An analytic behavior model for disk drives with readahead caches and request reordering

Elizabeth Shriver[*], Arif Merchant[†] and John Wilkes[‡]

Abstract

Modern disk drives read-ahead data and reorder incoming requests in a workload-dependent fashion. This improves their performance, but makes simple analytical models of them inadequate for performance prediction, capacity planning, workload balancing, and so on. To address this problem we have developed a new analytic model for disk drives that do readahead and request reordering. We did so by developing performance models of the disk drive components (queues, caches, and the disk mechanism) and a workload transformation technique for composing them. Our model includes the effects of workload-specific parameters such as request size and spatial locality. The result is capable of predicting the behavior of a variety of real-world devices to within 17% across a variety of workloads and disk drives.

1 Motivation

There are many reasons for wanting analytical performance models of disk drives and other storage devices. In our case, we were working to develop an automatic *attribute-managed storage system* that takes in descriptions of the storage workload and automatically designs and configures a storage system that meets those needs [11]. One component of the solution was an assignment engine that explored the design space – which workload element to assign to which storage device. Each trial in this search required a performance prediction, which meant that we needed a performance model that was both fast (a few milliseconds to execute), accurate (within 30% of the real device), and capable of representing a variety of storage devices.

Although much effort was expended in producing analytical models of disk drives in the 1960s and 1970s, most recent modeling work has been concentrated on disk arrays, so advances in disk drives such as on-board controllers that cache requests and do readahead and write-behind have largely been ignored. As [25] showed, accurate modeling of these behaviors is important: ignoring caching effects can result in performance prediction errors of over 100%.

Additionally, since we needed to match workloads to devices to produce good assignments, the performance model we needed had to take account of a range of workload characteristics. And finally, our timing requirements meant that analytical models were our only choice, since the detailed simulation models that can provide high accuracy (e.g., [25, 18, 10]) run for too long to be included in the inner loop of an optimization engine.

This paper presents a significant step in a solution to this problem: it presents an analytic model for realistic modern disk drives that support request reordering and read-ahead. We believe that the model can readily be generalized to handle disks with a variety of caching and queueing policies, as well as disk arrays, as well as a wider range of workloads than it already does.

The remainder of this paper is organized as follows. The next section describes our overall approach; Section 3 lists the attributes that we use to characterize our workloads and the devices we model. Section 4 is the meat of the paper – in it, we present the models themselves in sufficient detail to allows others to reproduce them. Since a model is only as good as its predictive abilities, Section 5 presents a summary of the extensive validation work we performed. Section 6 discusses related work and Section 7 offers some conclusions and discusses potential future work.

2 Approach

We model complex storage devices by modeling the individual physical *components* of the device, such as queues, caches, and disk mechanisms, and then composing the components to give a *composite device model* for the entire storage device. For example, a queueing, caching disk drive is modeled as three components: a disk mechanism, a cache, and a request queue (Figure 1).

Our approach is based on four important ideas:

- A storage device model can be created by *composing* models for the individual components of a storage device: queues, controllers, caches, and disk mechanisms.
- The performance characteristics of a storage workload can be captured by a small set of *behaviors* that describe the workload such as the mean request size and the request arrival process.
- The service time predictions for a component can be determined from the predictions of the next lower level plus the characterization of the workload presented to the component. That is, the component models can be largely developed in isolation, although their predictions depend on each other's output. Our service time derivations are presented in Sections 4.1–4.3.
- Each component may modify the workload it presents to the next lower-level component. We formalize these as *workload transformations*, which are presented in Sections 4.4 and 4.5.

[*]Lucent/Bell Labs. Supported in part by the National Science Foundation under grant CCS-9504175. Work performed while a PhD student at NYU and a visiting scholar at Hewlett-Packard Laboratories; shriver@research.bell-labs.com; http://www.bell-labs.com/~shriver/.

[†]Hewlett-Packard Laboratories; arif@hpl.hp.com.

[‡]Hewlett-Packard Laboratories; wilkes@hpl.hp.com; http://www.hpl.hp.com/personal/John_Wilkes.

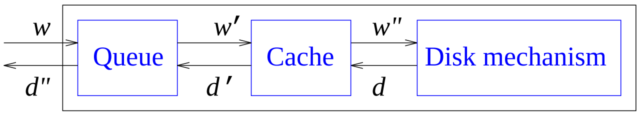

Figure 1: A composite model of a queueing, caching disk drive built from three separate component models. The input to the composite model is the workload w; this is transformed by the queue to workload w', and by the cache to workload w'', which is what the disk mechanism model sees. In turn, the disk mechanism model emits performance predictions d, which are transformed by the cache and queue component models to d'', which is the output of the entire composite device model. (Note: although we write about "higher" and "lower" level components, it is convenient to draw the models from left to right, with "higher" being leftmost, nearest to the host system.)

In summary, each component model takes as input a workload characterization, transforms this to a workload to impose on lower-level components. In turn, the performance predictions of a component typically depend on the service time predictions of the component(s) on which it relies. For example, a request queue may increase spatial locality by selecting requests in an order that minimizes physical arm movement (a workload transformation), but increase the perceived service time because of queueing delays (a behavior transformation); a cache may reduce request rates seen by a lower-level component by absorbing some of the requests on cache hits, and offer reduced service times on those same hits.

Our method is an instance of the general category of decomposition models with weakly coupled components. Although the component characteristics are often specified through mean values, the method has no connection with mean value analysis (MVA), which is an approximate method for analysis of closed queueing networks [20].

3 Device and workload specifications

A performance prediction for a storage device is a function of both the storage device itself and the workload presented to it. This section discusses how we characterize each of these. Although variances for many of the items discussed here are sometimes important in practice, we concentrate our presentation in this paper solely on mean values.

3.1 Device specification

In some ways, the characterization of the disk drive is simpler than that of the workload, since the number of different underlying processes is known in advance and bounded. ([25, 28] provide a more detailed introduction to disk drives and these processes.) Our model handles the subset shown in Figure 1 and summarized here. Table 1 lists the parameters that we use to characterize the components of our composite disk model.

- Incoming requests are placed into a *request queue*, and serviced in an order that depends on the queue's scheduling algorithm. One commonly implemented algorithm is Shortest Positioning Time First (SPTF, also known as SATF [27, 16]), which services requests in an order that minimizes the total positioning time.

- The *disk cache* acts both as a speed-matching buffer and as a means for caching data recently read from or written to the disk.

Table 1: The characteristics of the components of our composite disk model.

component	*property*	*units*
disk mechanism	seek time as a function of distance revolution time track switch time cylinder switch time number of bytes, number of cylinders, sectors per track, and tracks per cylinder	seconds seconds seconds seconds —
cache	cache segment size read-ahead policy (on, off, ...) amount to read-ahead write-to-disk policy (write-through, write-back, ...) cache-to-host transfer rate	bytes — bytes — bytes/second
queue	scheduling algorithm (FCFS, SCAN, CSCAN, ...)	—

Since the disk cache is typically small compared to the file buffer cache of the host that is using it, "random" and "reuse" cache hits are relatively rare, but requests for contiguous extensions of prior reads can be common. As a result, a useful optimization is for the disk drive to perform *readahead*: once the disk mechanism completes a read request, it may continue to read data into the cache, in anticipation of a future request for this data. Readahead is usually only done when the disk mechanism has no other useful work to perform.

