Optimal Write Batch Size in Log-Structured File Systems *

Scott Carson Tracer Technology Inc.
Sanjeev Setia George Mason University

ABSTRACT: Log-structured file systems create a performance benefit by maximizing the rate at which disk write operations can occur. However, this benefit comes at the expense of individual read operations, which can experience long queuing delays due to batched write service. This paper presents an analytic derivation of the write batch size that minimizes read response time, while retaining the benefit of write batching. Simulations that relax the assumptions of the analytic model demonstrate that the analytic result can be used to advantage in a practical setting.

* The work described in this paper was supported in part by a grant from the Digital Equipment Corporation. The views expressed herein are those of the authors. This work was performed while the authors were at the University of Maryland, College Park, MD.

1. Introduction

Recent studies [Rosenblum and Ousterhout 1992] have shown that log-structured file systems can exhibit substantial system performance gains over "traditional" file systems by batching write operations. On the other hand, in a recent paper [Carson and Setia 1992], we showed that bulk write operations can have an adverse effect on read performance experienced by processes. The conflict between these results is due to their different performance metrics; in the former, disk bandwidth utilization is considered important, whereas in the latter, response time for synchronous operations is of greater interest. In this paper, we continue to hold that response time is the more important metric, though we acknowledge that write batching can benefit both system and process performance. Based on this assumption, we derive the write batch size that minimizes read response time. Most significantly, we show that the optimal write batch size is largely insensitive to system load and that it is primarily dependent on disk characteristics. Thus, it is possible to create a nearly optimal write-batching policy based on statically determined parameters.

Log-structured file systems [Rosenblum and Ousterhout 1992; Ousterhout and Douglis 1989] are based on the idea that write operations can be postponed, in a disk cache, until there is a large backlog. Then the write operations are executed as a small number of disk operations, each of which transfers a large amount of data. This procedure amortizes a small number of seek and rotational latencies over a large number of data blocks and allows the log-structured file system to write data at a rate that approaches the transfer rate of the disk, given a sufficiently high write load.

Unfortunately, the performance benefits of this method do not extend directly to read operations, since in general, they are not part of the batching scheme. Our analysis of the "periodic update" write policy used in some operating systems showed that creating large sequences of write operations that are served in bulk produces an interruption of read service that causes read operations to experience long queuing delays. In our study, we considered only groups of write operations that were served singly, without the benefit of write batching. Although write batching does ameliorate the problem by reducing the collective service time of

a group of writes, it is clear that the same phenomenon can occur in write-batched systems, albeit to a lesser degree.

2. Solution

In nonbatching, write-bulk-arrival systems, one simple way to reduce read queuing delays is to give read operations nonpreemptive priority over writes, which eliminates the interruption of service caused by the write bulk. In write-batching systems, however, no such scheme is possible, because the writes are serviced as a small number of large disk operations (each a "segment" in LFS). One solution is to break up each batched write operation into a series of smaller batch operations and to give read operations nonpreemptive priority over these write batches, thereby introducing a fundamental conflict between write batch size and read response time.

Suppose that a read request arrives at the disk queue and that a series of write requests (each of which represents a batch) is already in queue. Minimizing the size of these write batches (and increasing the number of batches accordingly) then minimizes the waiting time for that read operation, since the read must only wait for the batch write operation in service to complete. If a read request arrives when no writes are in queue, then it is serviced immediately. Maximizing the size of these write batches (and reducing their number correspondingly) minimizes the fraction of time the disk is busy servicing writes and therefore minimizes the probability that an arriving read will find the system busy. This situation suggests that there is an optimal write batch size that balances the ability of reads to "slip in" between write operations with the need to complete write operations quickly. The derivation of this optimal write batch size is the focus of this paper.

