Issues and Challenges in the Performance Analysis of Real Disk Arrays

Elizabeth Varki, *Member*, *IEEE*, Arif Merchant, Jianzhang Xu, and Xiaozhou Qiu

Abstract—The performance modeling and analysis of disk arrays is challenging due to the presence of multiple disks, large array caches, and sophisticated array controllers. Moreover, storage manufacturers may not reveal the internal algorithms implemented in their devices, so real disk arrays are effectively black-boxes. We use standard performance techniques to develop an integrated performance model that incorporates some of the complexities of real disk arrays. We show how measurement data and baseline performance models can be used to extract information about the various features implemented in a disk array. In this process, we identify areas for future research in the performance analysis of real disk arrays.

Index Terms—RAID, analytical performance model, array cache, parallel I/O, enterprise storage systems, I/O performance evaluation, disk array.

1 Introduction

As computer applications have become more data intensive, the demands on storage systems in terms of efficient storage and retrieval have correspondingly increased. Currently, RAID (Redundant Array of Independent Disks) architectures have become the architecture of choice for large storage systems since they employ striping (i.e., parallelism) and redundancy across multiple disks to increase capacity, speed, and availability of storage systems. Modern RAID disk arrays contain multiple disks, large caches, and sophisticated array controllers that perform optimizations such as load-balancing, request coalescing, and adaptive prefetching. Thus, disk arrays are complex devices, and it is difficult to understand the behavior of disk arrays and compute their performance measures. Since storage systems are often the system bottleneck in computer systems running I/O intensive applications, the degradation in system performance as a consequence of a nonoptimal disk array configuration can be considerable [23]. Consequently, improving the performance of the storage system would result in an overall system performance upgrade.

The key performance metrics studied by performance engineers are disk array response time, throughput, and queue length. Response time is the time spent by an I/O request at the disk array, waiting for and receiving service. Throughput is the rate at which I/O requests are serviced by the disk array. Thus, response time measures how fast an individual request can be stored and retrieved, and throughput measures how many such requests can be serviced within a specified time. A related measure of interest is queue length, the number of I/O requests at a disk array, and by Little's Law, mean queue length is the product of mean response time and mean throughput. Both simulation methods and analytic techniques have been used to compute storage system performance measures under various design trade offs. Typically, simulation models are more accurate than analytic models since they make fewer simplifying assumptions. By contrast, analytic models are much less expensive and significantly faster than simulation models. Thus, analytic models are useful as they provide a "quick and dirty" way to isolate potential problem areas, with comparatively little effort. This can be of advantage in complex storage management systems such as Minerva [1], which have to evaluate thousands of candidate data-to-device configurations to arrive at an optimal-cost solution.

In our paper, we focus on using standard performance techniques to develop an analytic model for understanding the effect of various disk array features on the performance of real disk arrays. In order to analyze the mechanisms of real disk arrays, we extend prior performance modeling work by including the effects of caching, parallelism, and array controller optimizations in our performance model. This is in contrast to earlier models that have either focused on modeling the parallelism of disk arrays or on modeling the caching policies of disk arrays, but not both. (See Section 2 for survey of related work.) Further, our performance model can be used to analyze the effects of array controller optimizations such as write-back caching policies, coalescing of multiple disk accesses into a single disk access, redundancy layout in the disk array, and adaptive data prefetching by the array cache. Thus, our study helps shed light on 1) how disks and caches interact with each other and jointly impact the performance of disk arrays, and 2) how the array controller optimizations impact the performance of disk arrays.

Our results indicate that standard performance techniques are useful in isolating disk array features that could be improved for better disk array performance. For example, in the array that we studied (the HP Surestore E FC-60 [14]), our model could isolate the fact that the adaptive prefetching feature performed well only when the number of sequential streams accessing a disk in the array was small. We expect the

- *E. Varki is with the Computer Science Department, Nesmith Hall, University of New Hampshire, Durham, NH 03824-3591. E-mail: varki@cs.unh.edu.*
- *A. Merchant is with the Storage Systems Department, Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304. E-mail: arif@hpl.hp.com.*
- *J. Xu and X. Qiu are with Falconstor Software Inc., 2 Huntington Quadrangle, Melville, NY 11747. E-mail: {jianzhang.xu, xiaozhou.qiu}@falconstor.com.*

Manuscript received 23 May 2003; revised 29 Dec. 2003; accepted 15 Feb. 2004.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0079-0503.

disk array analysis and model developed here to be useful to designers of disk arrays and to researchers in the area of distributed and parallel computing where programs move large chunks of data between the storage system and main memory. Examples of I/O intensive applications are database management systems, multimedia applications, and large-scale scientific computations. Researchers who want to get maximum performance benefit of their storage system can use our modeling approach to understand how their disk array functions and learn its limitations and performance bottlenecks. For example, if adaptive prefetching at the disk array level performs best when the number of sequential streams accessing the disks is small (as determined for FC-60), it would imply that in applications that submit sequential I/O requests (e.g., multimedia), data should be distributed across the disks such that the number of requests submitted to an individual disk in the array is small. Thus, from a larger perspective, the results of our model (if not the model itself) could be integrated into the performance study of a larger-scope parallel or distributed system.

The rest of the paper is organized as follows: Section 2 presents related work in the area of disk array modeling. Section 3 describes the architecture of the real disk array used in this study. Section 4 presents the general disk array model and the standard techniques employed to evaluate the performance of the model. Section 5 presents a baseline disk array model whose input parameter values may be known a priori (because it is published and made available by the manufacturer) or unknown (in which case we infer it from measurement data). Section 6 shows how the baseline model can be used to extract details of array controller optimizations. The effects of these optimizations are subsequently incorporated in the baseline model. Section 7 discusses interesting observations and challenges that require further research. Section 8 presents the conclusion.

2 Survey of Related Work

A review of the literature on performance analysis of disk arrays reveals an interesting progression of increasing complexity, as researchers attempt to model more features of real disk arrays. The early papers focused on the parallel disks and ignored the array cache. Kim and Tantawi [20] were among the first to present an analytic method for approximating the disk service time of requests striped across n disks. In this early paper, redundancy and queueing of outstanding requests are not considered. Chen and Towsley [8], [9] subsequently incorporated both redundancy and queueing in their performance model of RAID 5 disk arrays in normal mode. Merchant and Yu [25], [26], [27] then analyzed RAID 5 and RAID 1 disk arrays in both normal and recovery modes, which Thomasian and Menon [33], [34] and Kuratti and Sanders [21] extended to normal, degraded, and rebuild modes in their RAID 5 performance model. Bachmat and Schindler [7] analyzed reconstruction in RAID 1 disk arrays. Lee and Katz [22] extended the analysis of disk arrays to include synchronous I/O workloads. DiskSim [13], RaidFrame [11], and Pantheon [40] are some simulation models of disk arrays. Only the disk components of these models have been validated against real disks.

Papers that include the effect of caching policies on disk array performance are relatively scarce compared to the number of papers that have analyzed the effect of multiple disks on array performance. While most caching papers have focused on generating faster, more efficient caching algorithms, there are only two papers that directly analyze the effect of caching mechanisms on disk array performance. Menon [24] models explicit read-ahead and write-back by the array cache of a RAID 5 disk array. Uysal et al. [35] present an analytic model that predicts the mean throughput of disk arrays under asynchronous I/O workloads when the mean queue length at the disk array is known. Interestingly, they incorporate the effect of some controller optimizations whose implementation details are known.

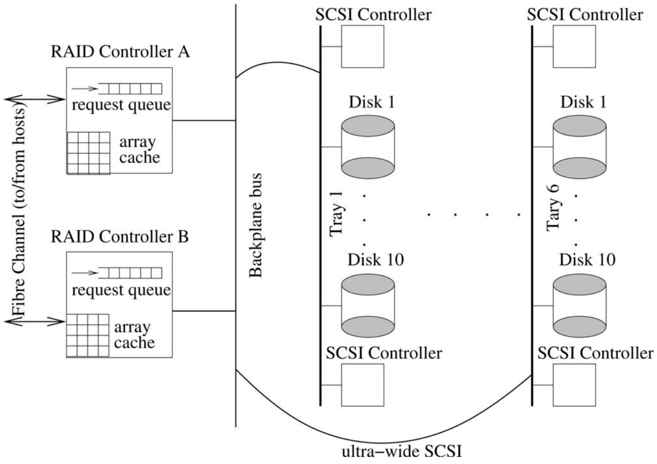


Fig. 1. The HP FC-60 disk array.

We develop an analytical model that analyzes the effects of caching policies and the effects of parallelism of disks along with the effects of array controller optimizations on the performance of a disk array when read-only and write-only workloads are submitted to the array. Our model incorporates caching policies that relax many of the assumptions made by Menon. For example, Menon's model assumes that the read hit rate is known and that the write hit rate is always one. Also, Menon's model does not consider how caching affects the workload distribution submitted to the disks. We extend Uysal et al.'s work to include synchronous workloads which make up a nonnegligible portion of I/O workloads. Further, we do not assume that controller initiated optimizations are known, since manufacturers rarely reveal the details of various features implemented in their system. We show how details of controller optimizations can be extracted from a real disk array using measurement data and baseline disk array models. Finally, compared to the majority of prior work which has employed simulations, we test our model against a real disk array using synthetic workloads. In the next section, we describe the configuration of the real disk array used in our study.

3 Disk Array Architecture

The Hewlett-Packard SureStore E FC-60 disk array [14] is used in our study. This midsize storage system implements several of the features found in typical disk arrays. Fig. 1 shows a representation of the FC-60 disk array. The FC-60 has two array controllers. Each controller has 256 MB of battery backed cache memory (NVRAM). We refer to this battery backed cache memory as an *array cache*. An I/O request that hits in the array cache can be serviced immediately by the cache without forwarding the request to the disks. Thus, an array cache that performs efficient read-ahead and write-behind can have a major impact on the performance of the disk array.

Both array controllers of the FC-60 are connected to a single backplane bus. The backplane bus has six ultrawide SCSI buses, each connected to a separate tray. Each of the six trays has two SCSI disk controllers and up to 10 disks. Each disk has a 4 MB volatile on-board cache memory that can be used to improve disk read access times by prefetching data. Since there are 60 disks in a fully configured FC-60, it is easier to manage the disk array if the disks are logically partitioned into smaller, disjoint groups of disks. Each group of disks is treated as an independent unit and is referred to as a LUN (logical unit). An I/O request is striped across disks belonging to a single LUN. In the FC-60, a LUN is formed by combining disks controlled by different SCSI controllers. For example, a LUN could consist of the first disk on each tray. A typical configuration for the FC-60 is a fully configured array with 60 disks, 10 to each SCSI controller, for a total of 10 6-disk LUNs. Each array controller controls access to a disjoint set of LUNs. However, if one array controller fails, then the other array controller takes over the responsibilities of the failed controller.