Disk caches are made from volatile memory, so they are (by default) write-through to preserve data on power failure. Although this choice can usually be overridden by the disk's client, doing so brings the risk of data loss and the need for more complicated error recovery after a power failure. In some write-intensive environments these write-behind algorithms can provide similar benefits to readahead, and are considered worth the risks. Our model doesn't yet support them, although we believe that an approach similar to the one we use to model readahead would be quite successful.

- The *disk mechanism* comprises the rotating platters and the arms that move the disk heads to the correct tracks. The time to position the disk head has two parts: a *seek time* while the heads are moved to the correct track; and a *rotational latency* while the platter carrying the data rotates into position under the heads. Once in place, data is transferred to or from the platter at a rate determined by the rotation speed of the platter and its recording density, with occasional interruptions to switch to a new platter or a new cylinder if the transfer crosses a track or cylinder boundary.

3.2 Workload specification

Workload characterization is a hard research problem: arbitrarily complex patterns can occur in workloads, and capturing the important aspects of these with only a few parameters is difficult. We started with a small subset that allowed us to represent several of the behaviors that we have observed in real-life I/O traces. In particular, we chose to emphasize support for spatial locality in the form of runs of

Figure 2: A workload with a bursty arrival process. Each vertical bar represents a request arrival. Some requests arrive while their predecessors are being serviced, and so queues build up. This can happen even when the device is able to service the long term mean arrival rate.

requests to contiguous data, and temporal locality on the form of bursty arrival patterns.

Table 2 lists the attributes we used to characterize workloads for the performance models described here. The following paragraphs describe these in greater detail.

Arrival-process attributes The first four workload attributes capture information about the temporal access pattern of requests as they arrive at the storage device in addition to the long-term average rate. We model three different arrival processes:

- constant: the interarrival time between requests is fixed. The workload request process can be closed (blocking), which means that only one request can be in the device at a time, or open (non-blocking), which allows multiple simultaneously outstanding requests – most of which will probably be blocked internally at a queue inside the storage device drive.
- Poisson: the interarrival times are independent and exponentially distributed; Poisson arrival processes are always open.
- bursty: some of the requests arrive sufficiently close together that their interarrival time is less than the service time, so a backlog of outstanding requests builds up. Bursty arrival processes are always open.

Burst attributes Prior work [24] has shown that bursts are important to model in I/O systems. We define a *burst* as a group of consecutive requests in the request stream whose interarrival times are less than the *burst request interarrival time*. For a particular device, the burst request interarrival time is the mean device service time. Figure 2 shows an example of a bursty arrival process.

The *burst fraction* indicates what portion of all the requests occur in such bursts; the *requests per burst* identifies the mean number of requests in a burst. The *request rate* is the arrival rate during a burst, averaged over the burst.

Spatial locality attributes The next set of attributes from Table 2 allow us to capture different kinds of spatial locality on the workload. Although many such characterizations are possible, we concentrated on the notion of sequentiality, because of the very important part this plays in predicting the performance of a disk drive: by eliminating positioning time, sequential accesses can be very much faster than random ones. In addition, readahead can make sequential runs of read requests particularly fast.

A *run* is a sequence of consecutive requests that access sequentially contiguous locations (see Figure 3). Fully sequential workloads consist of a single run and are rare in practice; more common are partially sequential workloads which have multiple runs, perhaps intermixed with other requests. The fraction of requests that are part of a run is the locality_fraction. The *run stride* is the mean distance between the beginnings of two consecutive runs.

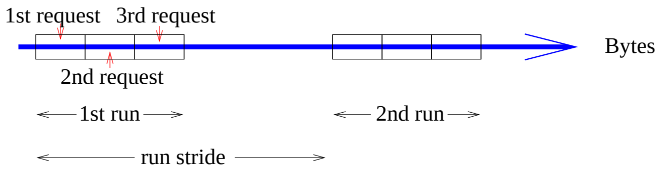

Figure 3: Two sequential strided runs, each consisting of three requests.

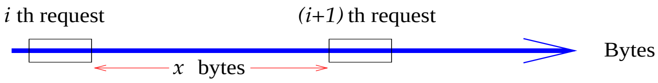

Figure 4: A sparse, non-contiguous run of requests, with inter-request distance x.

A special form of sequential spatial locality occurs in some database workloads, when the consecutive requests are not quite contiguous – that is, the runs have "holes" in them. Since these can also benefit from readahead, we represent these as *sparse runs*. A sparse run is a group of consecutive requests that each access data within x bytes of their predecessor (see Figure 4). The largest interesting value of x is a function of the disk drive's cache segment size and readahead policy, and is discussed further in Section 4.2.

Now that the device and workload parameters have been described, we can proceed to see how they are used in the model proper.

4 The models

This section presents our models. We begin with a discussion of the service time prediction (or behavioral) component models. These provide the output from the model in the form of predictions of the performance seen by a workload arriving at the device or one of its components. The disk mechanism is described in Section 4.1, the cache in 4.2, and the queue in 4.3. We then present the workload transformations performed by the component models for caches (Section 4.4) and queues (4.5).

4.1 Disk mechanism service time model

The service time of a request in the disk mechanism (Mech_Service_Time) is a function of: (1) the seek time (ST), which is the amount of time that it takes to move the disk heads to the desired cylinder; (2) the rotational latency (RL), which is the time for the platter to rotate to the desired sector; and (3) the transfer time (TT), which is the time to transfer the data from the disk mechanism to the next higher level. Since these are typically independent variables, we can approximate the expected value of the disk mechanism service time as

$$\begin{aligned}&\mathbf{E}[\mathsf{Mech\_Service\_Time}]\\&\quad\approx\ \mathbf{E}[\mathsf{ST}]+\mathbf{E}[\mathsf{RL}]+\mathbf{E}[\mathsf{TT}[\mathsf{request\_size}]]. \qquad (1)\end{aligned}$$

The transfer time is the easiest of these: let Transfer_Rate be the transfer rate of data off/onto the disk; TT[request_size] can then be approximated as $\mathbf{E}[\mathsf{request\_size}]/\mathbf{E}[\mathsf{Transfer\_Rate}]$. (Variations will occur as a result of track and cylinder switches and different track sizes in different zones on the disk.) The rest of this section describes our technique for approximating seek time and rotational latency.

