within the feedback loop.

We employ a storage cache performance model for each application which helps our cache controller better partition the cache amongst the various applications. A simplistic cache model shown in Figure 17 tries to capture the effect of varying the cache size on the hit rate achieved at the storage cache. Note that this model is developed by looking at the entire execution of an application under different cache sizes. We target to achieve an aggregate hit rate during the entire execution of the application, we develop and update a similar cache model for our defended scheme. The updating will help make the predictions more accurate (by keeping track of the changes in application behavior) for the cache size required to achieve the required hit rate. As an example illustration, consider Figure 18. Here, we consider two applications with different QoS targets. The targets are mentioned after the applications name in the legend bar. Note that, this curve will be different from the one shown in Figure 17 as the prior is depicting the hit rate achieved after the entire execution of a single application on a particular cache size and the latter (Figure 18) is updated every quanta of the control interval when multiple applications share the same storage cache. Note that there is a separate curve for every application sharing the cache. Also, a value on the y-axis corresponding to a particular x-axis value can change with time. In Figure 18, we look at the minimum cache space required to meet the QoS demands for each application, marked as P_1 and P_2 in the figure. This predicted cache size is used in the governing equation of the main controller (for our defended scheme) to suggest a cache size. The updating of this curve is done when the final allocation is made and the hit rate achieved with that allocation is measured. This continuous updating during runtime will ensure that the cache performance curve captures the dynamic variations in application behavior and one is able to partition the storage cache wisely amongst the applications.

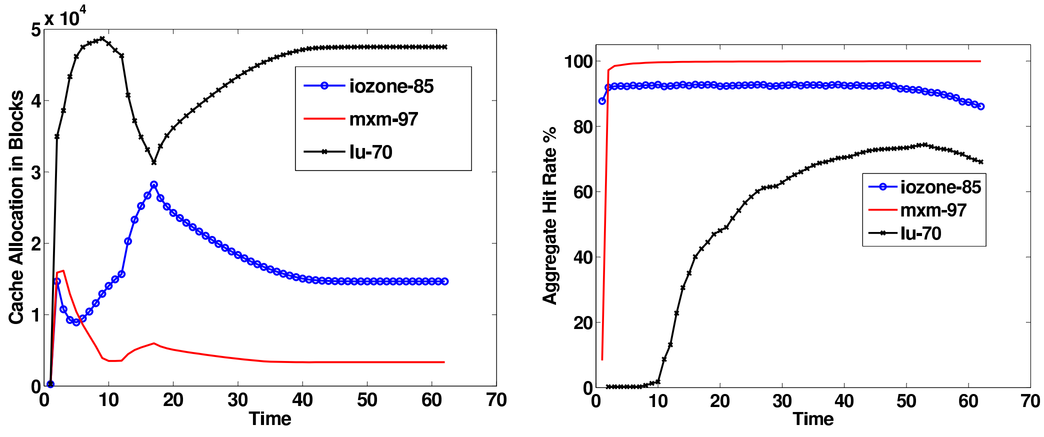
The cache performance model developed is used by the two components in our control architecture, namely, the main controller and the redistributor. As mentioned earlier, these components, when generating their outputs, can base their decisions on some higher level policies. The policy used by both the components can be targeted towards improving the hit rate of the system as a whole. Let us consider the case where we are left with some free (unused) cache space after meeting the QoS demands of all applications. In this case, redistributor comes into the picture by dividing this free cache space amongst the running applications such that the overall hit rate (of the storage system) improves to the maximum possible extent. The cache performance curves for the applications give an insight as to which application will benefit most in terms of hit rate if a fixed amount of cache was given to it. This can be interpreted by looking at the slopes of various curves and then allocating the free cache in proportion to the slope values attained form their respective cache performance models. An application with a higher slope indicates more incremental gain. Similarly, the redistributor can take away the extra cache from the suggestion given by the individual main controllers such that the depreciation in overall hit rate is minimized. In this way, we make use of the simple and characteristic cache performance model to introduce machine learning into our design.