The RAID 1/0 configuration is a popular method of disk striping, so we model the FC-60 with its LUNs configured as RAID 1/0 arrays. To protect against failure of any one disk, each disk in a RAID 1/0 configuration is paired with another disk that mirrors its data. All write data have to be written both to the disk and its mirror, and read data can be read from either disk. Each disk in a RAID 1\0 LUN is logically divided into blocks of equal size called *stripe units*. Consecutive stripe units are assigned in round-robin order to the mirrored disk pairs. The set of corresponding stripe units on the disks (e.g., the *third* stripe unit on each disk) forms a *stripe*. Typically, stripe units for consecutive stripes are placed sequentially on a disk (Fig. 2).

All our experiments are run on one FC-60 LUN containing six disks, one from each tray. Table 1 presents the disk array configuration parameters of significance to our work. These parameter values are obtained from the manufacturer's specifications [31] or directly measured.

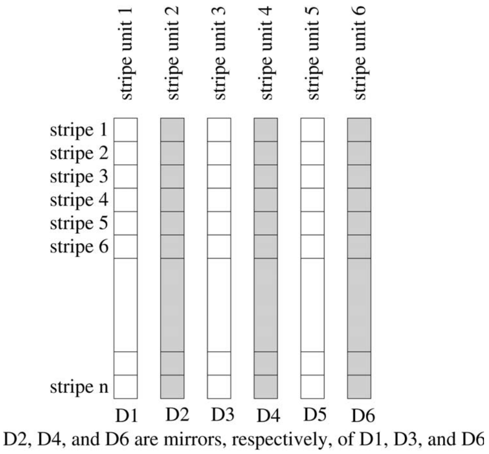


Fig. 2. RAID 1/0 configuration.

4 Disk Array Performance Model

A real disk array is a complex system made up of several components such as array caches, array controllers, disk (SCSI) controllers, disk caches with their own internal caches, and internal buses. If an analytical disk array performance model explicitly models all these components, then it would also be very complex. This level of detail in a disk array model is not only unnecessary but also counterproductive since it would be difficult to extract the behavior of the disk array from all the details. Hence, we use a top-down approach to modeling disk arrays by first identifying the key components and explicitly modeling them. In order to identify the key components of a disk array, it is necessary to understand how the disk array services its workload. All read and write I/O requests to a disk array are first submitted to the array cache. A read request that hits in the array cache is immediately serviced from the array cache, else the request is submitted to the disks. A write request is first written into the array cache and later written to the disks. Based on this analysis, we identify the array cache and the disks as the key components of the disk array.

Our disk array performance model is developed as a network of queueing service centers that each represents a

TABLE 1
Disk Array Characterization

Parameter	Description	Value/Units
cache_size	size of the array cache at each controller	256 MB
cache_transfer_rate	mean array cache transfer rate	62 MB/sec
read_ahead_size	mean number of additional bytes read from disk	$\geq$ 0 bytes
stripe_unit_size	size of a stripe unit	16 KB
stripe_width	number of stripe units in a stripe (logical row)	6
disk_type	type of disks in the FC-60 disk array	Cheetah73
disk_capacity	total formatted capacity of a disk	73.4 GB
disk_read_seek_time	mean disk read seek time	6.05 ms
disk_write_seek_time	mean disk write seek time	6.55 ms
disk_max_read_seek_time	maximum disk read seek time	14.2 ms
disk_max_write_seek_time	maximum disk write seek time	15.2 ms
disk_revolutions	revolutions per minute	10000 rpm
disk_transfer_rate	mean disk transfer rate	31 MB/sec
disk_cache_size	size of disk cache	4 MB

TABLE 2
Workload Characterization

Parameter	Description
M	number of I/O workload streams (*i.e.*, multiprogramming level)
CPU_delay	mean time spent by a job at its CPU/terminal
request_size	mean request size
request_type	read or write requests
run_count	mean number of sequential requests per run [1]
random_count	mean number of random requests between two runs

[1] *A run in a workload stream is a string of sequential requests to contiguous bytes.*

key disk array component, namely, the array cache and the disks. The effects of all other (secondary) components are implicitly captured in the service times of the explicitly modeled components. For example, the bus transfer time is added to the array cache service time, while the disk caching effect is incorporated in the disk service time. During a later modeling phase, a secondary component can be made a key component if it is found that the effect of the secondary component cannot be accurately reflected in the service time of another key component. A secondary component can also be made a key component if one wants to analyze the effect of this secondary component on performance.

Once the key components are identified, the next phase in the modeling process is to characterize the workload submitted to the disk array. Assume that there are M jobs in the system each generating an I/O stream. The key characteristics of an I/O stream are the request type (read or write), the request size (request_size), and the degree of sequentiality. The degree of sequentiality of a workload stream is defined by run_count, which is the mean number of requests to sequential data, and random_count, which is the mean number of random requests between two runs (of sequential requests). We assume that the jobs accessing the disk array generate synchronous I/O requests. This implies that a job must wait until its I/O request completes and there is at most one request from each stream at the disk array. Each of the M jobs spends some time at the CPU/terminal before submitting an I/O request to the disk array. A job waits until the disk array completes processing its I/O request, and then the job spends time at the CPU/terminal before issuing another I/O request. Such cyclical behavior is modeled using a closed queueing network with jobs cycling between the CPU/terminals and the disk array. Hence, we refer to synchronous I/O workloads in the closed system context. Let CPU_delay represents the mean time spent by a job at its CPU/terminal. Since the focus of this study is the behavior of the disk array, the only nonstorage system parameter of interest is the arrival rate of I/O requests. Hence, for simplicity, each job is modeled with its own CPU/terminal and there is no queueing at the CPU/terminal. The workload parameters of significance to our model are summarized in Table 2.

Once the system components and the workload parameters are identified, the next step in the modeling process is to develop the detailed model and the analytical technique used to evaluate the model. The model and techniques for read and write workloads are presented below. Table 3 summarizes the performance measures computed by our performance techniques when there are M workload streams accessing the disk array.

4.1 Read Model

We analyze the behavior of disk arrays when the submitted workload consists of read-only I/O requests. Read requests are first submitted to the disk array cache. With probability cache_hit_probability, a read request's data are found in the cache, and the disk array controller signals service completion. With probability

$$\mathrm{cache\_miss\_probability} = 1 - \mathrm{cache\_hit\_probability},$$

a request is forwarded to the disks, and the disk array controller signals service completion after all disk I/Os for the request complete service. From a performance perspective, the key components of a disk array are the array cache and the disks. The array cache is modeled by a single queueing server with mean service time cache_service_time. The disks are modeled by a parallel queueing system since a request submitted to the disks is divided into one or more disk I/Os issued in parallel to some (or all) of the disks, and the request completes service only after all its disk I/Os complete. The parameter disk_access_probability represents the probability that a request accesses a disk in the array. The parameter disk_service_time represents the mean time to service a disk I/O access. Even if all disk I/Os for a request are issued simultaneously, these disk I/Os need not all start and finish at the same time since the disks are independent devices. The parameter parallel_overhead represents the additional time (over disk_service_time) taken to execute all the disk I/Os of a request.

Fig. 3 presents the queueing network model of the disk array system with read workloads. The MVA technique for

TABLE 3
Performance Technique Outputs for a Disk Array Accessed by M I/O Streams

Performance Measures	Description
array_response_time[M]	mean response time of the disk array
array_throughput[M]	mean throughput of the disk array
array_queue[M]	mean number of I/O requests at the disk array

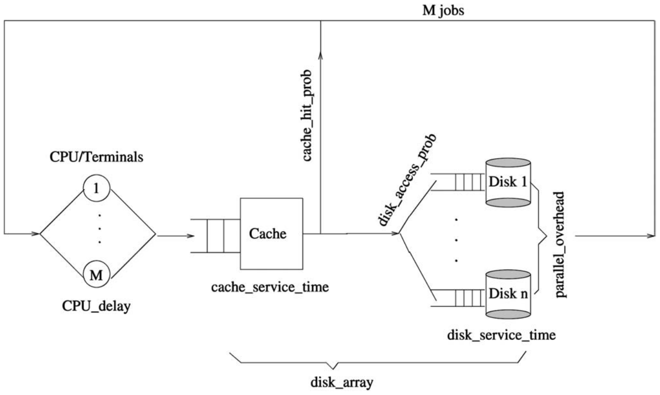


Fig. 3. Queueing network model of disk arrays with read workloads.

parallel queueing networks [36] is the standard technique used to evaluate mean performance measures of closed parallel queueing networks. Appendix A.1 presents the MVA technique using the disk array input parameters presented above. In Section 5, we explain how values for these input parameters are computed.

4.2 Write Model

We analyze the behavior of disk arrays when the submitted workload consists of write-only workloads. Write requests are first written to the disk array cache and completion is signaled as soon as the write-to-cache is completed. The "dirty" data in the cache are eventually written to the disks. The write-back caching policy of a disk array determines when dirty data are written to disks, and most disk arrays implement write-back caching [12], [17].

Our write-back caching model for the FC-60 is determined by two key parameters, namely, the destage_threshold and max_dirty_blocks. The parameter destage_threshold determines the maximum number of dirty blocks that can be held in the cache without triggering disk writes. Therefore, in steady state, there is zero probability that the cache has less than destage_threshold dirty blocks. The parameter max_ dirty_ blocks determines the maximum number of dirty data blocks that can be held in the array cache. Write requests that arrive when the cache is full must wait until enough dirty data are written out to the disks. Thus, there is zero probability that the cache has more than max_dirty_blocks blocks. Let i represent the number of dirty blocks in the cache. In steady state, the cache can have destage_threshold $\leq i \leq$ max_dirty_blocks dirty blocks.