Our strategy for finding the optimal write batch size is as follows. First, we express the read response time as a function of write batch size (assuming read priority) analytically, under simplifying assumptions. Next, we derive the optimal write batch size and show it to be dependent only on disk characteristics: seek time, rotational latency, and transfer rate. Recognizing that the assumptions used in the analytic model may not hold in practice, we then relax the assumptions and simulate the system. Our simulations show that the optimal write batch size predicted by the analytic model is a reasonable estimate of the optimal write batch size in a realistic system. Thus, the results of the analytic model can be used to drive a disk service policy in a nearly optimal way. Next, we discuss the implementation of this scheme and show example calculations based on our model and on the characteristics of several popular disks.

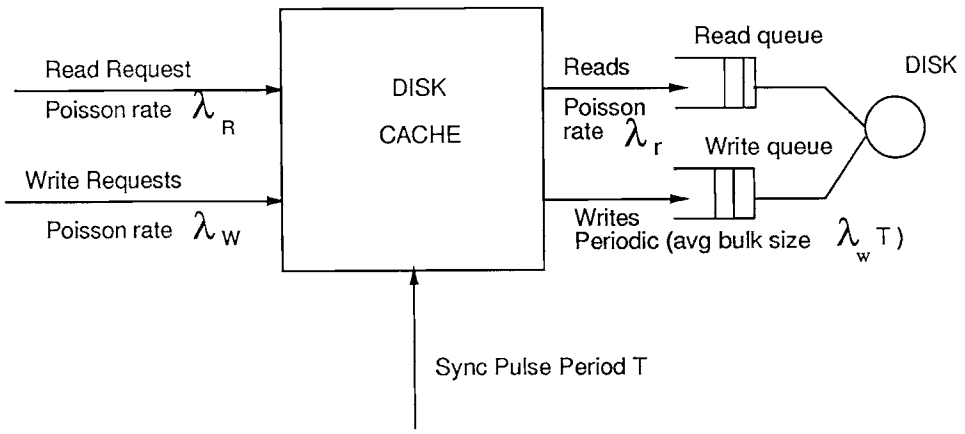

Figure 1. The model.

3. Analytic Model

This section presents an analytic model of response time for read operations for the disk service scheme described in the previous section. Under this scheme, read operations are given nonpreemptive priority over write operations, and the write batch size is a policy parameter. In concept, the disk server maintains separate queues for read and write requests (see Figure 1).

We assume, for now, that read operations arrive at the cache according to a Poisson process with a rate of λ_R requests/s. In the model, these requests enjoy a fixed cache hit ratio of h_r, $0 \leq h_r \leq 1$ so that the resulting disk read traffic is a Poisson process with arrival rate $\lambda_r = (1 - h_r)\lambda_R$.

A typical method of batching writes is to save them in a main memory cache and flush them on a periodic basis in response to a "sync pulse." LFS [Rosenblum and Ousterhout 1992] is a "pure" periodic update policy; all write operations are delayed and written out at periodic intervals. We assume that writes arrive at the cache according to a Poisson process with rate λ_W. Let T be the length of the interval between two updates. The write hit ratio $h_w(T)$, $0 \leq h_w(T) \leq 1$ is the fraction of writes to blocks that are already dirty or that will not be written to disk due to deletion. At the beginning of each update period, the dirty blocks are written to disk. Under LFS, the dirty blocks in the cache are aggregated into "segments" of a fixed size and sent to the disk queue; under our modified scheme, these segments are further divided into batches of size c blocks. The number of dirty blocks written at the update interval is Poisson distributed with mean $\overline{A} = \lambda_w T = (1 - h_w(T))\lambda_W T$. For a batch size of c blocks, $[\frac{\overline{A}}{c}]$, full batches and (at most) one partial batch are placed in the write disk queue. To simplify

our analysis, we assume that the number of blocks written out every T second is always $\overline{A}$; thus the partial batch has $p = \overline{A} \bmod c$ blocks.