4.5 Algorithm

All the pieces and components of our approach have already been discussed in the previous section. Let us now see how these components work together. Our feedback control based storage cache



(a) **Mix of Three TPCC Instances.** (b) **Mix of Three TPCC Instances.**



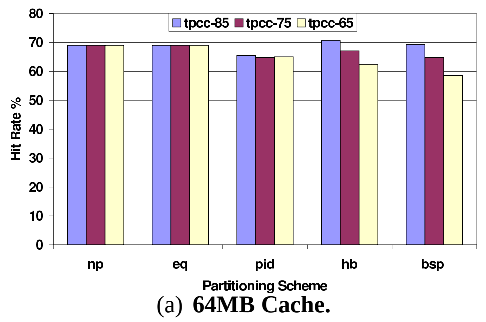
(c) **Mix of Iozone, Lu, and MXM.** (d) **Mix of Iozone, Lu, and MXM.**

Figure 21: Hit rates and cache allocation variations during the execution of an application mix with HB control on a shared storage cache.

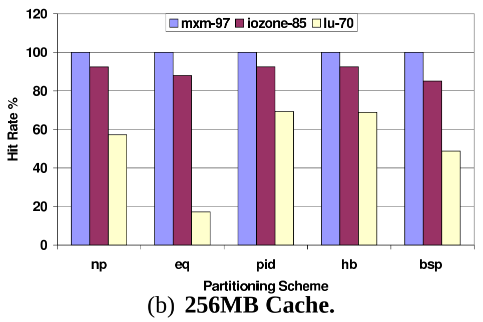
partitioning algorithm can be divided into three phases of operation.

• Phase 1: We call this phase the prediction phase. As discussed earlier, we maintain a storage cache hit rate curve for each application that will be used to make a coarse predictions about the size of storage cache that should be made available for the application in the next control interval.[2] We make this prediction for the cache size in a conservative fashion by looking at the minimum cache that can possibly achieve the demanded QoS (see Figure 18). Based on the curves in Figure 17, the minimum cache size required by each of the application is provided in Figure 19. Note that this information can only be available if one has profiled the entire execution of an application. One

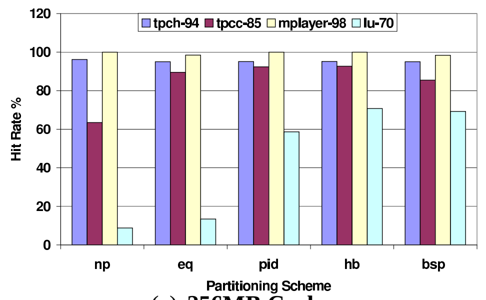
[2]As stated earlier, this curve maintains the (cache space, hit rate) points observed so far, for the application during execution.



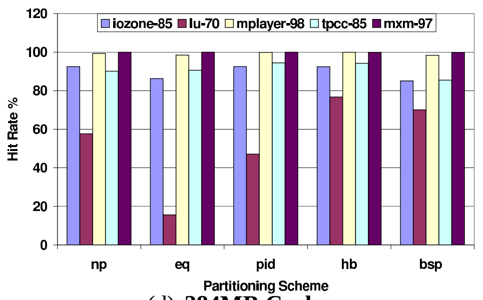
(a) **64MB Cache.**



(b) **256MB Cache.**



(c) **256MB Cache.**



(d) **384MB Cache.**

Figure 22: Aggregate hit rate achieved by individual applications in different workloads (application mixes) using various cache partitioning schemes.

of the major assumptions we make here is that we start off with three initial points in the cache performance curves. These points correspond to cache sizes equal to 1MB, total storage cache size (C), and C/2.

• Phase 2: This phase is where the control laws come into the effect. The predicted storage cache size is used as a suggestion by the main controller that in turn suggests another cache size based on its control law. The history based control law used by our main controller has already been discussed earlier. Recall that the decision for allocations suggested by these main controllers are independent of each other. Due to this reason, the cache size suggestions made by all the main controllers cannot be directly applied to partition the storage cache. Some of these decisions may lead to an infeasible partitioning. Consequently, the redistributor component is used to make a final cache allocation decision which is feasible and efficient. The allocation suggested by the redistributor is done so as to use exactly the amount of total storage cache available and the distribution is such that it improves the overall hit rate of the system. Infeasibility stems in two cases where the sum of the storage cache size suggested by the individual controllers is more than that is available or is less than that can be used.