Let λ[M] represent the rate at which dirty blocks arrive at the cache and let μ[M] represent the rate at which dirty blocks are written out to disks when there are M workload streams accessing the disk array. In steady state, the cache can be modeled as a Markov birth-death $M/M/1/K$ process [10], where K = (max_dirty_blocks $-$ destage_threshold $+1$). The corresponding Markov chain shown in Fig. 4 has states {destage_threshold, $\ldots$, max_dirty_blocks}. Appendix A.2 presents the technique used to compute the performance of disk arrays under writes using the input parameters described above. Note that the $M/M/1/K$ model assumes that the interarrival time distribution of dirty blocks to the cache and the service distribution of dirty blocks written to disks are exponential. The advantage of assuming exponential distribution is the efficiency and simplicity of the corresponding performance technique. The exponential assumption may not be strictly satisfied, but during the modeling process, we found that any errors caused by this assumption were, at worst, comparable to other inaccuracies during the modeling process (e.g., errors in measurement data).

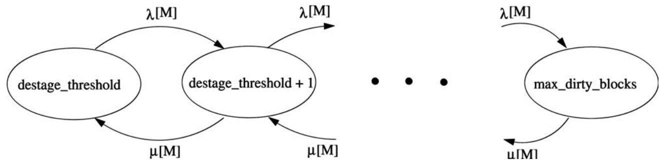


Fig. 4. Markov chain representation of disk arrays with write workloads.

The input parameters for the read model and write model are summarized in Table 4. In the next section, we present the computation of these input parameters.

5 Baseline Model Inputs

In this section, we present the computation of the input parameters to the disk array modeling techniques. The input parameter values (presented in Table 4) must reflect the effects of system and workload specifications and also the effects of various optimizations implemented by the array controller. In the case of system specifications, one can get accurate estimates of the parameter values using measurement data or one can rely on manufacturer published data. For example, the mean bus transfer rate for a disk array can be extracted by taking direct measurements. In the case of controller optimizations, however, it is difficult to understand the implementation details using measurement data since manufacturers rarely specify the optimizations used in an array. Even when particular optimizations are disclosed through patents [15], [29], [38], it is difficult to identify the specific variants that are used in a particular array.

TABLE 4
Model Input Parameters When There are M I/O Streams Accessing the Disk Array

Parameters	Description
	Read Model
disk_service_time[M]	mean disk service time
parallel_overhead[M]	additional time to execute all disk I/Os of a request
disk_access_probability[M]	probability that a request accesses a disk in the array
cache_service_time[M]	mean array cache service time
cache_hit_probability[M]	array cache hit probability
	Write Model
λ[M]	arrival rate of dirty blocks to the cache
μ[M]	rate at which dirty blocks are written from cache to disks

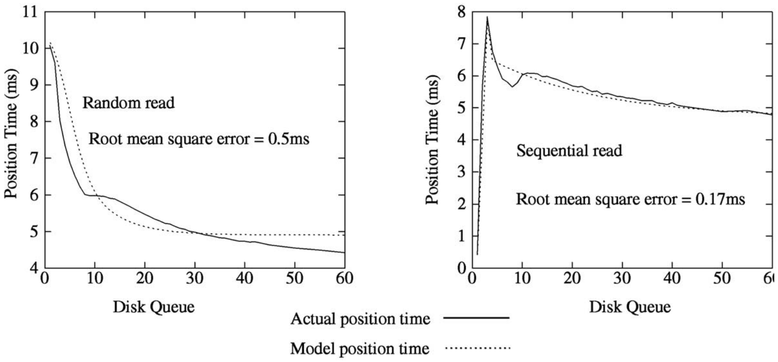


Fig. 5. Disk mean read positioning time for random and sequential workloads.

We address this issue by using a two-step approach to compute the values of the input parameters to our performance model. We first develop a baseline disk array model whose input parameter values only incorporate the effects of "known" disk array features. The algorithms used in implementing these features are either known or can be deduced by taking measurements on the real disk array. In this baseline model, the effects of array controller optimizations are not incorporated, since the details of these "unknown" features are difficult to deduce from just measurement data. Once the baseline model is determined, we compare this baseline model's performance measures against the disk array's performance measures using workloads designed to isolate specific array controller optimizations. We then incorporate the effects of the "unknown" features of the disk array in the input parameter values, resulting in an integrated model of real disk arrays.

The rest of this section presents the issues involved in the computation of input parameter values for the baseline model. Note that for notational convenience, the multiprogramming level M is dropped from parameter representation.

5.1 Disk Service Time

The disks are the slowest component of a disk array, so errors in estimating disk service time would affect the disk array model's performance predictions more than errors in estimating other parameter values. Hence, it is important to get accurate estimates of disk service time, but this is a difficult task since the value of disk service time can vary widely depending on the disk workload, as we explain below.

The disk service time is the sum of seek time, rotational latency, and transfer time. Seek time is the time taken for the disk head to move to the correct disk track, rotational latency is the time taken for the correct disk sector to rotate under the disk head, and transfer time is the time taken to write/read data to/off the disk. The sum of seek time and rotational latency is known as the disk positioning time. It is easy to compute disk transfer time since it depends on the disk I/O request size and the disk transfer rate (disk_transfer_time = diskIO_size/disk_transfer_rate). It is much harder to compute positioning time since this value depends on the distance that the disk head has to move and the distance the disk has to rotate. These distances are dependent on the disk specifications and the disk scheduling policy, and on several workload features such as the arrival rate, distribution, and access pattern of I/O workload streams, the number of workload streams, and the per-stream sequentiality. It is important to select a disk service time computation technique that incorporates the effects of all these features. Several papers [4], [6], [27], [30], [32] present disk service time computation techniques that incorporate the effects of some of these features. For the greatest accuracy, we compute disk positioning time by analyzing disk measurement data. During the measurement process, the disk was disconnected from the disk array.

We model disk positioning time measurement data as a function of disk queue length. The mean rotational latency is typically approximated by the time for half a disk rotation [32], which is constant for a given disk. We approximate the seek time of a disk as $\alpha + \beta\sqrt{seek\_distance}$ by simplifying the seek curve formulas proposed by Ruemmler and Wilkes [30], where α and β are constants depending on the disk. The seek distance depends on the disk scheduling policy and the location of disk I/O requests. Most disk controllers use disk scheduling policies that reduce average seek time, such as SCAN.[1] We assume that when there are disk_queue number of requests from different workload streams at a disk, they are uniformly located over the disk. Since the disk scheduling policy minimizes seek distance, the seek distance is approximately the full-stroke distance (i.e., the distance from one end of the disk to the other end) divided by disk_queue $+ 1$. Thus, the seek time time is given by $\alpha + \beta\sqrt{full\_stroke\_distance/(1 + \mathrm{disk\_queue})}$. The positioning time, which is the sum of seek time and rotational latency, can be written as

$$\mathrm{disk\_position\_time} = a + \frac{b}{\sqrt{1 + \mathrm{disk\_queue}}}, \qquad (1)$$

where a and b incorporate the effects of constants α and β for the disk along with the rotational latency and the full-stroke distance.

The first graph in Fig. 5 plots disk positioning time measurement data as a function of disk queue length when the workload streams are all random. For random workloads, the disk positioning time is given by (1). By minimizing the root-mean square errors between the measured values and those given by the equation, it is determined that for the Cheetah 73 disks used in our experiments, the constants are $a = 3.53$ ms and $b = 8.81$ ms.

The second graph in Fig. 5 plots disk positioning time measurement data as a function of disk queue length when the workload streams are all sequential. When the disk scheduling policy minimizes the seek distance, the disk

[1] In the SCAN scheduling policy, the disk arm starts from one end of the disk and services requests as the arm moves to the other end of the disk.

positioning time depends on the degree of sequentiality of the disk workload. The more sequential the workload, the smaller the mean positioning time. When there is a single stream of sequential requests accessing the disk, the positioning time is close to zero since the disk workload is sequential. As the number of sequential streams increases, the disk workload appears random because of interleaving of requests from different sequential streams, resulting in an increase in positioning time. However, even if there are two or more sequential streams accessing the disk, the request service time can be reduced significantly if a read request can be serviced from the disk cache due to read-ahead [19]. Most modern disk drives have on-board caches that act as speed matching buffers between the disk drive and its interface and as read-ahead stores of sequential request data. The disk cache is divided into segments, and the amount of read-ahead is dependent on the size of a segment and the disk queue length. During a read operation, the disk drive typically reads an entire segment's worth of data into an available cache segment (not just the request data) if there are no other requests waiting in the disk queue. The graphs in Fig. 5 incorporate the effect of disk caching by computing disk positioning time as the average of positioning time values when there are cache hits and misses. As the second graph indicates, the sequentiality of the workload streams has minimal effect on disk service time for disk_queue > 3. (That is, disk caching is effective when there are less than four sequential streams accessing the disk.) This result could be because there are four or more sequential streams accessing the disk when there are only three disk cache segments in the disks used in our study. In fact, the default number of cache segments for these disks is three, although they can be set to vary between 1 and 16. Thus, read-ahead data are possibly being replaced before they result in read-ahead hits. For simplicity, we model the positioning time for disk_queue ≤ 3 as a linear function of the disk queue length. For disk_queue > 3, (1) is used. For sequential workloads, the mean positioning time is given by

$$\text{disk\_position\_time} = \begin{cases} (1) & \text{disk\_queue} > 3 \\ c + d * \text{disk\_queue} & \text{disk\_queue} \leq 3. \end{cases} \tag{2}$$

For Cheetah 73, $c = -2.73$ and $d = 3.68$.

Given a particular disk and its workload, the disk service time is a function of the disk queue length. However, disk queue length is an output of the disk array performance model. Thus, disk service time is a function of disk queue length, but disk queue length is a function of disk service time. To address this circular relationship between disk service time and disk queue length, disk service time computation techniques typically assume that the disk queue length is a known input parameter. In our modeling study, disk queue length is a known input parameter only when CPU_delay $= 0$ in which case disk_queue = M $*$ disk_access_probability. For workloads with CPU_delay > 0, a reasonably accurate approximate value of disk queue length must be computed and validated. Our computation of the approximate disk queue length is presented in Appendix A.3. The validation of the approximate disk queue length is presented in a technical report [37].

5.2 Parallelism Overhead

Parallelism overhead (in Table 4) refers to the additional time (over disk_service_time) taken to execute all the disk I/Os of a request. Since all sibling disk I/Os are the same size, the transfer time for each disk I/O is the same, but the positioning time for each disk I/O could be different since positioning time depends on several disk and workload parameters as explained above. Let max_position_time represent the mean of the maximum positioning time from among num_diskIOs_per_request disk I/Os.

$$\text{parallel\_overhead} = \text{max\_position\_time} - \text{disk\_position\_time}.$$

The value of max_position_time is dependent on the distribution of positioning time, but the exact distribution of positioning time is unknown. So, one has to approximate the distribution of positioning time. We compute parallel overhead by assuming positioning times are modeled by the beta distribution with values ranging from [0, full_ rotation_time + disk_max_read_seek_time] [37]. Since parallelism overhead is a small factor in the performance prediction computation (see Appendix A.1), errors in the model's performance predictions due to inaccuracies in the value of parallelism overhead are within the bounds of other errors inherent in the modeling process.