Table 2: The attributes used to characterize workloads. Unless otherwise noted, all these are mean values averaged across the lifetime of the workload.

attributes	*description*	*units*
temporal locality measures		
request_rate	rate at which requests arrive at the storage device	requests/second
arrival_process	inter-request timing (constant [open, closed], Poisson, or bursty)	—
request_per_burst	size of a burst	requests
bursty_fraction	fraction of requests that occur in a burst	0–1
spatial locality measures		
data_span	the span (range) of data accessed	bytes
request_size	length of a host read or write request	bytes
run_length	length of a *run*, a contiguous set of requests	bytes
run_stride	distance between the start points of two consecutive runs	bytes
locality_fraction	fraction of requests that occur in a run	0–1
requests_per_sparse_run	the size of a *sparse run* (a run with internal holes)	requests
sparse_run_length	length of a sparse run	bytes
sparse_run_fraction	the fraction of requests that are in sparse runs	0–1
other measures		
read_fraction (write_fraction)	fraction of the requests that are reads (writes)	0–1

Seek time The seek time can be approximated as the following function of dis, the number of cylinders to be traveled:

$$\mathsf{SeekTime[dis]} = \begin{cases} 0 & \mathsf{dis} = 0 \\ a + b\sqrt{\mathsf{dis}} & 0 < \mathsf{dis} \leq e \\ c + d \cdot \mathsf{dis} & \mathsf{dis} > e \end{cases}$$

where a, b, c, d, and e are device-specific parameters.

To determine the mean seek time we use the fact that the expected value of a continuous real random variable is the integral of 1 minus its cumulative distribution function [13]:

$$\mathbf{E}[\mathsf{ST}_{\mathrm{ran}}] = \int_0^\infty t \cdot d\mathbf{Pr}[\mathsf{ST} < t] = \int_0^\infty \mathbf{Pr}[\mathsf{ST} \geq t]\, dt$$

where $\mathbf{Pr}[\mathsf{RV} < v]$ is the probability that RV is less than v, and $\mathbf{E}[\mathsf{ST}_{\mathrm{ran}}]$ represents the mean seek time for a workload whose requests are randomly distributed across the disk.

Let SD be a continuous random variable representing the integer-valued seek distance between consecutive requests, measured in cylinders. Assume that the requests are uniformly distributed across a range of C cylinders; $C = \mathsf{data\_span/Ave\_Bytes\_per\_Cylinder}$. Let $\mathsf{Cyl}[t]$ be the inverse function of the seek-curve function: $\mathsf{SeekTime}[\mathsf{Cyl}[t]] = t$. Then,

$$\mathbf{Pr}[\mathsf{ST} \geq t] = \begin{cases} 1 & \text{if } 0 < t \leq a \\ \mathbf{Pr}[\mathsf{SD} > \mathsf{Cyl}[t]] & \text{if } a < t \leq \mathsf{SeekTime}[C] \\ 0 & \text{if } t > \mathsf{SeekTime}[C] \end{cases}$$

$$\mathbf{E}[\mathsf{ST}_{\mathrm{ran}}] = a + \int_a^{\mathsf{SeekTime}[C]} \mathbf{Pr}[\mathsf{SD} \geq \mathsf{Cyl}[t]]\, dt. \quad (2)$$

A closed form can be determined at this point (and can be found in [28]). In addition, this closed form can be used to generate a closed form that takes into account spatial locality (as specified in Section 3.2). For example, if the workload has spatial locality (defined here in terms of runs) and the first request of each run is uniformly spatially distributed across the disk, the mean seek time can be approximated as

$$\mathbf{E}[\mathsf{ST}_{\mathrm{seq}}] \approx \frac{\mathsf{request\_size}}{\mathsf{run\_length}} \cdot \mathbf{E}[\mathsf{ST}_{\mathrm{ran}}]. \quad (3)$$

If only a fraction of the requests are part of runs, (3) can be improved using the locality fraction attribute.

A particular case of interest is that with n requests uniformly distributed across a range of cylinders C. This captures the behavior of some of the flavors of request reordering that queues can perform. Since $n+1$ random points on the interval $(0, C)$ create $n+2$ intervals whose lengths have the same distribution [12, 28],

$$\mathbf{Pr}[\mathsf{SD} \geq y] = \left(1 - \frac{y}{C}\right)^{n+1}, \quad (4)$$

which can be combined with (2) and (3) to provide a closed-form solution for the expected seek time in a workload-specific manner.

Rotational latency If we assume that the requests are randomly distributed on the sectors of the given cylinder using a uniform distribution, we have

$$\mathbf{E}[\mathsf{RL}] \approx \mathsf{Revolution\_Time}/2. \quad (5)$$

Combining (2) through (5) as specified in (1) gives us a way to approximate the disk mechanism service time in a workload- and device-dependent manner.

4.2 Cache service time model

The service time of a caching device depends both on the mechanism service time and the cache miss and hit probabilities. The hit rate for simple LRU access processes is typically zero for disk drives, because hosts caches are much bigger than disk caches, and the disk cache covers a tiny fraction of the underlying storage. However, the hit rate for readahead requests is often quite high, because readahead often successfully predicts that data immediately following requested data will also be requested. If the disk drive would otherwise be idle, it does so at near-zero cost. Modeling the cache effects of readahead is the main content and contribution of this section.

Let Cache_Service_Time be a continuous random variable representing the service time for a caching device. The general form is

$$\mathbf{E}[\mathsf{Cache\_Service\_Time}] \approx \frac{\mathbf{E}[\mathsf{request\_size}]}{\mathsf{Cache\_Transfer\_Rate}} + \mathsf{Miss\_Prob} \cdot \mathbf{E}[\mathsf{Mech\_Service\_Time}] \quad (6)$$

where $\mathsf{Cache\_Transfer\_Rate}$ is the data transfer rate between the cache and the host, and $\mathsf{Miss\_Prob}$ is the probability that a request will be a cache miss. (If readahead is not enabled, or if request is a write, the miss probability will be 1.) The rest of this section presents our approximation of the cache miss probability—the only unknown in (6). We also present a slightly more refined version of (6).

The cache miss probability For most workloads, the first request of a run will be a cache miss. We assume here that a read-ahead is performed only after a cache miss and (for simplicity of exposition) that all the requests are reads.

The amount of data read-ahead ($\mathsf{Read\_Ahead\_Length}$) is bounded above by the cache segment size; some disk drives also stop read-ahead once a track or cylinder boundary is reached. We define the amount of data read into cache by a cache miss to be:

$$\mathsf{data\_read} \quad = \quad \mathsf{request\_size} + \mathsf{Read\_Ahead\_Length}. \quad (7)$$

For a small sequential run ($\mathsf{run\_length} \leq \mathsf{data\_read}$), the number of requests that can be serviced by the data read in by each cache miss is the same as the number of requests in the run: $\lfloor \mathsf{run\_length}/\mathsf{request\_size} \rfloor$. If the run is large (i.e., $\mathsf{run\_length} > \mathsf{data\_read}$), the number of requests that can be serviced by the data read in by one cache miss is determined by the amount of data read into the cache: $\lfloor \mathsf{data\_read}/\mathsf{request\_size} \rfloor$. This can be written as

$$\begin{aligned} &\mathsf{requests\_serviced\_by\_one\_disk\_access} \\ &= \left\lfloor \frac{\min\{\mathsf{run\_length}, \mathsf{data\_read}\}}{\mathsf{request\_size}} \right\rfloor. \end{aligned}$$

If the workload has sparse-run spatial locality, and the amount of data read-ahead is bounded only by the cache segment size, then we can approximate the length of the sparse run by the cache segment size. The number of requests serviced by one disk access is equal to the number of requests in one sparse run:

$$\begin{aligned} &\mathsf{requests\_serviced\_by\_one\_disk\_access} \\ &\approx \quad \mathsf{requests\_per\_sparse\_run}. \end{aligned}$$

For both regular sequential runs and sparse runs the miss probability can be approximated as

$$\mathsf{Miss\_Prob} \approx 1/\mathsf{requests\_serviced\_by\_one\_disk\_access}. \quad (8)$$

However, the cache miss probability also depends on the request rate. If the disk wins the race to read-ahead the next request's data before the request arrives, then the request will hit in the cache. If it loses, a cache miss will occur, and the request will be stalled until the data it needs is transferred off the platter. Since disk caches tend to purge data from their cache that has already been sent to the requesting host, this process can continue indefinitely, with the cache acting as a speed-matching circular FIFO.