In the first scenario, the cache size suggestions are summed up to find the extra space that is being demanded overall from the total storage cache available in the system. All the cache suggestions are pruned so as to fit in the available storage cache. This is done by reducing each cache space size suggestion in the proportion of the slope values in their respective cache performance curves. In this manner all the applications will be hit in a way so as to minimize the impact on the overall progress of the system. We choose this policy to provide fairness when there is overload present in the system. In the second scenario, where we are left with free cache after providing each application with the minimum cache space that will satisfy its QoS, we allocate each application more cache space so as to maximize the overall hit rate of the storage cache (when all applications are considered). This is achieved by allocating a cache space to each application in proportion to the benefit it might achieve in terms of improving its hit rate with the same amount of incremental cache given to it. This decision is made by looking at the cache hit rate curves we maintain for each application. The slope of the curve between the points for minimum cache size allocated and the incremental cache given is used as the comparison metric. Note that we base our pruning decision by considering slope values assuming the curve is linear in the region of interest. This assumption about the linearity has been applied before in [16].

• Phase 3: The final phase is the cache hit rate curve updating phase. In this phase, we look at the hit rate achieved by the cache allocation decision made by the redistributor. If this new cache allocation resulted in a different hit rate as already present in the hit rate curve, we make an update.

One iteration that illustrates the flow of decisions in our control approach to partitioning the cache is given in Figure 20. This iteration repeats in every control interval until the application execution finishes.

4.6 Experimental Setup

We made use of a cache simulator called Accusim [5] for our experiments. Accusim is a trace-driven, buffer cache simulator originally designed for studying cache replacement algorithms. We modified it to simulate a shared storage cache scenario with multiple applications sharing it such that it can be partitioned and the individual hit rate behavior can be studied. It can accurately simulate

Table 6: Workloads (application mixes) used for experiments.

Workload	Included Applications
MIX2	tpc-c, tpc-c, tpc-c
MIX3	mxm, iozone, lu
MIX4	tpc-h, lu, mplayer, tpc-c
MIX5	mxm, lu, mplayer, tpc-c, iozone

Table 7: Default QoS targets in terms of aggregate hit rate for individual applications.

Application	QoS
tpc-c	85
tpc-h	94
mplayer	98
lu	70
iozone	85
mxm	97

I/O time under prefetching. We enabled prefetching in all our experiments. The layout of our target system is given earlier in Figure 13. The main controller and the redistributor are implemented inside Accusim. The workloads used for our experiments are described briefly below:

We selected a few representative applications and merged their traces to form mixed traces which mimic traces captured when those individual applications would have run simultaneously on a system. We used TPC-C and TPC-H benchmarks which are online transaction processing (OLTP) applications. An open source implementation of TPC-C known as TPCC-UVa [32] was used. Both these applications generate a high volume of I/O read and write requests. These applications have mostly low data reuse, and exhibit sequential data access pattern. While TPC-H was run on a data set of size 1GB, TPC-C ran on a data set of size 137MB. We also used Mplayer which is a software used in Linux to play audio/video files. This application was used to provide us a streaming kind of I/O behavior. Mplayer has very good spatial reuse. Our experimental suite also includes an out-of-core implementation of LU decomposition from ScaLAPACK [12]. Most of the applications we tested ran over data sets of sizes greater than 100MB. Iozone [3] is a filesystem benchmark tool which generates and measures a variety of filesystem operations, effectively generating a lot I/O requests. We ran Iozone in the automatic mode on a Linux machine. Finally, we also implemented an application called MXM, which is again an out-of-core matrix multiplication application. The composition of various mix traces we used for our experiments are given in Table 6. All these mix traces were used to evaluate the various storage cache partitioning schemes. The intuition behind the choice of applications when forming a certain mix was to create mixes with diverse set of I/O requirements, such that they could exercise the storage cache partitioning scheme well.