5.3 Disk Access Probability

The disk_access_probability represents the probability that a request accesses data from a disk in the array. The value of disk_access_probability depends on the cache hit probability, the I/O workload's disk access pattern, and on array controller optimizations such as access coalescing. In this baseline model, we ignore the effects of array controller optimizations, but in the next section, we will address the impact of these optimizations on the value of disk_access_probability. It is assumed that requests access data uniformly from all the disks. Hence, the value of disk_access_probability depends on the cache hit probability and the distribution of the number of disk I/Os per request for the I/O workload.

$$\begin{aligned} &\text{disk\_access\_probability} = \\ &\text{cache\_miss\_probability} * \frac{\text{num\_diskIOs\_per\_request}}{\text{stripe\_width}}. \end{aligned} \tag{3}$$

5.4 Cache Parameters

A cache hit on a read request occurs if 1) this request is part of a sequential stream of requests submitted to the disk array and was read into the cache as part of read-ahead data, or 2) this request had been referenced in the past and the request's data are still in the cache. The random variables representing read-ahead and rereference hits are independent since the probability that a request's data results in a read-ahead hit is not related to whether this request's data results in a rereference hit. The cache hit probability is then given by

$$\begin{aligned} \text{cache\_hit\_probability} =& 1 - (\text{read\_ahead\_miss\_probability} * \\ & \text{re\_reference\_miss\_probability}). \end{aligned}$$

Techniques for computing the rereference probability are presented in several papers [5], [32], [35], [42]. The read-ahead probability is a function of explicit read-ahead (i.e., every read access from the disks results in an additional system-defined number of bytes being read into the cache) and adaptive prefetching based on detection of I/O sequentiality. A technique for computing the explicit read-ahead hit rate is given in a technical report [37]. We will address adaptive prefetching in the next section.

TABLE 5
Inputs to the Synthetic Workload Generator

Parameter	Values
M	1 to 12
CPU_delay	exponentially distributed with mean 0ms (no delay), 10ms, 30ms, 100ms
request_size	4KB, 8KB, 16KB, 32KB, 48KB, 64KB, 128KB, 256KB
request_type	read-only, write-only
run_count	1 for random requests, 64 for sequential requests
random_count	0 (*i.e.*, a workload stream consists of runs of length run_count)

The cache service time is the rate at which requests are transferred from the array cache to the main system and is equal to request_size/cache_transfer_rate.

5.5 Write Model Input Parameters

The parameter μ represents the rate at which dirty blocks are written out to the disks from the array cache. Once the destage_threshold is reached, the controller writes out data to all disks at a time. Since data written to a disk must also be written to the disk's copy,

$$\mu = \frac{\mathrm{stripe\_width}}{2 * \mathrm{disk\_service\_time}}.$$

The parameter λ represents the rate at which dirty blocks arrive at the write cache when it is not full (i.e., when the number of dirty blocks in the cache is less than max_dirty_blocks). In this state, the rate of arrival of dirty blocks is constrained mainly by the multiprogramming level and the CPU delay of the workload. We can therefore approximate λ by max_array_throughput, the throughput of the disk array system with an infinite write cache under this workload. The MVA technique can be used to compute max_array_throughput of a disk array with an infinitely large array cache.

6 Extracting Controller Optimizations

Most array controllers implement the following optimizations:

1. access coalescing policy that determines whether several disk I/Os that access contiguous data on the same disk are coalesced into a single disk I/O [15], [16],
2. redundancy-based load balancing policy that determines how the disk controller distributes the load between a disk and its mirror [18], [38], and
3. adaptive data prefetching from the disks to the array cache based on detection of I/O workload sequentiality [29], [39], [43].

The baseline model developed in the last section does not incorporate the effects of these optimizations.

Here, we show how a validated baseline model and a carefully crafted workload can be used to isolate and understand the details of each controller optimization. The simplest baseline disk array model is one that models disk arrays under small random read requests (which access only 1 disk in the array) with CPU_delay $= 0$. In this case, the only input parameters of significance are disk service time and cache service time. This model can be validated against measurement data from the disk array. This baseline model can be extended by incorporating the effects of one feature at a time, and by validating the model against measurements from the actual system at each step. We use this incremental approach to incorporate the effects of each of the controller initiated optimizations. Before presenting details of the extraction of controller optimizations, we present the workload used in this study.

6.1 Experimental Setup

An HP server generates the synthetic I/O workloads that are used to extract the disk controller details. Unlike real workload traces, synthetic workloads can be characterized accurately, so we can eliminate errors in our model's performance predictions that arise due to inaccuracies in workload characterization. The inputs to the synthetic workload generator are presented in Table 5.

In order to analyze the disk array's performance under synthetic workloads, a trace of all I/O activity at the device driver level is collected. The trace contains I/O submission and completion times, the logical addresses and sizes of requests, and all other relevant workload information. An analyzing tool then analyzes the trace file and generates statistical results such as mean values, variances, and 95 percent confidence intervals for response times and throughputs. We ran each experiment until the 95 percent confidence interval for each metric was less than 4 percent of the point value.

6.2 Access Coalescing and Load Balancing Policies

The access coalescing policy determines whether several disk I/Os that access contiguous data on the same disk are coalesced into a single disk I/O. For example, consider a request of size 96 KB striped across three disks when the stripe unit size is 16 KB. In this case, the request's data are stored on two adjoining stripes (rows) of the three disks, and the disk controller has to read two stripe units from each disk in order to access all the request's data. The access coalescing policy determines whether the two individual disk I/Os to each disk are coalesced into a single disk I/O. The redundancy-based load balancing policy determines how the disk controller distributes the load between a disk and its mirror. For example, consider a request of size of 96 KB striped across six disks configured using RAID 1/0, where each disk has a mirror. As explained above, the request's data are stored on two adjoining stripes. Since each disk has a mirror, the controller can either read all the data off three disks or the controller can distribute the load between disks and their mirrors by submitting read requests to all the six disks. That is, one disk I/O can be submitted to all the six disks instead of submitting two disk I/Os (or one coalesced disk I/O) to three of the six disks. The advantage of distributing the request's load across all six disks is that the disk service time for the request can be

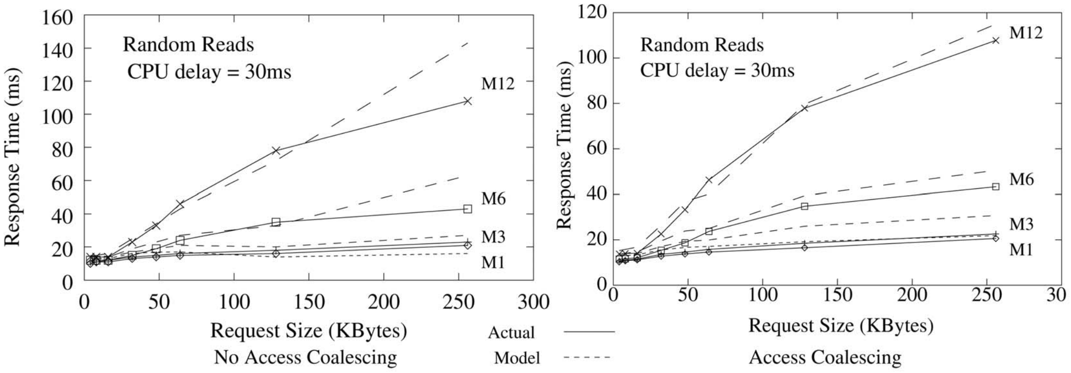


Fig. 6. Isolating the effect of the access coalescing policy.

reduced by reducing the transfer time. The disadvantage of distributing the request's load across all six disks is that the queue length at each disk increases. For example, consider two requests of size 96 KB submitted to the disk array. If both requests are read off all six disks, then each disk has a queue length of two. On the other hand, if each request is read off three disks, then the disk controller can read one request off three disks and the other request off the other three disks, thus ensuring that the queue length of each disk is one. We refer to the load balancing policy that balances load by dividing each large request's data read access between a disk and its mirror as the *disk-transfer-time-reducing* load balancing policy. The policy that balances load by assigning separate requests to a disk and its mirror is referred to as the *disk-queue-length-reducing* load balancing policy.

The access coalescing and load balancing policies together determine the distribution of the number of disk I/Os per request. For example, consider large request sizes $\geq$ stripe_width $*$ stripe_unit_size. Assume that access coalescing occurs for such large requests. Then, depending on the load balancing policy, num_diskIOs_per_request = stripe_width if request data access is distributed between a disk and its mirror (disk-transfer-time-reducing policy), else num_diskIOs_per_request = stripe_width/2 (disk-queue-length-reducing policy). As (3) (in Section 5.3) shows, the disk access probability for a given workload is dependent on num_diskIOs_per_request. So, disk_access_probability is a function of the load balancing and the access coalescing policies of a disk array.

In order to extract details of the access coalescing and load balancing policies for the FC-60 disk array, we incorporate different combinations of access coalescing and load balancing policies in the validated baseline disk array model. In particular, we develop four updated model versions by separately incorporating each of the following policies in the baseline model:

1. no access coalescing and disk-queue-length-reducing policy,
2. no access coalescing and disk-transfer-time-reducing policy,
3. access coalescing and disk-queue-length-reducing policy, and
4. access coalescing and disk-transfer-time-reducing policy.

The workload consists of random read-only requests with large request sizes and varying multiprogramming levels. By using random workloads, one can ensure that array caching has minimal impact on performance of the disk array. By using large requests (greater than stripe_width/2) one can evaluate both the effects of access coalescing and redundancy-based load balancing. We then compare each updated model's predictions against actual measurements, and by the process of elimination, select the model that best matches the system.

To evaluate whether access coalescing occurs, we compare the performance of access coalescing and no-access coalescing models against actual measurements. Fig. 6 shows two graphs that compare the mean response times of the actual system against model response times. The first graph plots model predictions when the model does not perform access coalescing, while the second graph plots model predictions when the model performs access coalescing. We deduce that access coalescing is used in the actual system because of the close match of the actual and model curves in the right-hand graph.