To determine the likelihood of the new request coming in before the FIFO is filled and thereby causing a cache miss, let $\mathsf{Mech\_Service\_Time}$ be the mechanism service time for a cache miss. If $\mathsf{request\_rate} < \mathsf{Mech\_Service\_Time}$, the following equation replaces (8):

$$\begin{aligned} &\mathsf{Miss\_Prob} \\ &\approx \frac{\mathsf{Mech\_Service\_Time}}{\mathsf{interarrival\_time} \cdot \mathsf{requests\_serviced\_by\_one\_disk\_access}}. \end{aligned}$$

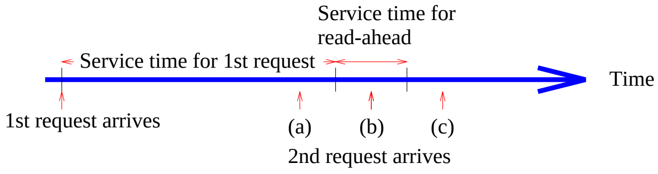

Figure 5: Request arrival time line for the first two requests of a sequential run where the cache performs read-ahead.

Partial cache hits We say that the ith request of a run is a *partial hit* if the readahead for its data has begun but not finished when the request arrives (case (b) in Figure 5). For example, a partial hit will occur for the second request in a run if the request interarrival time is larger than the lower level service time for a singleton request but less than the time to do this singleton plus a readahead, which is typically much less than twice the singleton case. Such partial hits affect the cache service time given in (6) by adding the product $\mathsf{Partial\_Hit\_Prob} \cdot \mathbf{E}[\mathsf{Mech\_Service\_Time}']$ where $\mathsf{Mech\_Service\_Time}'$ is the time for the mechanism to service the partial hit. If we assume that the arrival times for the requests that are partial hits are evenly distributed across the interval (b) in Figure 5, then this is just the time to transfer $\mathsf{request\_size}/2$ bytes off the platter since there will be no positioning time involved: $\mathsf{Mech\_Service\_Time}' \approx \mathsf{TT}[\mathsf{request\_size}/2]$. The rest of this section discusses our approximation for the $\mathsf{Partial\_Hit\_Prob}$.

If the mean request interarrival time is shorter than the mean service time of a readahead request (and thus much shorter than the service time of a random request), the second and subsequent requests will all be partial hits, and we can approximate $\mathsf{Partial\_Hit\_Prob} \approx 1$. Conversely, if the mean request interarrival time is much longer than the mean service time of a random request, the second and subsequent requests will usually be hits, and we can approximate $\mathsf{Partial\_Hit\_Prob} \approx 0$. The tricky case to handle is when the request interarrival time is similar to the service time of a random request. If we assume that the time between requests and the transfer rate are both constant, then the probability that the ith request in a run is a partial hit can be formalized as

$$\mathbf{Pr}[(i-1) \cdot \mathsf{interarrival\_time} \leq \mathsf{ST} + \mathsf{RL} + \mathsf{TT}[i \cdot \mathsf{request\_size}]].$$

which can be written as $\mathbf{Pr}[z \leq \mathsf{ST} + \mathsf{RL}]$ where $z = (i-1) \cdot \mathsf{interarrival\_time} - \mathsf{TT}[i \cdot \mathsf{request\_size}]$. Using the definition of probability and the fact that the seek time and rotational latency are independent, we have

$$\begin{aligned} &\mathbf{Pr}[z \leq \mathsf{SeekTime} + \mathsf{Rot\_Lat}] \\ &= \int_{x=0}^{z} \mathbf{Pr}[\mathsf{SeekTime} \leq x] \, d\mathbf{Pr}[\mathsf{Rot\_Lat} < z - x]. \quad (9) \end{aligned}$$

A closed form can be determined at this point. Graphing the above probability as a function of z shows that as the request interarrival time increases, the probability of a partial hit decreases, as expected.

A similar analysis can be performed for the sparse run case. It is also possible to relax the assumption of constant interarrival time, and a similar analysis can also be performed if the arrival process is Poisson. This technique allows us to approximate the workload-dependent service time of a disk with a cache that performs readahead.

4.3 Queue service time model

Let $\mathsf{Queue\_Delay}$ be a continuous random variable for the time that a request is delayed in the queue. The mean service time of a request in a queueing, caching device is a function of the queue delay:

$$\begin{aligned}&\mathbf{E}[\mathsf{Service\_Time}]\\&\quad= \mathbf{E}[\mathsf{Queue\_Delay}] + \mathbf{E}[\mathsf{Cache\_Service\_Time}]. \quad (10)\end{aligned}$$

The rest of this section discusses how to approximate queue delay for workloads with an open queueing model.

Queue delay—Poisson arrivals The standard M/G/1 queueing model allows us to approximate the mean queue delay from the request rate of the workload and the mean and variance of the cache service time [17]:

$$\mathbf{E}[\mathsf{Queue\_Delay}] = \frac{\rho^2(1+C_s^2)}{2(1-\rho)\mathsf{request\_rate}} \quad (11)$$

where

$$\begin{aligned}\rho &= \mathsf{request\_rate}\cdot\mathbf{E}[\mathsf{Cache\_Service\_Time}]\\ C_s &= \frac{\mathbf{StDev}[\mathsf{Cache\_Service\_Time}]}{\mathbf{E}[\mathsf{Cache\_Service\_Time}]}.\end{aligned}$$

The variance of $\mathsf{Cache\_Service\_Time}$, and hence $\mathbf{StDev}[\mathsf{Cache\_Service\_Time}]$, can be computed by methods similar to those used for $\mathbf{E}[\mathsf{Cache\_Service\_Time}]$; see [28] for details. In the absence of other information, we may approximate $\mathbf{StDev}[\mathsf{Cache\_Service\_Time}] = \mathbf{E}[\mathsf{Cache\_Service\_Time}]$, which corresponds to an M/M/1 queue model. The storage system community has generally used the above method of determining the mean queue delay to analyze the mean queue delay of a FCFS queue (e.g., [32, 21]). We have extended this approach to analyze other scheduling algorithms that are frequently implemented in queues in the I/O path such as LOOK and Shortest Seek Time First. We continue to use an M/G/1 queue model approximation for these cases, but substitute in a cache service time that is specific to the actual scheduling algorithm used by modifying the run stride spatial locality measure in a scheduling-algorithm-specific way (this is discussed further in Section 4.5).