The numerical value suffixed to the application name denotes the hit rate percentage required by the application (translated QoS). A table showing the hit rate targets we used for the individual applications is given in Table 7. Again, these values were chosen keeping in mind the maximum achievable hit rates by these individual applications. The schemes we evaluated in this work are:

- No-Partition (np): This is the default scheme used in the current Unix systems. In fact, it does not perform any kind of partitioning. Instead, the default scheme (LRU [53]) works on the principle of supply as per demand.

- Equal-Partition (eq): This is a naive static partitioning scheme in which the total storage cache is equally shared by the simultaneously running applications.

- PID (pid): This is the base control scheme, which partitions the cache dynamically at runtime. PID is a very general and useful control law which applies well to most common scenarios. The aim of the controller will be to achieve the QoS objectives as specified by individual applications.

- History-based (hb): The main idea and working of this controller has been presented in the earlier sections. This is the storage cache partitioning strategy defended in this work.

- Best-static partition (bsp): This scheme provides us some insight into what an ideal static cache allocation could do in terms of achieving the QoS targets. This is not a dynamic or runtime scheme. Applications are given the partitions looking at their storage cache performance behavior assuming we have prior knowledge about their execution. This scheme can be thought of as an oracle predictor which tells us the minimum cache an application would need to achieve a certain QoS target. such predictions are given in Figure 19, showing the minimum cache required in MB by each application to meet its QoS target. This prediction will only work when the application requirements do not fluctuate over time. The results from this scheme would help us compare our dynamic schemes for their success in achieving the required QoS.

4.7 Discussion of Experimental Results

We conducted experiments to test the feasibility and efficiency of our storage cache partitioning scheme. The various mixes we used are given in Table 6. As pointed out earlier, our main goal in this work is to achieve an overall hit rate target that meets its demanded data access latency and also keeps the disk utilization levels under the range expected by the system administrator. Figure 22 summarizes the main results where different cache partitioning schemes are compared. We varied the size of the total available cache for different mixes to pressurize the I/O system and hence create a more interesting scenario to test various schemes. Figure 22(a) shows results when a mix with three instances of the same application with different QoS was tested. It can be seen that no-partition and equal-partition cannot differentiate amongst the QoS requirements. History based control provides differentiation amongst the different instances and makes the best effort in achieving the QoS targets. We also show how our history-based control scheme is able to quickly track the QoS requirement in Figure 21(b). Figure 21(a) shows how the corresponding cache allocation decisions are made for achieving the QoS. It is evident that there are not many fluctuations in the cache allocations and it stabilizes around a certain size. Similarly, Figures 21(c) and (d), show the hit rate variations and cache allocation decisions for mix3. The corresponding overall hit rate graph is shown in Figure 22(b). It can be seen that none of the schemes were able to achieve the QoS posed by LU. Specifically, the two static schemes, i.e., no-partition and equal-partition perform very poorly. The control based schemes have the best performance. Note that in scenarios where the QoS posed are such that they are infeasible to achieve from the available storage cache, our defended scheme tries to make a best effort. For best static partitioning, even if we have prior knowledge about its cache performance model, it might not be feasible to allocate the predicted size due to limited cache space. Relatively, history-based control outperforms all other adaptive schemes. Even the best-static-partition, which is based on the prior knowledge of applications execution performs poorly as compared to our defended scheme. Figures 22(c) and (d) validate the usage of our scheme even when the application mixes are very diverse. It confirms the applicability of the scheme even if the mix contains large number of applications with diverse I/O requirements. Our history-based control outperforms all other schemes. Not only it is able to achieve the QoS demanded, but it also improves the overall hit rate whenever possible.

We also performed sensitivity experiments to test our history based scheme when various related parameters used in the scheme change. We varied the total storage cache size available, the cache block size that is the minimum unit of allocation, the size of control interval time, and the QoS requirements posed to experiment on mix4. Results are not depicted here due to space constraints. In summary, the results indicated that the history-based scheme is robust with regards to changes in these parameters.