We use a similar approach to evaluate the redundancy-based load balancing policy which determines how the FC-60 distributes load between a disk and its mirror. We compare the performance of the access coalescing, disk-transfer-time-reducing load balancing model and the access coalescing, disk-queue-length-reducing load balancing model against actual measurements. Fig. 7 shows two graphs that compare the mean response times of the actual system against model response times. In the first graph, the model predictions reflect disk-queue-length-reducing load balancing, while in the second graph, the model predictions reflect disk-transfer-time-reducing load balancing. The graphs indicate that FC-60 performs disk-transfer-time-reducing load balancing. The graphs show that for the given workload consisting of all same-sized requests, disk-transfer-time-reducing load balancing policy lowers performance significantly as the multiprogramming level increases. This implies that for the tested workloads, the performance degradation due to increasing disk queue lengths is not offset by the performance improvement due to reduction in disk service times (by reducing disk transfer times).

Our analysis shows that the FC-60 performs access coalescing and disk-transfer-time-reducing load balancing. We then incorporate the effects of these two policies in our baseline model as follows:

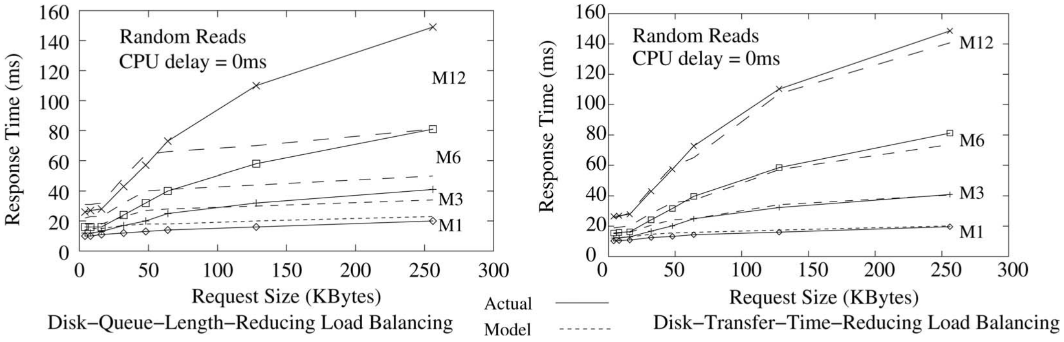


Fig. 7. Isolating the effect of the load balancing policy.

$$\text{num\_diskIOs\_per\_request} = \begin{cases} \text{stripe\_width} & \text{if request\_size} \geq \text{stripe\_width}* \\ & \text{stripe\_unit\_size} \\ \left\lceil \frac{\text{request\_size}}{\text{stripe\_unit\_size}} \right\rceil & \text{otherwise.} \end{cases}$$

6.3 Adaptive Prefetching Policy

Current disk arrays typically implement adaptive prefetching of data based on automatic detection of sequential I/O streams. In order to understand how adaptive prefetching is implemented, we again compare our model predictions against actual measurements. We use our baseline disk array model now updated with the incorporation of the effects of load balancing and access coalescing policies. The workload consists of sequential read requests. As explained in Section 5.4, read hits at an array cache occur because of explicit and adaptive read-ahead by the array cache and by rereferencing of data currently in the array cache. In order to isolate the effects of the adaptive prefetching policy, we turn off explicit read-ahead by the array cache, thus ensuring that explicit read-ahead hit rate is 0. We also ensure rereference hit rate is 0 by ensuring that the workload contains no rereferences.

Since the FC-60 performs adaptive prefetching and our model does not, one would expect the actual system performance to be better than the model predictions. This improvement in system performance over model performance would indicate the effectiveness of FC-60's adaptive prefetching policy. Fig. 8 plots the model mean response times and the actual response times for sequential read workloads for varying request sizes, multiprogramming levels 1, 3, 6, and 12, and CPU delay 0 ms, 10 ms, 30 ms, and 100 ms. Note that, for a given multiprogramming level, as the CPU delay increases, the disk queue length decreases. For example, if a request accesses all the disks, the disk queue length equals M when CPU delay is 0 ms, while the disk queue length is less than M when CPU delay is greater than 0 ms. Referring to Fig. 8, for CPU delay of 0 ms, the disk array outperforms the model by a small margin for $M = 1, 3$ (disk_queue < 4). As CPU delay increases, the disk array outperforms the model at additional multiprogramming levels, as long as disk_queue < 4. For CPU delay of 100 ms, the disk array outperforms the model for all multiprogramming levels plotted, since disk_queue < 4. Thus, our analysis indicates that for the workloads considered, the FC-60 adaptive prefetching policy is effective when disk_queue < 4.

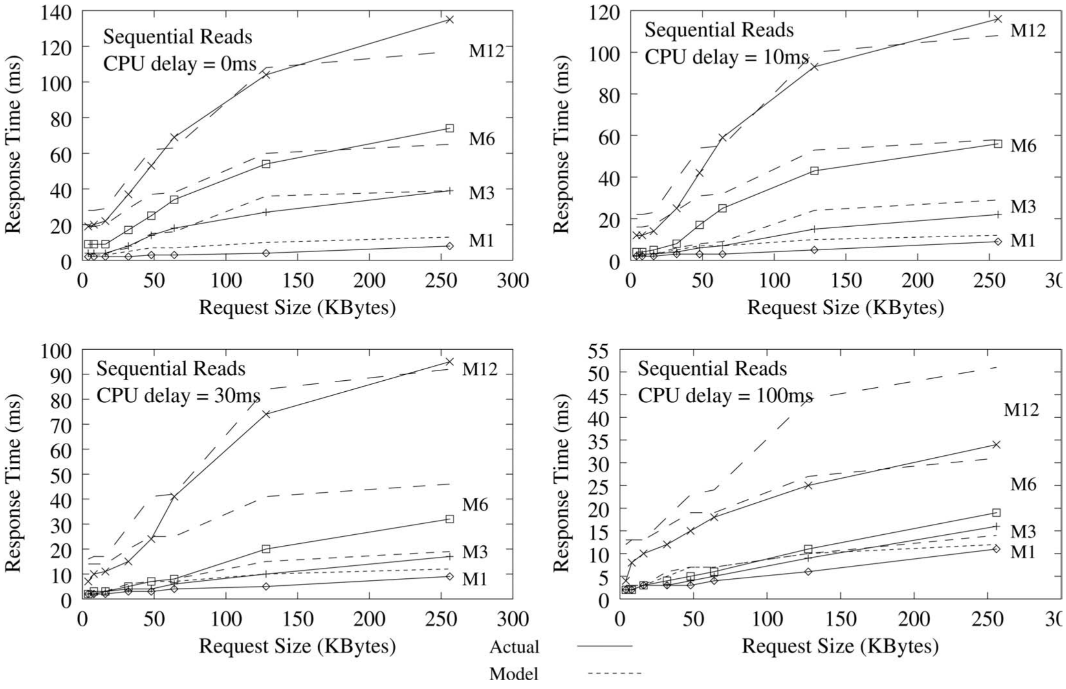


Fig. 8. Isolating the effect of the adaptive prefetching policy.

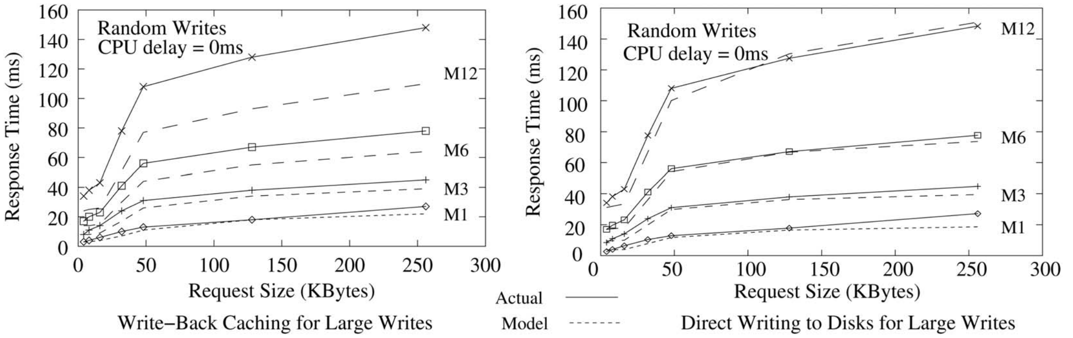


Fig. 9. Extracting details of the write-back policy.

Our model isolates the effects of adaptive prefetching in the FC-60 disk array. In order to incorporate the effects of adaptive read-ahead, one has to understand the details of when prefetching occurs and also the adaptive read-ahead data size. We were unable to extract these details even through our reverse engineering approach, so adaptive prefetching is not implemented in our disk array model.

6.4 Write-Back Caching Policy

The write-back caching policy details for FC-60 are presented in Section 4.2. When comparing our write baseline model performance measures against system performance measures, we determined that write-back caching in the FC-60 occurs only for write requests of size at most equal to two stripes. As Fig. 9 indicates, the write policy determination is made because our write-back caching model outperforms the system by a large margin while a direct-write model matches the system closely. In the direct-write model, large write requests larger than two stripes are written directly to the disks bypassing the array cache. The disk array controller signals service completion of a large write request after all its disk I/Os complete. The performance of disk arrays with large write workloads can be computed using the technique for evaluating the disk array model with read requests (Section 4.1). The model parameterization for large writes is relatively straightforward and the details are presented in a technical report [37].

6.5 Validation

In addition to the optimizations mentioned above, a disk array controller may implement other optimizations. In addition, there are overheads involved with the operations of the controller. Without any knowledge of what these optimizations and overheads are, it is difficult to select an appropriate workload that can isolate the effect of a particular optimization or overhead. So, we compare our model performance predictions against the system performance measurements to check whether it is possible to extract information about the combined effects of other unknown optimizations and overheads on the overall performance of the disk array.