Queue delay—bursty arrivals Although there are many ways to characterize a burst, we use a simple definition that nonetheless captures the essence of the issue: a *burst* is the group of consecutive requests that arrive while the storage device is still servicing a prior member of the burst (i.e., the queue length is non-zero, or the device is active). The analysis in [24] of real workloads showed that significant fractions of I/O requests occur in such bursts. This is not a Poisson arrival process, so additional work is needed to model it.

Suppose the first request in the burst arrives at time t_0, the ith interarrival time in the burst is A_i, and the lower-level service time of the ith request in the burst is B_i. Then the ith request in the burst arrives at time $t_0 + A_1 + \cdots + A_{i-1}$, and it enters service at time $t_0 + B_1 + \cdots + B_{i-1}$. Thus, the queueing delay is $\sum_{j=1}^{i-1}(B_j - A_j)$. The mean value of the queue delay of the ith request in the burst is $(i-1)(\mathbf{E}[\mathsf{LL\_Service\_Time}] - \mathbf{E}[\mathsf{interarrival\_time}])$ where $\mathsf{LL\_Service\_Time}$ represents the lower level service time. If there are n requests in the burst, the mean queue delay over all the requests is

$$\begin{aligned}&\mathbf{E}[\mathsf{Burst\_Queue\_Delay}]\\&= \frac{\sum_{i=1}^{n}(i-1)\,(\mathbf{E}[\mathsf{LL\_Service\_Time}] - \mathbf{E}[\mathsf{interarrival\_time}])}{n}\\&= \frac{n-1}{2}(\mathbf{E}[\mathsf{LL\_Service\_Time}] - \mathbf{E}[\mathsf{interarrival\_time}]). \quad (12)\end{aligned}$$

We call a request *stray* if it arrives outside of a burst (i.e., the queue length is zero and the device is idle). If there are a significant number of strays, the mean queue delay is reduced since each stray request will experience a zero queueing delay. Thus, we compute the queue delay as

$$\mathbf{E}[\mathsf{Queue\_Delay}] \approx \mathsf{bursty\_fraction}\cdot\mathbf{E}[\mathsf{Burst\_Queue\_Delay}].$$

4.4 Cache workload transformations

To determine cache service time, it is necessary to know the service times at the lower level of regular misses ($\mathsf{Mech\_Service\_Time}$) and of readahead misses ($\mathsf{Mech\_Service\_Time}'$). Thus, the component immediately below a cache sees two types of requests, each with its own characteristics. The following paragraphs describe how these characteristics are determined.

Request size The cache passes through read requests that miss in the cache to the lower level: $\mathsf{request\_size}_{r,D} = \mathsf{request\_size}_C$. (The subscript C denotes a characteristic of requests entering the cache, and D denotes requests entering the lower-level disk mechanism component.) Suppose the cache performs read-aheads by reading chunks of $\mathsf{Read\_Ahead\_Granule}$ bytes at a time. On a cache miss that provokes readahead, the cache will first pass through the request ($\mathsf{request\_size}_C$ bytes), and follow this with readahead requests ($\mathsf{request\_size}_{r,D} = \mathsf{Read\_Ahead\_Granule}$) until either the cache segment is full, another request comes in, or (in some disks) a track or cylinder boundary is reached.

With a write-through cache, writes are passed through unchanged: $\mathsf{request\_size}_{w,D} = \mathsf{request\_size}_C$.

Request rate and arrival process In the general case, both the read arrival rate and the arrival process will be transformed. However, if the arrival process starts out as Poisson or constant, it will remain so. We thus restrict our discussion to changes in the arrival rate.

Assume the original request rate is small enough so that the readahead can complete before the next request is issued (case (c) in Figure 5). The lower-level device sees the following behavior: idle, request, read-ahead request, ..., read-ahead request, idle, ... etc. That is, the transformed arrival process becomes bi-modal: a subset of the original requests arrive at the disk with something like the original arrival process, and the readahead requests arrive with a constant process with mean request rate $1/\mathsf{TT}[\mathsf{Read\_Ahead\_Granule}]$ where $\mathsf{TT}[y]$ is the amount of time needed to transfer y bytes from the lower level into the cache. The rate of original requests (of size $\mathsf{request\_size}_C$) that miss in the cache is $\mathsf{request\_rate}_{r,D} = \mathsf{Miss\_Prob}\cdot\mathsf{request\_rate}_C$. The rate of read-ahead requests (of size $\mathsf{Read\_Ahead\_Granule}$) is $\mathsf{request\_rate}_{r,D} = 1/\mathsf{TT}[\mathsf{Read\_Ahead\_Granule}]$. Thus, a readahead sequence lasts for time

$\mathsf{TT}[\mathsf{Read\_Ahead\_Length} - \mathsf{request\_size}_C]$ and contains $(\mathsf{Read\_Ahead\_Length} - \mathsf{request\_size}_C)/\mathsf{Read\_Ahead\_Granule}$ such requests.

If the cache is a write-through cache, neither the write request rate nor its arrival process will be transformed. Otherwise, the transformed arrival process will be a function of the kind of delayed-write policy used: it can be more bursty if the policy delays until it needs space or meets a threshold [24], or less so if it uses a rate-controlled trickle-out of dirty blocks. The rate can effectively be lower if some of the blocks are written out when the disk would otherwise have been idle because these do not affect the perceived drive performance seen by foreground activity.

A delayed write-back policy can also reduce the transformed write rate by absorbing overwrites, which are quite prevalent in some workloads; we currently do not have a workload behavior attribute that captures this.

Spatial locality Readahead alters the spatial locality of the workload seen by the lower-level component: it makes workloads more sequential because the readahead looks like a run of requests of length Read_Ahead_Granule. As a result, the transformed run length of a random workload that uniformly provokes readahead is $\mathsf{run\_length}_D = \mathsf{data\_read} = \mathsf{request\_size} + \mathsf{Read\_Ahead\_Length}$.

A cache performing read-ahead also makes a sparse run seem like a standard run to the lower-level device, so if the spatial locality is specified as a sparse run, the transformed workload will have $\mathsf{run\_length}_{r,D} = \mathsf{sparse\_run\_length}$.

The modifications made by the cache to the run stride attribute are typically small, so $\mathsf{run\_stride}_D \approx \mathsf{run\_stride}$.

4.5 Queue workload transformations

Queues can significantly modify the arrival process. Since there will typically be only one request active in the lower level components of the device, the transformed workload is a closed one. That is, the mean interdeparture time for the queue is the same as the mean interarrival time for the lower-level device.

Non-FCFS scheduling algorithms can also modify the run stride spatial locality properties of the workload if their queue lengths become greater than one. Typically, they do so by reducing the mean seek (and possibly rotation) distance traversed by the disk mechanism in order to reduce the physical positioning times experienced in the lower level component. For a workload that is randomly distributed across a range of cylinders C, the transformed run stride is

$$\mathsf{run\_stride}_D = \frac{C \cdot \mathsf{Bytes\_per\_Cylinder}}{\mathbf{E}[\mathsf{Queue\_Size}] + 2}.$$

Since this depends on the length of the queue, we have a feedback loop. This means the implementation of the model can require more than one iteration through the model. In practice we find that only one or two iterations are necessary to reach convergence.

We assume that the queue does not reorder sequential requests, so queues do not transform run lengths.