4.8 Discussion of Related Work

Control theory has been vastly used to help make resource allocation decisions. One of such works [30], present an operating system where a global resource allocator using control theory partitions various shared resources in order to achieve the QoS demands of distributed multimedia applications. Multimedia streams posses temporal and informational property that help in design and implementation of the controller. In [27], hit ratio is being controlled. The results plot hit ratio vs the number of accesses as the execution progresses. Although, the number of accesses show progress of the application but it does not indicate the application behavior with respect to time. It hides the fact that the number of accesses can be different for each time interval depending on the application behavior. [34] introduces control theory for managing web-proxy cache shared by the web applications which may belong to different classes and hence may pose different performance goals. [35, 71, 28, 40] also present adaptive techniques to achieve some form of performance differentiation applicable to different scenarios. Cache replacement policies [70, 68] try to make the best selection of the candidates for eviction when a new data has to be brought in to the cache. Such an approach may tend to improve the overall hit rate from the point of view of the cache, but it is hard to make performance differentiation amongst the applications sharing the cache. Scheduling of the I/O requests is another way of improving the I/O performance but it has the same limitation of not being able to differentiate amongst applications QoS requirements [65, 51, 50, 52]. Applications can provide hints [66] which can be used to partition the cache with a similar objective of maximizing the overall hit rate. There have been numerous other techniques trying to improve the I/O performance wither by partitioning disk bandwidth wisely [67, 66] or prefetching data in a smarter way [58]. Partitioning of cache can be done for performance insulation and efficiency [60] or to minimize power [48]. Minimizing the miss-rate of the memory in a multiprocessor by proper scheduling and partitioning was presented in [56]. They make use of some hardware counters to accurately estimate the isolated miss-rates of each process. Most of these prior works have focussed on maximizing the overall hit rate of the system by cache partitioning. Our idea is to make a best effort in meeting the QoS requirements of all the applications running on the system by partitioning the storage cache space using a control theoretic approach. May be the most closely related work to ours is [27].

Chapter 5

Conclusions

Overall we presented three bodies of work keeping in mind power and performance requirements in different scenarios. We started with a novel Markov model based disk idleness prediction scheme that can be used for reducing disk power consumption when used with a three-speed disk. The work explained in detail why the defended prediction mechanism was better than others and why it saves disk power. To evaluate the effectiveness of our approach, we implemented it using DiskSim and performed experiments with both synthetic traces and real application traces. This was followed by a novel algorithm for managing the disks under your storage system. One of the critical problems in managing a complex storage system is the allocation of available disks across competing applications. The main characteristic that makes this problem challenging is the fact that different applications demand different amount of disk space and can tolerate different disk latencies. As a result, allocating disks for a disk pool that contains multiple heterogeneous disks is not trivial. Providing performance guarantees considering additional constraints such as power and reliability makes the problem even harder. Finally, the storage cache was realized as an important resource for controlling the I/O performance of applications using that cache. Through extensive experiments using a variety of real application traces, we verified the applicability of feedback control for the storage cache partitioning and the utility of our cache performance model to the success of the approach. As future work, it would be interesting to investigate dynamic migration of processes running simultaneously on a compute node, where the number of compute nodes is more than one. Observing the current trends where multiple cache hierarchies exist in the storage system, a wiser choice of grouping the applications sharing a cache should help improve performance.

Bibliography

[1] `http://en.wikipedia.org/wiki/EMC_Invista`.

[2] `http://www.datacore.com/products/prod_sanmelody_govirtual.asp`.

[3] `http://www.iozone.org/`.

[4] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13, 2002.

[5] A. R. Butt, C. Gniady, and Y. C. Hu. The performance impact of kernel prefetching on buffer cache replacement algorithms. *IEEE Trans. Comput.*, 56(7), 2007.

[6] E. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *Proceedings of the International Conference on Supercomputing*, pages 86–97, 2003.

[7] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *ICS '03: Proceedings of the 17th Annual International Conference on Supercomputing*, San Francisco, CA, USA, 2003.

[8] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[9] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-Performance, Reliable Secondary Storage. *ACM Comput. Surv.*, 26(2):145–185, 1994.

[10] Y. Diao, N. G, J. L. Hellerstein, S. Parekh, and D. M. Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the apache web server. In *Proceedings of the Network Operations and Management Symposium 2002*, 2002.

[11] Y. Diao, J. L. Hellerstein, and S. Parekh. A business-oriented approach to the design of feedback loops for performance management.