Figs. 10 and 11 plot the model mean response times and the actual response times for read and write workloads for varying request sizes and multiprogramming levels, and CPU delay of 0 ms and 10 ms. Fig. 12 plots model and actual throughputs for varying request sizes

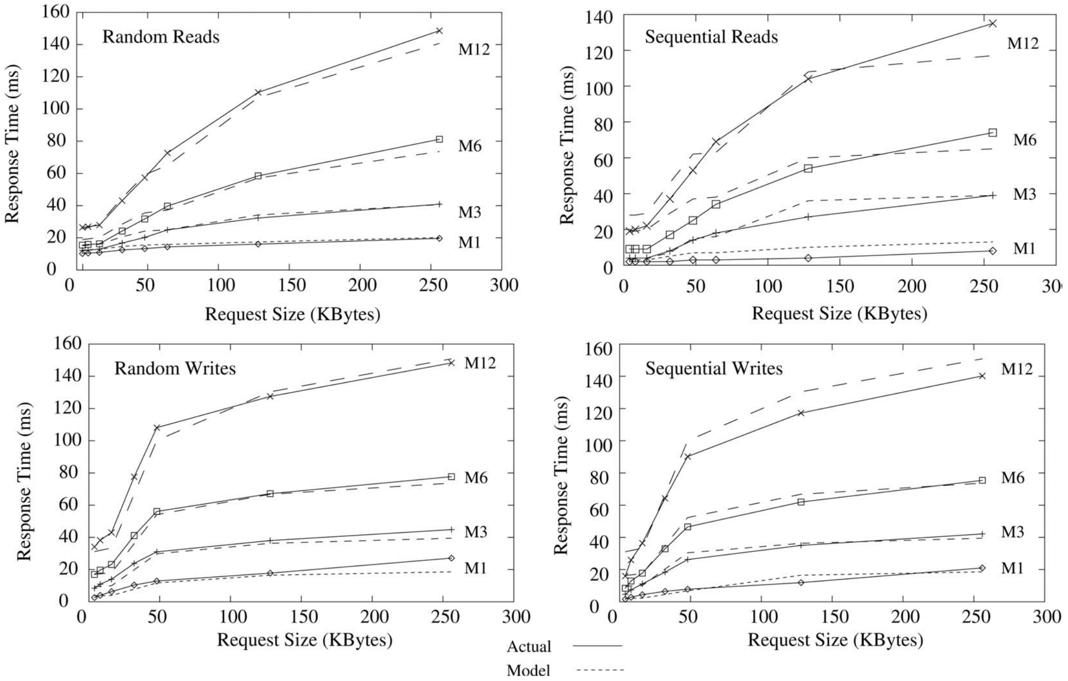


Fig. 10. Model versus actual response times for CPU_delay $= 0$ ms.

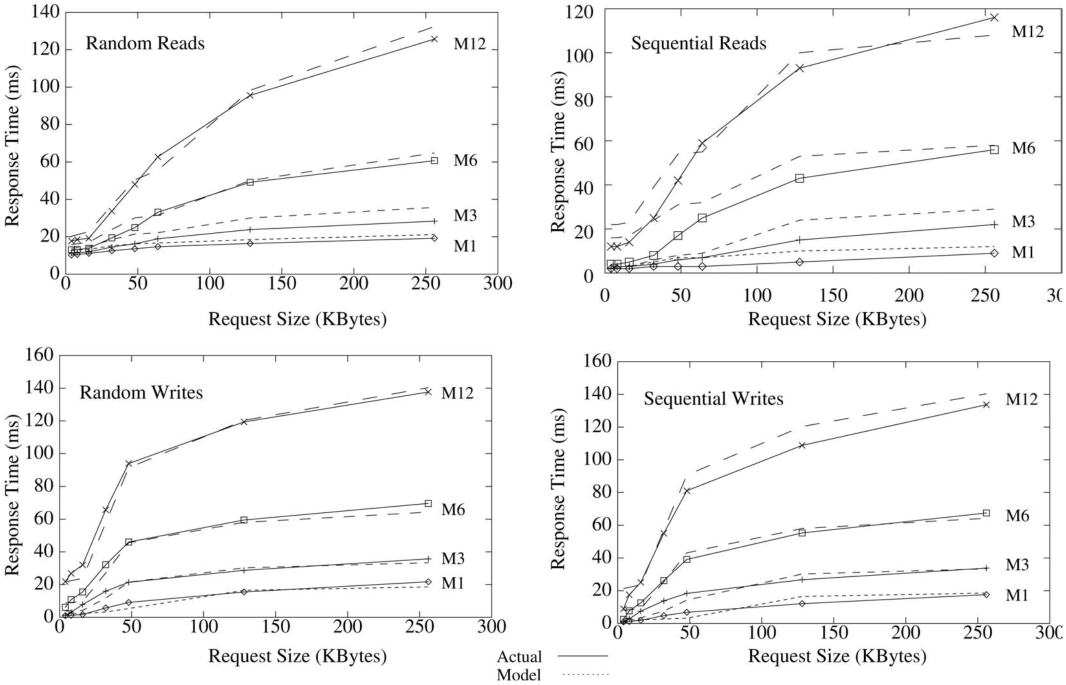


Fig. 11. Model versus actual response times for CPU_delay $= 10$ ms.

and multiprogramming levels, and CPU delay of 30 ms. These graphs indicate that the model's performance predictions are a good match for the experimental performance measures. The model is better for random I/Os than for sequential I/Os, and better for reads than for writes. The average errors range from 3 ms (for random I/Os) to 5 ms (sequential I/Os). Across all workloads and experiment sets, the accuracy of the model is within 7.1 percent on average. For random reads, the model is accurate within 3.7 percent, on average, and for random writes, the model is accurate within 5.5 percent, on average. For sequential workloads, the model predictions are, on average, within 8.8 percent and 8.0 percent for read and write workloads respectively. For FC-60 under the tested workloads, the graphs indicate that our model adequately incorporates the effects of the various controller optimizations and overheads.

6.6 Further Analysis

Our results indicate that the disk array's performance under random and sequential workloads is quite similar, which indicates that the sequentiality of the workload has little impact on performance when the disk queue length is 4 or more. This result is not very surprising given the similarity of disk service time values for random and sequential workloads, and the earlier analysis indicating that FC-60's adaptive prefetching algorithms perform better for small disk queue length values.

Our results indicate that the disk array's performance under read and write workloads is quite similar. Given that large writes are written directly to disks and given that most read requests result in cache misses, this result is not surprising. However, the performance under small random reads is equivalent to the performance under small writes when there is a high probability that an arriving request finds the cache full. This result is a little surprising since one would expect the performance with write-back caching to be better than direct writes (or random reads) even when the cache is mostly full, because the array controller can judicially pick dirty data from the full cache to write out to the disks. This implies that the algorithm that selects dirty cache data to write to the disks can be improved.

7 Discussion

In this paper, we evaluate how well standard performance techniques can be used to analyze real disk arrays whose internal details may not be known. In contrast to the purely measurement-based table approach[2] used by the Hippodrome project [2], [3], our modeling approach provides insights into how various features of a disk array affect the performance of the array. Our analytic approach, although developed and validated for the FC-60 disk array (a midsize storage system), could be used to compute the performance of other vendors' storage systems, since the modeling techniques employed are not specific to FC-60. Also, there is considerable flexibility allowed by our modeling approach. For example, if one wants to analyze the effects of disk caching on disk array performance, then one could model the disk cache as a separate service center rather than incorporating caching effects in disk service time.

Our study of the interaction between array caches and disks and their joint impact on the performance of disk arrays indicates that array caching policies affect not only the cache hit rate, but also the disk service time and disk access probability by changing the distribution of I/O workload. Similarly, disk parameters such as disk queue length affect not only the disk service time, but also the array caching policies. For example, our study indicates that for the FC-60, adaptive prefetching caching policy is most effective for disk_queue < 4.

[2] The table-based approach is based on taking performance measurements of a disk array under synthetic workloads. Performance measurements which were not measured are found by interpolation.

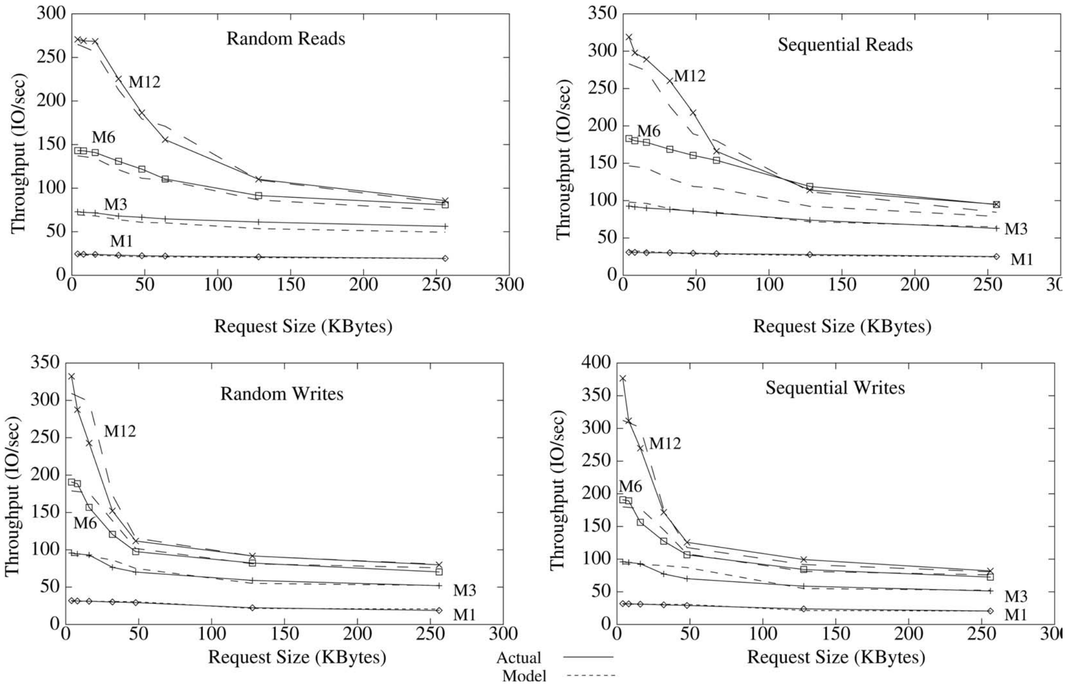


Fig. 12. Model versus actual throughput for CPU_delay $= 30$ ms.