5 Validation of the model

One way to evaluate our model would have been to compare it with other models, but, as we discuss in Section 6, other models do not support non-random/non-Poisson workloads. In addition, this comparison would not tell us how well we were modeling the disks. Ideally, we would have liked to compare our model's predictions to measured data from a range of real disk drives under a number of real-life workloads. In practice, this is difficult to do, because reproducing interesting workloads reliably is hard, and most disk drives do not contain the internal measurement points that we needed in order to perform our component-model validations. So instead we turned to the Pantheon disk simulator [34], which has disk models that have been calibrated against real HP97560 disk drives to a demerit figure of 5.7% [24].

Using Pantheon allowed us to replay workload traces captured from real-life systems, and to generate synthetic workloads to test specific model behaviors. It also allowed us to use the measurement points inside the simulator to determine what was happening between the different components of the disk model. Since Pantheon reproduced the architectural components of real disk drives fairly faithfully, the result is that we had an independent, convenient testbed against which to compare our analytic models. This section reports on the results of that comparison.

We validated our model by comparing its performance predictions on test workloads against the Pantheon measurements when fed the same workloads: first component-by-component, and then as a whole. All of the equations presented in this paper (and many more found in [28]) were validated. In particular, (10) was used to determine the mean device service time, $\mathbf{E}[\mathsf{Device\_Service\_Time}]$.

We used many workloads in our experiments, beginning with simple, predictable synthetic workloads with Poisson arrivals and uniform random access across the disk cylinders, and progressing up through synthetic models of video-on-demand applications and the data staging and checkpointing phases of a parallel scientific file system, up to disk I/O-level traces of a number of real-life workloads including an unaudited run of the TPC-C database benchmark [31], an HP-UX timesharing system, and a number of IBM AS-400 production systems. This set included workloads with constant, Poisson, and bursty arrivals, open and closed queueing models, and a large range of spatial locality measures and request sizes. More details of the workloads and how we determined the values of the workload attributes can be found in [28]. For this paper, we have chosen representative samples to indicate the range of results that we obtained.

We made one change in our configuration from that of Figure 1: instead of modeling the request queue in the drive, we used the queue in the disk driver component. This allowed us to take advantage of Pantheon's previously-calibrated disk models without change, and meant that we could avoid the issues that arise in practice when the queue length in the disk drive is artificially limited by the host. ([35] discusses many of the variations possible in this design space.) The main performance effects of the different queue placement have to do with the timing of interconnect transfers and controller overheads; we believe that it would be straightforward to alter the models to accommodate either position.

Unless otherwise stated, the models are for an HP97560 disk drive configured with a single-segment write-through cache, and readahead that stopped when the cache segment became full.

Poisson workload Even though workloads with Poisson arrivals and random uniform access are almost unheard of in practice, we first present our model using such workloads to allow it to be compared with others. Figure 6 displays the relative error in the mean device service time as the arrival

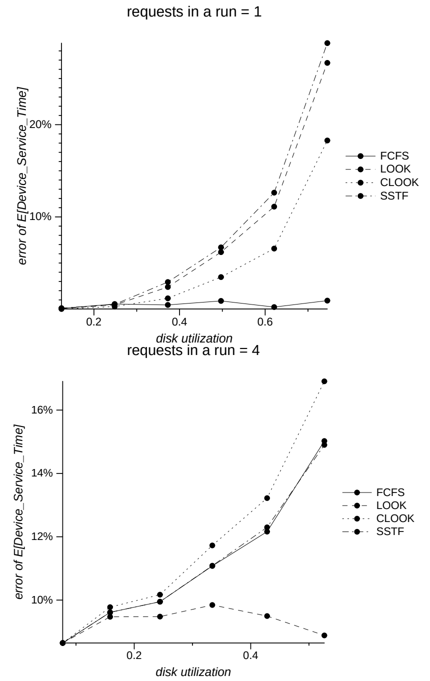

Figure 6: Validation results for a Poisson arrival process. Each graph shows the relative error of the mean device service time as a function of the disk utilization across the scheduling algorithm (i.e., the absolute value of the performance prediction error divided by the measured Pantheon value). FCFS is first-come, first-served (i.e., no reordering); LOOK and CLOOK are variants of a sort-by-cylinder policy; SSTF is shortest-seek time first. The number of requests in a run was varied from 1 to 8; the figures here are for values 1 and 4. In all cases the request size was 8KB; the arrival process was Poisson; the queueing model was open; the requests were 100% reads. The disk utilization was varied by altering the mean request rate.

rate and number of requests in a run is varied, for four different request scheduling algorithms. For these workloads, our largest relative error is 29% (for SSTF queueing at a disk utilization of 80% and no spatial locality), but the majority of errors are less than 15%.

Synthetic parallel scientific file system workload We performed experiments with synthetic workloads to validate that the mean device service time could be determined for a parallel scientific file system workload. We used synthetic workloads since traces of current scientific applications would not represent the current trend in parallel file system research. Our synthetic workloads captured both data-staging (i.e., moving the entire file from disk to main memory in stages) and checkpointing, which occurs when the application writes a subset of the application data structures to disk. Data-staging accesses the file in sequential runs of length 1 or 2, while checkpointing accesses can be captured with sequential runs of length 64. Accesses tend to be blocks that correspond to the file system block size; we used 16KB. The arrival process is constant, and the request rate should keep the disk streaming. Since the workloads have a closed arrival process, the choice of scheduling algorithm is irrelevant, so we used FCFS. Table 3 shows the results. Similar errors were obtained for other device and workload configurations. For example, we performed

Table 3: Parallel scientific file system workload results. The number of requests in a run varied from 2 to 64. The request size was 16KB; the request rate was 120 requests/second; the arrival process was constant; the queueing model was closed. The queue scheduling algorithm was FCFS.

Workload		**E**[Device_Service_Time]		
run count	*read fraction*	*measured (ms)*	*predicted (ms)*	*error (%)*
1	1	31.6	31.7	0.3%
2	1	18.8	19.7	5.1%
64	1	4.5	4.5	0.3%
1	0	28.2	28.3	0.2%
2	0	25.4	25.6	1.0%
64	0	22.5	23.0	2.3%

Table 4: The workload attribute values for an unaudited TPC-C benchmark. Cst, log, and stk correspond to three different tables in the TPC-C benchmark data; the trailing number is the disk number; the traces were taken on a 38-disk configuration.

attribute	stk.11	log.14	stk.17	stk.20	cst.24
request_rate	13.32	23.23	13.90	13.80	22.53
effective request_rate	18.82	31.81	17.28	16.39	29.62
request_size	2048	5126	2052	2049	2048
read_fraction	0.79	0.02	0.57	0.76	0.83
sparse_run_length	2.0	30.2	2.1	2.0	2.0
sparse_run_fraction	0.17	0.94	0.04	0.01	0.01
frac. of cylinders	0.05	0.22	0.02	0.05	0.05

additional experiments with request rates of 100 and 160 requests/second and obtained similar degrees of accuracy.