[12] J. Dongarra and E. F. D'Azevedo. The design and implementation of the parallel out-of-core ScaLAPACK LU, QR, and cholesky factorization routines. Technical Report UT-CS-97-347, 1997.

[13] F. Douglis, P. Krishnan, and B. N. Bershad. Adaptive disk spin-down policies for mobile computers. In *MLICS '95: Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, 1995.

[14] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, San Francisco, California, 1994.

[15] G. Ganger, B. Worthington, and Y. Patt. The DiskSim simulation Environment Version 3.0 Reference Manual. `http://www.pdl.cmu.edu/DiskSim/`.

[16] P. Goyal, D. Jadav, D. S. Modha, and R. Tewari. Cachecow: Qos for storage system caches. In *Eleventh International Workshop on Quality of Service (IWQoS 03), Monterey, CA*, 2003.

[17] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: dynamic speed control for power management in server class disks. In *ISCA '03: Proceedings of the 30th Annual International Symposium on Computer Architecture*, San Diego, California, 2003.

[18] J. L. Hellerstein. Challenges in control engineering of computing systems. In *American Control Conference*, 2004.

[19] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. Feedback control of computing systems, 2006.

[20] D. P. Helmbold, D. D. E. Long, T. L. Sconyers, and B. Sherrod. Adaptive disk spindown for mobile computers. *Mob. Netw. Appl.*, 5(4), 2000.

[21] IBM. Ultrastar 36z15 hard disk drive. `http://www.hgst.com/hdd/ultra/ul36z15.htm`, 2003.

[22] M. I. Inc. Power, heat, and sledgehammer. *White Paper*, 2002.

[23] Intel. Addressing Power and Thermal Challenges in the Datacenter. `http://download.intel.com/design/servers/technologies/thermal.pdf`.

[24] Intel. Increasing Data Center Density While Driving Down Power and Cooling Costs. `http://www.intel.com/business/bss/infrastructure/enterprise/power_thermal.pdf`.

[25] R. Joseph, D. Brooks, and . M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *Ninth International Symposium on High Performance Computer Architecture*, 2003.

[26] M. Kandemir, S. W. Son, and M. Karakoy. Improving disk reuse for reducing power consumption. In *ISLPED '07: Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, Portland, OR, USA, 2007.

[27] B.-J. K. Kang-Won, K. won Lee, K. Amiri, and S. Calo. Scalable service differentiation in a shared storage cache. In *ICDCS*, 2003.

[28] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance differentiation for storage systems using adaptive control. *Trans. Storage*, 1(4), 2005.

[29] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modelling*. PH Distributions, 1999.

[30] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7), 1996.

[31] X. Li, Z. Li, Y. Zhou, and S. Adve. Performance directed energy management for main memory and disks. *Trans. Storage*, 1(3), 2005.

[32] D. R. Llanos. Tpcc-uva: an open-source tpc-c implementation for global performance measurement of computer systems. *SIGMOD Rec.*, 35(4), 2006.

[33] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms, 2001.

[34] Y. Lu, T. F. Abdelzaher, C. Lu, and G. Tao. An adaptive control framework for QoS guarantees and its application to differentiated caching services, 2002.

[35] Y. Lu, A. Saxena, and T. F. Abdelzaher. Differentiated caching services; a control-theoretical approach. In *21st International Conference on Distributed Computing Systems*, 2001.

[36] Y.-H. Lu, E.-Y. Chung, T. Simunic, L. Benini, and G. D. Micheli. Quantitative Comparison of Power Management Algorithms. In *Proceedings of the Conference on Design, Automation and test in Europe*, pages 20–26, 2000.

[37] C. Mellor. Western Digital launches power-efficient disk drives. `http://www.techworld.com/green-it/news/index.cfm?newsid=10711&email`.

[38] M. C. Michael and M. J. Quinn. Overlapping computations, communications and i/o in parallel sorting. *Journal of Parallel and Distributed Computing*, 28, 1994.

[39] K. Okada, N. Kojima, and K. Yamashita. A novel drive architecture of hdd: ”multimode hard disc drive”. In *Proceedings of the International Conference on Consumer Electronics*, 2000.