Given our objective of identifying the issues and challenges to performance modeling of disk arrays, we believe more work is needed on the following issues:

1. We found considerable paucity of data about the internal specifications of disk arrays, despite a study of manufacturer manuals and other published information such as the US patent Web site where information on patented algorithms is available. Too often we found it difficult, if not impossible, to associate a patented algorithm with a specific disk array. Also, in the instances where published data was available (e.g., bus transfer rate), the parameters seem based on performance under best-case conditions and do not reflect the effect of traffic at the device. For example, for the workloads tested, we found that the bus transfer rate presented in the manufacturer's specifications was about 35 percent greater than the measured values. In the absence of such information, we had to rely on measurement data and carefully designed baseline models to understand the implementation details of various features. It would be useful if manufacturers publish the exact conditions under which their published rates can be reproduced or, alternatively, publish both best case and worst case scenarios.
2. Several papers analyze the performance of disk arrays as a function of the disk service time, the stripe unit size, and stripe width [8], [9], [20], [21], [22], [25], [26], [27], [33], [34]. Our study indicates that, in addition to the features mentioned above, a disk array's performance depends on disk controller optimizations, array caching policies, and workload distribution. It is important to understand how all these features together impact the performance of a disk array for different workloads. For example, in our study, the disk-transfer-time-reducing load balancing policy performed poorly against the disk-queue-length-reducing load balancing policy. This could be because all requests were the same size in our study. Another related aspect that requires further study is the difficulty of configuring a disk array to meet the often conflicting requirements of all applications accessing the disk array. For example, applications that submit large I/O requests would benefit from policies that distribute each request's load across all disks thereby reducing the time taken to service the requests, while applications that submit several small requests would benefit from policies that distribute individual requests to separate disks, thereby increasing request throughput of the disk array. Designing a storage system that balances the requirements of all applications accessing the device requires a thorough understanding of how the various system and workload parameters impact on the response time and throughput of the disk array.
3. Our study indicates that write-back caching improves performance over direct writing to disks when there is a low probability that an arriving request finds the cache full. This result is a little surprising since a disk array controller can judicially select dirty data from the full cache of dirty data to write to the disks, thereby improving performance over a system that writes each incoming write request directly to the disks. Thus, there is scope for improving the algorithm used by FC-60 to determine which data are to be written out to the disks from the array cache.
4. The FC-60 has two 256 MB array caches, and each disk within the disk array has a 4 MB cache. Our study indicates that for the FC-60 disk array, per-stream sequentiality has little impact on performance if there are four or more sequential streams accessing a disk at a time. This indicates that for the FC-60 disk array, adaptive prefetch hit rates are low

at high multiprogramming levels. Previous studies [28], [41] have found that disk array cache rereference hit rates are typically low. Since the FC-60 is a typical example of a mid-size disk array, this raises the question as to whether the disk array read caches are being optimally used.

5. A real disk array is a dynamic system since several of its policies such as the adaptive prefetching cache policy, the load balancing policy, etc., are dependent on the disk queue length. However, the disk queue length is dependent on the policies and parameter values of the disk array and its workload. This recursive dependence between the inputs and outputs of a disk array's performance model represents one of the most challenging problems in the performance analysis of real disk arrays. In our paper, we address the dependence of disk service time on disk queue length by computing an approximate disk queue length value (see Appendix A.3). Clearly, more work is needed on resolving the circularity involving disk queue length and the inputs of a disk array model.

Finally, these findings need to be evaluated in the context of this study, which was limited by reasons of access to the analysis of only one real disk array architecture, namely, the FC-60 disk array. Manufacturers typically do not allow the analysis and publication of the performance results of their systems, but we were fortunate that HP Labs allowed us to study the FC-60 disk array and publish our results. Consequently, some of the problems and issues identified above may be specific to the FC-60 array. We would strongly encourage researchers with access to real disk arrays to verify whether the issues we have identified above are reproduced in other proprietary systems. Also, since we were keen to understand how well standard performance techniques modeled the various "unknown" optimizations within real disk arrays, simple synthetic workloads were used. These workloads consisted of read-only and write-only streams with all I/O requests of same size. Consequently, all performance predictions in this paper are necessarily restricted to the simple workloads studied in this paper.

8 Conclusion

We show how standard performance techniques can be used to develop a disk array performance model that incorporates the system effects of caching, multiple disks, and various controller optimizations, in addition to the workload effects of sequentiality, concurrency, and CPU delay. Our modeling approach is useful in determining the performance of a real disk array whose internal algorithms may not be completely known. Researchers in academia, who often do not have access to all internal algorithms implemented in their storage system, can use our approach to develop reasonably accurate models of their storage system and understand how their storage system functions. Storage analysts in industry, with access to the internal algorithms implemented in their storage system, could also use our approach to quickly develop models that isolate the effects of different features of their system on the performance of their disk array. Thus, our model provides them with a "quick and dirty" way to isolate potential problem areas, with comparatively little effort. Our model would be useful in sensitivity analysis studies that analyze the impact of various parameters and features on a disk array's performance. For example, our model can be used to analyze the impact of optimizations such as access coalescing and load balancing.

Our study identifies several issues and challenges of disk array performance modeling that need further analysis. While the disk array model developed here is more detailed than previous disk array models, there is much more work to do. We analyzed our disk array under synthetic read-only and write-only workloads, because we wanted to keep things reasonably simple in our initial analysis of the issues and challenges of performance modeling of "black-box" (real) disk arrays. Hence, our model does not analyze the impact that read-write workloads have on the performance of disk arrays. In future work, we intend to "muddy the waters" a bit more and study disk arrays under read-write synthetic workloads and real workloads.

Appendix A

A.1 Performance Modeling Technique for Reads

We show how the parallel MVA technique is used to compute the mean response time, throughput, and queue length of a disk array under read workloads. For multiprogramming level m varying from 1 to M, iteratively compute

1.
$$\begin{aligned}\text{array\_response\_time}[m] = {}& \text{cache\_response\_time}[m] + \\ & \text{disks\_response\_time}[m],\end{aligned}$$

where

$$\begin{aligned}\text{cache\_response\_time}[m] &= \text{cache\_service\_time}[\text{M}] \\ &\quad * (1 + \text{cache\_queue}[m-1]) \\ \text{disks\_response\_time}[m] &\approx \text{cache\_miss\_probability}[\text{M}] \\ &\quad * (\text{parallel\_overhead}[\text{M}] \\ &\quad + \text{disk\_service\_time}[\text{M}]) \\ &\quad + (\text{disk\_access\_probability}[\text{M}] \\ &\quad * \text{disk\_service\_time}[\text{M}] \\ &\quad * \text{disks\_queue}[m-1]).\end{aligned}$$

The parameter cache_queue represents the mean number of requests at the array cache. The parameter disks_queue represents the mean number of requests at the disks. For $m = 1$,

$$\text{cache\_queue}[m-1] = \text{cache\_queue}[0] = 0$$

and

$$\text{disks\_queue}[m-1] = \text{disks\_queue}[0] = 0.$$

For $m > 1$, the computation of cache_queue and disks_queue is presented in Step 3 of this algorithm.

2.
$$\text{array\_throughput}[m] = \frac{m}{\text{CPU\_delay} + \text{array\_response\_time}[m]}.$$

3. The queue lengths at the disks and the cache are computed using Little's Law,

$$\begin{aligned}\text{disks\_queue}[m] &= \text{disks\_response\_time}[m] \\ &\quad * \text{array\_throughput}[m] \\ \text{cache\_queue}[m] &= \text{cache\_response\_time}[m] \\ &\quad * \text{array\_throughput}[m].\end{aligned}$$

The disk array queue length represents the total number of outstanding requests at the array cache and disks.

$$\text{array\_queue}[m] = \text{disks\_queue}[m] + \text{cache\_queue}[m].$$

A.2 Performance Modeling Technique for Writes

The disk array under write workloads is modeled using the $M/M/1/\mathsf{K}$ queue, and here, we show how the disk array's mean performance measures are computed. Let $\rho = \lambda[\mathrm{M}]/\mu[\mathrm{M}]$ represent the utilization of the $M/M/1/\mathsf{K}$ queue. Then, the steady state probability distribution of dirty blocks in the cache is [10]

$$P_i = \begin{cases} \frac{(1-\rho)*\rho^{(i\,-\,\text{destage\_threshhold})}}{(1-\rho^{\mathrm{K}+1})} & \text{for destage\_threshold} \leq i \leq \\ & \quad \text{max\_dirty\_blocks} \\ 0 & \text{otherwise.} \end{cases}$$

Each job submits write requests that are equivalent to one or more dirty blocks. Let number_of_dirty_blocks represent the number of dirty blocks per I/O request.

$$\text{number\_of\_dirty\_blocks} = \left\lceil \frac{\text{request\_size}}{\text{stripe\_unit\_size}} \right\rceil.$$

The mean throughput (i.e., requests serviced per unit time) of the disk array is computed from the $M/M/1/\mathsf{K}$ queueing model

$$\text{array\_throughput}[\mathrm{M}] = \frac{\lambda[\mathrm{M}] * (1 - P_{\text{max\_dirty\_blocks}})}{\text{number\_of\_dirty\_blocks}}.$$

The mean response time is computed using Little's Law on the entire system.

$$\text{array\_response\_time}[\mathrm{M}] = \frac{\mathrm{M}}{\text{array\_throughput}[\mathrm{M}]} - \text{CPU\_delay}.$$

The disk array queue length is computed using Little's Law on the disk array.

$$\begin{aligned}\text{array\_queue}[\mathrm{M}] = {} & \text{array\_throughput}[\mathrm{M}] * \\ & \text{array\_response\_time}[\mathrm{M}].\end{aligned}$$

A.3 Computation of Approximate Disk Queue Length

If there are M jobs, then in the worst case, all the jobs are at the disks. Since the probability of a job being at any particular disk in the array is disk_access_probability, $\text{disk\_queue} = \mathrm{M} * \text{disk\_access\_probability}$. Hence, disk_position_time (and, subsequently, disk_service_time) for this disk queue length can be computed using (1) or (2) presented in Section 5.1. Then, the disk response time for this worst case is given by

$$\begin{aligned}\text{disk\_response\_time} = {} & \text{disk\_service\_time} * \mathrm{M} \\ & * \text{disk\_access\_probability}.\end{aligned}$$

A job spends CPU_delay at the CPU and, in the worst case,

$$\text{cache\_response\_time} = \mathrm{M} * \text{cache\_service\_time}.$$

Using Little's Law, the throughput of the disk array is given by

$$\text{array\_throughput} = \frac{\mathrm{M}}{(\text{CPU\_delay} + \text{cache\_response\_time} + \text{disk\_response\_time})}.$$

Using Little's Law again, the approximate queue length at a disk is given by

$$\text{disk\_queue} = \text{disk\_response\_time} * \text{array\_throughput}.$$

Acknowledgments

The authors thank Eitan Bachmat, Manish Madhukar, Mustafa Uysal, Sajeev Varki, John Wilkes, the associate editor, and three anonymous reviewers for their helpful comments. This work was supported in part by Hewlett Packard Labs and by the US National Science Foundation under grants ITR-0082399 and CCR-0093111.