Transaction processing workload We validated our model using traces of a TPC-C workload with a FCFS queue scheduling algorithm. The disk modeled was an HP C2490A, since this was the kind used in the system on which the traces were obtained; its cache was configured in the same way as the HP97560 disks used in the other tests. Table 4 contains the workload attribute values that we used as input for the model. Some of the traces had significant periods of idleness; we compute the effective request rate for the model by discarding segments of the traces where there was no activity for greater than 1 second. Stk.11 has moving "hot spots" (i.e., cylinders that are heavily accessed); we capture these with the sparse run attributes. Note that most of the traces access a small percentage of the disk cylinders. The validation results are shown in Table 5. Our largest relative error is 15.7%, with most below 7.5%. This is an encouraging result for such real-world data.

AS400 database-like workload We validated our model using traces of a 2-disk IBM AS400 system running a production database-like workload supplied to us by Bruce McNutt of IBM. The workload had very little spatial locality and the arrival process was bursty.

Table 5: Transaction-processing database workload results.

Workload		**E**[Device_Service_Time]		
trace	*effective disk utilization*	*measured (ms)*	*predicted (ms)*	*error (%)*
stk.11	14.7%	9.1	9.8	7.5%
log.14	22.5%	7.7	8.9	15.7%
stk.17	13.5%	8.8	8.8	0.1%
stk.20	14.1%	9.6	10.0	4.4%
cst.24	25.7%	10.7	11.0	2.5%

Table 6: Summary of error statistics for the mean device service time of subtraces of IBM-AS-400-1 trace.

Workload			**E**[Device_Service_Time]		
subtrace	*burst size*	*read fraction*	*measured (ms)*	*predicted (ms)*	*error (%)*
0	48	0.67	388.2	397.2	2.3%
1	53	0.53	403.5	423.2	4.9%
2	71	0.51	542.8	566.3	4.3%
3	91	0.52	712.8	726.5	1.9%
4	46	0.59	367.6	374.4	1.9%
5	42	0.52	332.7	336.4	1.1%
6	37	0.57	281.2	299.4	6.5%
7	50	0.47	398.2	403.7	1.4%
8	77	0.42	573.6	608.9	6.1%
9	77	0.38	584.3	608.4	4.1%
all	59	0.52	394.0	394.2	0.7%

We split the first 10000 requests of the trace into 10 subtraces of 1000 requests each and determined the workload behavior for each subtrace; the only behavior attributes that varied were the number of requests in a burst (which varied from 37 to 91 requests), the read fraction (which varied from 0.38 to 0.67), and the lengths of the subtraces (which varied from 53 seconds to 444 seconds), long enough for steady state behavior to be achieved. As seen in Table 6, the relative errors are small (with a maximum error of 6.5%) and of the same magnitude, even though the device service time varies significantly (from 281.2 to 712.8 milliseconds). These experiments show not only that we can determine device service time when the arrival process is bursty, but also that our model is robust in light of changing read fractions, request rates, and burst sizes for this workload.

6 Related work

This paper has discussed our modeling methodology in terms of a queueing, caching disk. The usual approach to analyzing detailed disk drive performance is to use simulation (e.g., [14, 27, 36]). Although simulations can provide detailed, accurate models that accommodate arbitrary real-life workloads (e.g., measured I/O traces), it is unsuitable for the many applications that need a performance estimate quickly.

Most early modeling studies (e.g., [3, 33]) concentrated on rotational position sensing for mainframe disk drives, which had no cache at the disk and did no readahead. [5] introduced a workload-specific mechanism service time model, but the mean seek time had to be provided as part of the workload specification. Most prior work (e.g., [26, 22, 21]) has used uniform random spatial distributions. [8, 19] modeled the probability that no seek was needed; [15] reported that an exponential distribution of seek times matched measurements well for three test workloads. No general method of computing the seek time distribution given a workload was presented in any of these papers.

Trace driven analysis has been used extensively in I/O cache research, but almost all of the work on analytic cache modeling has been for processor caches. These have rather different characteristics than disk caches: the replacement costs are uniform, operations are on whole cache lines, there are no inter-line spatial locality effects, and they typically have many more cache lines than a disk does cache segments. As a result, most emphasize the importance of LRU-style cache hits, which are extremely rare in disk caches. For example, the independent reference models of [4, 6] fail to take spatial locality into account. [7] used queueing to analyze cache write-back policies for workloads that are restricted to a Poisson arrival process. [29] analyzed write-only disk caches and derived equations to calculate the cache size needed to guarantee that all writes would be written with a zero-cost piggy-back write-back policy. They validated their model with a simple disk simulator, reporting "slight fluctuations" between the approximations and the simulator values. Although they used skewed workloads as well as uniform random ones, only equations for the latter are presented in the paper, the arrival process is restricted to be Poisson, and no sequential spatial locality effects were discussed.

Queueing theory is the main analytic approach that has been used for analyzing the effects of request queueing. [32] used an M/G/1 model for the FCFS scheduling algorithm; [14] extended this by providing a measure of locality of the requests for the FCFS scheduling algorithm, but did not show how to determine the locality measure, and it would be difficult to extend the model to include caching because of its direct dependence on the seek distance. [9, 23, 30] analyzed the SCAN and LOOK algorithms, assuming that the seek time is proportional to the seek distance and the workload is Poisson and has random uniform accesses. It does not seem straightforward to extend these analyses to use more realistic assumptions of disk behavior. [1] analyzed the FCFS algorithm for a drum, where the arrival process has a squared coefficient of variation close to 2; we have found in our studies of disk behavior that the squared coefficient of variation varies between 0.06 and 0.16.

Probably the closest models to ours are the disk models developed for use in disk array models (e.g., [8, 19, 21]). With a Poisson arrival process, an open queueing model, and random uniform accesses across the cylinders of the disk, these models are as good as ours. If any of these assumptions are untrue – i.e., if the workload is closed, or the arrival process is constant or bursty, or the workload is sequential – we can approximate the service times with smaller errors since we take these workload behaviors into account. For example, the models of [8, 19, 21] would approximate the device service time of a workload with 100% reads, a request size of 8KB, a run length of 4 requests, and a Poisson arrival process as 26.2ms for a HP 97560 disk (the running time of a random workload) whereas we would approximate it as 8.1ms. The measured service time is 8.3ms.

We know of no other analytic models with the same support for modern disk drives and ability to represent readahead and queueing effects across a range of workloads.

7 Conclusions

Our model supports all combinations of the following, with mean errors less than 17% for disk utilization less than 60% for the workloads tested.

- queueing, caching disk with readahead
- FCFS, LOOK, CLOOK, SSTF scheduling algorithms
- constant, Poisson, and bursty arrival processes
- contiguous, strided, and sparse types of spatial locality

This was shown by validating the model with real-world transaction processing workloads and synthetic video-on-demand workloads and parallel file system workloads for scientific applications. As a side-effect, we developed analytical models of caches and queues that can be used to model other parts of the I/O path. Our cache model includes predictions for the effects of readahead, and allowed us to approximate the behavior of workloads with sequential locality in the form of runs of contiguous reads. Given the performance effects of sequential disk accesses, this is probably the most important contribution of this work.