[40] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. *SIGOPS Oper. Syst. Rev.*, 41(3), 2007.

[41] A. E. Papathanasiou and M. L. Scott. Energy Efficient Prefetching and Caching. In *Proceedings of the USENIX Annual Technical Conference*, pages 255–268, 2004.

[42] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Syst.*, 23(1/2), 2002.

[43] C. M. Patrick, S. W. Son, and M. T. Kandemir. Enhancing the performance of mpi-io applications by overlapping i/o, computation and communication. In *PPoPP ’08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008.

[44] D. Patterson and J. Hennessy. *Computer Organization and Design*. Morgan Kauffman, 2005.

[45] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *ICS ’04: Proceedings of the 18th Annual International Conference on Supercomputing*, 2004.

[46] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[47] R. Rao, S. Vrudhula, and M. S. Krishnan. Disk drive energy optimization for audio-video applications. In *CASES ’04: Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded systems*, Washington DC, USA, 2004.

[48] R. Ravindran, M. Chu, and S. Mahlke. Compiler-managed partitioned data caches for low power. In *LCTES*, 2007.

[49] J. Reumann, A. Mehra, K. G. Shin, and D. Kandlur. Virtual services: a new abstraction for server consolidation. In *ATEC*, 2000.

[50] A. Riska, J. Larkby-Lahet, and E. Riedel. Evaluating block-level optimization through the I/O path. In *USENIX*, 2007.

[51] S. R. Seelam. *Towards dynamic adaptation of I/O scheduling in commodity operating systems*. PhD thesis, The University of Texas at El Paso, 2006.

[52] P. J. Shenoy and H. M. Vin. Cello: a disk scheduling framework for next generation operating systems. In *SIGMETRICS*, 1998.

[53] A. J. Smith. Cache memories. *ACM Computing Surveys*, 14, 1982.

[54] S. W. Son and M. Kandemir. Energy-aware data prefetching for multi-speed disks. In *CF ’06: Proceedings of the 3rd Conference on Computing Frontiers*, Ischia, Italy, 2006.

[55] S. W. Son, M. Kandemir, and A. Choudhary. Software-directed disk power management for

scientific applications. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, 2005.

[56] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *HPCA*, 2002.

[57] UMass Trace Repository. `http://traces.cs.umass.edu`.

[58] S. P. Vanderwiel and D. J. Lilja. Data prefetch mechanisms. *ACM Comput. Surv.*, 32(2), 2000.

[59] T. Šimunić, L. Benini, P. Glynn, and G. D. Micheli. Dynamic power management of laptop hard disk. In *DATE '00: Proceedings of the Conference on Design, Automation and Test in Europe*, Paris, France, 2000.

[60] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: performance insulation for shared storage servers. In *FAST*, 2007.

[61] Y. Wang and A. Merchant. Proportional-share scheduling for distributed storage systems. In *FAST'07: Proceedings of the 5th Conference on USENIX Conference on File and Storage Technologies*, 2007.

[62] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning. PARAID: A Gear-Shifting Power-Aware RAID. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 245–260, 2007.

[63] A. Weissel, B. Beutel, and F. Bellosa. Cooperative I/O: a novel I/O semantics for energy-aware applications. In *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, 2002.

[64] P. Wong and R. F. V. der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. Technical Report NAS-03-002, NASA Advanced Supercomputing Division, January 2003.

[65] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. *SIGMETRICS Perform. Eval. Rev.*, 22(1), 1994.

[66] J. Wu and S. A. Brandt. Storage access support for soft real-time applications. In *RTAS*, 2004.

[67] J. Wu and S. A. Brandt. The design and implementation of AQuA: An adaptive quality of service aware object-based storage device. In *MSST*, 2006.

[68] G. Yadgar, M. Factor, and A. Schuster. Karma: Know-it-all replacement for a multilevel cache. In *FAST*, 2006.

[69] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, San Francisco, CA, 2003.

[70] Y. Zhou, J. F. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *USENIX*, 2001.

[71] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. 1000 islands: Integrated capacity and workload management for the next generation data center. In *Proceedings of the 2008 International Conference on Autonomic Computing*, 2008.