References

[1] G.A. Alvarez, E. Borowsky, S. Go, T.H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes, "Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems," *ACM Trans. Computer Systems,* vol. 19, no. 4, pp. 483-518, Nov. 2001.

[2] E. Anderson, "Simple Table-Based Modeling of Storage Devices," technical report, HP Laboratories SSP, July 2001.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch, "Hippodrome: Running Circles around Storage Administration," *Proc. USENIX Conf. File and Storage Technologies (FAST),* pp. 175-188 Jan. 2002.

[4] M. Andrews, M. Bender, and L. Zhang, "New Algorithms for the Disk Scheduling Problem," *Proc. IEEE Symp. Foundations of Computer Science,* pp. 550-559, Oct. 1996.

[5] O.I. Aven, E.G. Coffman, and Y.A. Kogan, *Stochatic Analysis of Computer Storage.* D. Reidel, ed., May 1987.

[6] E. Bachmat, "Average Case Analysis for Batched Disk Scheduling and Increasing Subsequences," *Proc. 34th Ann. ACM Symp. Theory of Computing,* pp. 277-286, May 2002.

[7] E. Bachmat and J. Schindler, "Analysis of Methods for Scheduling Low Priority Disk Drive Tasks," *Proc. ACM SIGMETRICS,* pp. 55-65, June 2002.

[8] S. Chen and D. Towsley, "The Design and Evaluation of RAID 5 and Parity Striping Disk Array Architectures," *J. Parallel and Distributed Computing,* vol. 17, nos. 1-2, pp. 58-74, Jan. 1993.

[9] S. Chen and D. Towsley, "A Performance Evaluation of RAID Architectures," *IEEE Trans. Computers,* vol. 45, no. 10, pp. 1116-1130, Oct. 1996.

[10] R.B. Cooper, *Introduction to Queueing Theory.* Md.: Mercury Press/Fairchild Publications, 1990.

[11] W.V. Courtright II, "A Transactional Approach to Redundant Disk Array Implementation," PhD thesis, Dept. of Electrical Eng. and Computer Science, Carnegie-Mellon Univ., May 1997.

[12] R.A. Dekoning and G. J. Fredin, "Method and Apparatus for Efficient Management of Non-Aligned I/O Write Request in High Bandwidth Raid Applications," technical report, United States Patent and Trademark Office, US Patent 5860091, June 1996.

[13] G.R. Ganger, "System-Oriented Evaluation of I/O Subsystem Performance," Technical Report CSE-TR-243-95, Univ. of Michigan, June 1995.

[14] Hewlett-Packard Company, HP SureStore E Disk Array FC60 User's Guide, Pub. No. A5277-90001, Dec. 2000.

[15] S.M.R. Islam and L.A. Riedle, "Coalescing Raid Commands Accessing Contiguous Data in Write-through Mode," technical report, United States Patent and Trademark Office, US Patent 6195727, Mar. 1999.

[16] S.M.R. Islam and L.A. Riedle, "Method and System for Updating Data in a Data Storage System," technical report, United States Patent and Trademark Office, US Patent 6334168, Feb. 1999.

[17] G.J. Mcknight, L.A. Riedle, and C.T. Stephan, "Method and System for Improving Raid Controller Performance through Adaptive Write Back/Write through Caching," technical report, United States Patent and Trademark Office, US Patent 6629211, Apr. 2001.

[18] R. S. Mason Jr., Y. Ofek, N. Vishlitzky, D. Arnon, and E. Bachmat, "Dynamic Adjustment of Mirror Service Policy for Logical Volumes in a Disk Drive System Based on Collected Statistics," technical report, United States Patent and Trademark Office, US Patent 6112257, Sept. 1997.

[19] R. Karedla, J.S. Love, and B.G. Wherry, "Caching Strategies to Improve Disk Performance," *Computer,* vol. 27, no. 3, pp. 38-46, Mar. 1994.

[20] M.Y. Kim and A.N. Tantawi, "Asynchronous Disk Interleaving: Approximating Access Delays," *IEEE Trans. Computers,* vol. 40, no. 7, pp. 801-810, July 1991.

[21] A. Kuratti and W.H. Sanders, "Performance Analysis of the RAID5 Disk Array," *Proc. IEEE Int'l Computer Performance and Dependability Symp.,* pp. 236-245, 1995.

[22] E.K. Lee and R.H. Katz, "An Analytic Performance Model of Disk Arrays," *Proc. ACM SIGMETRICS,* pp. 98-109, May 1993.

[23] J. May, *Parallel I/O for High Performance Computing.* Morgan Kaufmann Publishers, 2001.

[24] J. Menon, "Performance of RAID 5 Disk Arrays with Read and Write Caching," *Distributed and Parallel Databases,* vol. 2, no. 3, pp. 261-293, July 1994.

[25] A. Merchant and P.S. Yu, "An Analytical Model of Reconstruction Time in Mirrored Disks," *Performance Evaluation,* vol. 20, nos. 1-3, pp. 115-129, May 1994.

[26] A. Merchant and P.S. Yu, "Analytic Modeling and Comparisons of Striping Strategies for Replicated Disk Arrays," *IEEE Trans. Computers,* vol. 44, no. 3, pp. 419-433, Mar. 1995.

[27] A. Merchant and P.S. Yu, "Analytic Modeling of Clustered RAID with Mapping Based on Nearly Random Permutation," *IEEE Trans. Computers,* vol. 45, no. 3, pp. 367-373, Mar. 1996.

[28] D. Muntz and P. Honeyman, "Multi-Level Caching in Distributed File Systems—or—Your Cache Ain't Nuthin but Trash," *Proc. USENIX Assoc. Winter Conf.,* pp. 305-313, Jan. 1992.

[29] E. Ofer, N. Vishlitzky, and J. Fitzgerald, "Dynamically Adaptive Data Retrieval for a Disk Drive Storage System," technical report, United States Patent and Trademark Office, US Patent 5742789, Dec. 1995.

[30] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *Computer,* vol. 27, no. 3, pp. 17-28, 1994.

[31] Seagate Corp., Cheetah 73 Family: ST173404LW/LWV/LC/LCV Product Manual, vol. 1, http://www.seagate.com/support/disc/manuals/scsi/83329478f.pdf, 2004.

[32] E. Shriver, A. Merchant, and J. Wilkes, "An Analytic Behavior Model for Disk Drives with Readahead Caches and Request Reordering," *Proc. ACM SIGMETRICS,* pp. 182-191, June 1998.

[33] A. Thomasian and J. Menon, "Performance Analysis of RAID5 Disk Arrays with a Vacationing Server Model for Rebuild Mode Operation," *Proc. 10th Int'l Conf. Data Eng.,* pp. 111-119, Feb. 1994.

[34] A. Thomasian and J. Menon, "RAID5 Performance with Distributed Sparing," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 6, pp. 640-657, June 1997.

[35] M. Uysal, G. Alvarez, and A. Merchant, "A Modular, Analytical Model for Modern Disk Arrays," *Proc. IEEE Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS),* pp. 183-193, Aug. 2001.

[36] E. Varki, "Mean Value Technique for Closed Fork-Join Networks," *Proc. ACM SIGMETRICS,* pp. 103-112, May 1999.

[37] E. Varki, A. Merchant, J. Xu, and X. Qiu, "An Analytical Model of Disk Arrays under Synchronous I/O Workloads," technical report, Univ. of New Hampshire, Jan. 2003.

[38] N. Vishlitzky, Y. Ofek, and E. Bachmat, "Redundant Storage with Mirroring by Logical Volume with Diverse Reading Process," technical report, United States Patent and Trademark Office, US Patent 5987566, Oct. 1998.

[39] N. Vishlitzky, R. Wilson, and P. Tzelnic, "Prefetching to Service Multiple Video Streams from an Integrated Cached Disk Array," technical report, United States Patent and Trademark Office, US Patent 5737747, June 1996.

[40] J. Wilkes, "The Pantheon Storage-System Simulator," Technical Report HPL-1999-127, Hewlett-Packard Laboratories, Dec. 1995.

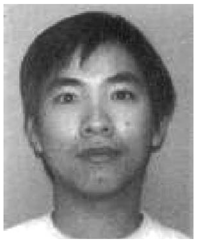
[41] T.M. Wong and J. Wilkes, "My Cache or Yours? Making Storage More Exclusive," *Proc. USENIX Ann. Technical Conf.,* pp. 161-175, June 2002.

[42] W.S. Wong and R.J.T. Morris, "Benchmark Synthesis Using the LRU Cache Hit Function," *IEEE Trans. Computers,* vol. 37, no. 6, pp. 637-645, June 1988.

[43] Y. Yochai and R.S. Mason, "Adaptive Prefetching of Data from a Disk," technical report, United States Patent and Trademark Office, US Patent 6529998, Nov. 2000.

Elizabeth Varki received the BS degree in mathematics from Delhi University, the MS degree in computer science from Villanova University, and the PhD degree in computer science from Vanderbilt University. She is an associate professor in the Department of Computer Science at the University of New Hampshire. Her research interests include performance evaluation of computer and storage systems, management of storage systems, and distributed/parallel systems. She is a member of the ACM, the IEEE, and the IEEE Computer Society.

Arif Merchant received the BTech degree in computer science from the Indian Institute of Technology at Bombay in 1984 and the PhD degree in computer science from Stanford University in 1991. He is currently a senior research scientist in the Storage Systems Department at Hewlett-Packard Laboratories. Prior to joining Hewlett-Packard, he worked at the IBM T.J. Watson Research Center and the NEC Computer and Communications Laboratories. His research interests include the modeling, design, and management of computer systems, particularly storage systems. He has served on several program committees for SIGMETRICS and performance conferences and is a cochair of the program committee for SIGMETRICS/Performance 2004.



Jianzhang Xu received the BS degree in microelectronics from South China University of Technology and the MS degree in computer science from the University of New Hampshire. He is a software engineer for Falconstor Software Inc. His research area is storage systems management.

Xiaozhou Qiu received the BS degree in physics from Peking University and the MS degree in computer science from the University of New Hampshire. He is a software engineer for Falconstor Software Inc., and he is responsible for storage management software development.