Future work Our model could usefully be extended in a number of directions. We are currently focusing on more general workload models, such as workloads with bursty arrival rates, on modeling disk array components, and on relaxing the requirement that only one workload is being serviced by a device at a time. We also are modeling the performance impacts of multiple disks on one SCSI bus [2].

Acknowledgments Our thanks to Marty Fouts, Pat Jacobson, David Jacobson, Alan Siegel, and our anonymous referees for comments on various drafts of this paper. David Kotz helped to define the synthetic workload for parallel scientific file systems.

References

[1] C. Adams, E. Gelenbe, and J. Vicard. An experimentally validated model of the paging drum. *Acta Informatica*, **11**:103–17, 1979.

[2] R. Barve, E. Shriver, P. B. Gibbons, B. K. Hillyer, Y. Matias, and J. S. Vitter. Modeling and optimizing I/O throughput of multiple disks on a bus. *ACM Conf. on Meas. and Modeling of Comp. Sys. (SIGMETRICS)/Perf.'98* (Madison, WI), June 1998. Extended abstract.

[3] A. L. Bastian. Cached DASD performance prediction and validation. *Proc. 13th Intl Conf. on Mgmt. and Perf. Eval. of Comp. Sys. (CMG)* (San Diego, CA), pages 174–7, M. Boksenbaum, G. W. Dodson, T. Moran, C. Smith, and H. P. Artis, editors, 14–17 Dec. 1982.

[4] A. K. Bhide, A. Dan, and D. M. Dias. A simple analysis of the LRU buffer policy and its relationship to buffer warm-up transient. *Proc. 9th Intl Conf. on Data Eng.*, pages 125–33, 23 Apr. 1993.

[5] G. Biagini. Evaluating I/O subsystem performance. *Proc. Intl Conf. for the Meas. and Perf. Eval. of Comp. Sys. (CMG'86)* (Las Vegas, NV), pages 299–306. Comp. Meas. Group, 1989.

[6] D. Buck and M. Singhal. An analytic study of caching in computer-systems. *J. of Par. and Distrib. Compt.*, **32**(2):205–14, Feb. 1996. Erratum published in volume **34**(2):233, May 1996.

[7] S. C. Carson and S. Setia. Analysis of the periodic update write policy for disk cache. *IEEE Trans. on Softw. Eng.*, **18**(1):44–54, Jan. 1992.

[8] S. Chen and D. Towsley. The design and evaluation of RAID 5 and parity striping disk array architectures. *J. of Par. and Distrib. Compt.*, **17**(1–2):58–74, Jan.–Feb. 1993.

[9] E. G. Coffman, Jr and M. Hofri. On the expected performance of scanning disks. *SIAM J. on Computing*, **11**(1):60–70, Feb. 1982.

[10] G. R. Ganger. *System-oriented evaluation of I/O subsystem performance*. PhD thesis, published as Technical report CSE–TR–243–95. Dept of Comp. Science and Eng., Univ. of Michigan, June 1995.

[11] R. Golding, E. Shriver, T. Sullivan, and J. Wilkes. Attribute-managed storage. *Workshop on Modeling and Specification of I/O* (San Antonio, TX), 26 Oct. 1995.

[12] C. C. Gotlieb and G. H. MacEwen. Performance of movable-head disk storage systems. *J. of the ACM*, **20**(4):604–23, Oct. 1973.

[13] P. G. Hoel, S. C. Port, and C. J. Stone. *Introduction to probability theory*. Houghton Mifflin Company, 1971.

[14] M. Hofri. Disk scheduling: FCFS vs. SSTF revisited. *Communications of the ACM*, **23**(11):645–53, Nov. 1980.

[15] A. Hospodor. Mechanical access time calculation. *Advances in Information Storage Systems*, **6**:313–36, 1995.

[16] D. M. Jacobson and J. Wilkes. *Disk scheduling algorithms based on rotational position*. Technical report HPL–CSP–91–7. Hewlett-Packard Labs., Palo Alto, CA, 24 Feb. 1991, revised 1 Mar. 1991.

[17] L. Kleinrock. *Queueing systems volume I: theory*. John Wiley and Sons, 1975.

[18] D. Kotz, S. B. Toh, and S. Radhakrishnan. *A detailed simulation model of the HP 97560 disk drive*. Technical report PCS–TR94–220. Dept of Comp. Science, Dartmouth College, NH, 18th July 1994.

[19] A. Kuratti and W. H. Sanders. Performance analysis of the RAID 5 disk array. *Proc. Intl Comp. Performance and Dependability Symp.*, pages 236–45, Apr. 1995.

[20] S. Lavenberg, editor. *Computer performance modeling handbook*. Academic Press, 1983.

[21] A. Merchant and P. S. Yu. Analytic modeling of clustered RAID with mapping based on nearly random permutation. *IEEE Trans. on Computers*, **45**(3):367–73, Mar. 1996.

[22] S. W. Ng. Improving disk performance via latency reduction. *IEEE Trans. on Computers*, **40**(1):22–30, Jan. 1991.

[23] W. Oney. Queueing analysis of the scan policy for moving-head disks. *J. of the ACM*, **22**(3):397–412, July 1975.

[24] C. Ruemmler and J. Wilkes. UNIX disk access patterns. *Proc. Winter 1993 USENIX* (San Diego, CA), pages 405–20, 25–29 Jan. 1993.

[25] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, **27**(3):17–28, Mar. 1994.

[26] B. Seeger. An analysis of schedules for performing multi-page requests. *Information Sys.*, **21**(5):387–407, July 1996.

[27] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. *Proc. Winter 1990 USENIX Conf.* (Washington, DC), pages 313–23, 22–26 Jan. 1990.

[28] E. Shriver. *Performance modeling for realistic storage devices*. PhD thesis. Dept of Comp. Science, New York University, May 1997.

[29] J. A. Solworth and C. U. Orji. Write-only disk caches. *Proc. ACM SIGMOD Conf.* (Atlantic City, NJ), pages 123–32, May 1990.

[30] T. J. Teorey and T. B. Pinkerton. A comparative analysis of disk scheduling policies. *Communications of the ACM*, **15**(3):177–84, Mar. 1972.

[31] F. Raab, editor. *TPC benchmark C, Standard Specification Revision 3.0*. Technical report. Transaction Processing Performance Council, 15 Feb. 1995.

[32] N. C. Wilhelm. An anomaly in disk scheduling: a comparison of FCFS and SSTF seek scheduling using an empirical model for disk accesses. *Communications of the ACM*, **19**(1):13–17, Jan. 1976.

[33] N. C. Wilhelm. A general model for the performance of disk systems. *J. of the ACM*, **24**(1):14–31, Jan. 1977.

[34] J. Wilkes. *The Pantheon storage-system simulator*. Technical Report HPL–SSP–95–14. Storage Systems Program, Hewlett-Packard Labs., Palo Alto, CA, 29 Dec. 1995.

[35] B. L. Worthington. *Aggressive centralized and distributed scheduling of disk requests*. PhD thesis, published as Technical report CSE–TR–244–95. Dept of Comp. Science and Eng., Univ. of Michigan, June 1995.

[36] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. *Proc. of ACM SIGMETRICS Conf. on Measurement and Modeling of Comp. Sys.* (Nashville, TN), pages 241–251, May 1994.