The disk service time for writing out a batch (or a partial batch) is a function of the batch size. The service time, $s(c)$ can be written as

$$s(c) = sk + rot + tr(c)$$

where sk is the seek time, rot is the rotational latency and $tr(c)$ is the transfer time. The transfer time is a function of the batch size c, as well as the characteristics of the disk, and the seek time and rotational latency depend only on disk characteristics. In general, the individual random variables that compose the service time are not independent. However, for the purpose of this analysis we shall assume that seek time and rotational latency are independent and have stationary probability distributions. Hence, the average service time for writing out a batch of size c is given by

$$\overline{s}_w = \overline{sk} + \overline{rot} + \overline{tr}(c) \tag{1}$$

Similarly, the average service time for writing out a partial batch containing p blocks is

$$\overline{s}_p = \overline{sk} + \overline{rot} + \overline{tr}(p)$$

In our model successive read requests are independent. In reality, this may not always be the case and reads may benefit from various optimizations. However, in our model the average read service time is given by

$$\overline{s}_r = \overline{sk} + \overline{rot} + \overline{tr}(1)$$

where $\overline{tr}(1)$ is transfer time for one block.

The variance of the service time for a disk operation is given by the sum of the variances of the seek time, rotational latency, and the transfer time. In general, the transfer time distribution is deterministic. We assume that

$$\overline{tr}(c) = k \cdot c \tag{2}$$

where k is a constant that depends on the disk. Thus, the service time variance for an operation is the sum of the seek time and rotational latency variances. That is, for a disk operation o where $o = w, p$ or r denoting a full batch write, a partial batch write, or a read, respectively, we have

$$\sigma_o^2 = \sigma_{sk}^2 + \sigma_{rot}^2$$

Note that the variance in the service time for disk reads, full batch writes, and partial batch writes is the same. Henceforth we shall drop the subscript o in σ_o^2.

Then, the coefficient of variation of service time for a disk operation o is given by

$$C_{s_o}^2 = \frac{\sigma^2}{\overline{s}_o^2} \tag{3}$$

We can see from the expression above that increasing the batch size c results in a reduction in the coefficient of variation of the write service time.

The system shown in Figure 1 can be considered a head-of-line priority queuing system with three classes of customers: reads, full batch writes, and partial batch writes, with reads having priority over the other two classes of customers.

The average response time for reads can be readily derived from results in Kleinrock [1976] for a k-class priority queuing system:

$$\overline{R} = \overline{s}_r + \frac{\rho_r \overline{s}_r(1 + C_{s_r}^2) + \rho_w \overline{s}_w(1 + C_{s_w}^2) + \rho_p \overline{s}_p(1 + C_{s_p}^2)}{2(1 - \rho_r)} \tag{4}$$

where

$$\rho_w = \lfloor \frac{\overline{A}}{c} \rfloor \cdot \frac{\overline{s}_w}{T}$$

and

$$\rho_p = \begin{cases} \frac{\overline{s}_p}{T} & \text{if } \overline{A} \bmod c > 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that the read response time depends on the write load but not on the manner in which the writes are presented to disk. Thus, the analysis applies equally to any scheme for aggregating writes and is not restricted to the "periodic update" policy. The write load is divided into two components: the write load due to full batch writes (ρ_w) and the write load due to partial batch writes (ρ_p). Note further that the disk utilization due to write batches is a function of the batch size, whereas the disk utilization due to reads is independent of the batch size. We define the *offered load* due to writes as $\varrho_w = \lambda_w \cdot \overline{s}$, where $\overline{s}$ is the service time for writing a single block.

A number of simulations were run to validate the model; all simulation results reported are with 95 percent confidence intervals of 1 percent about the mean. For the simulations, we considered a hypothetical disk with 815 cylinders, in which the seek time function $sk(d) = 1.775 + 0.725d^{0.5}$ ms, where d is the seek distance in cylinders. This condition results in a seek time of 2.5 ms for a single-cylinder seek and 22.5 ms for an end-to-end seek. We assume that the rotational latency is uniformly distributed between 0 and 16 ms. The block size is 8 kilobytes, and the disk transfers data at 2 Mbyte/s, so that the transfer time is given by $tr(c) = 4c$ ms, that is, the transfer time for one block is 4 ms. In Figure 2(a) we compare the results of our model with simulations. As the figure

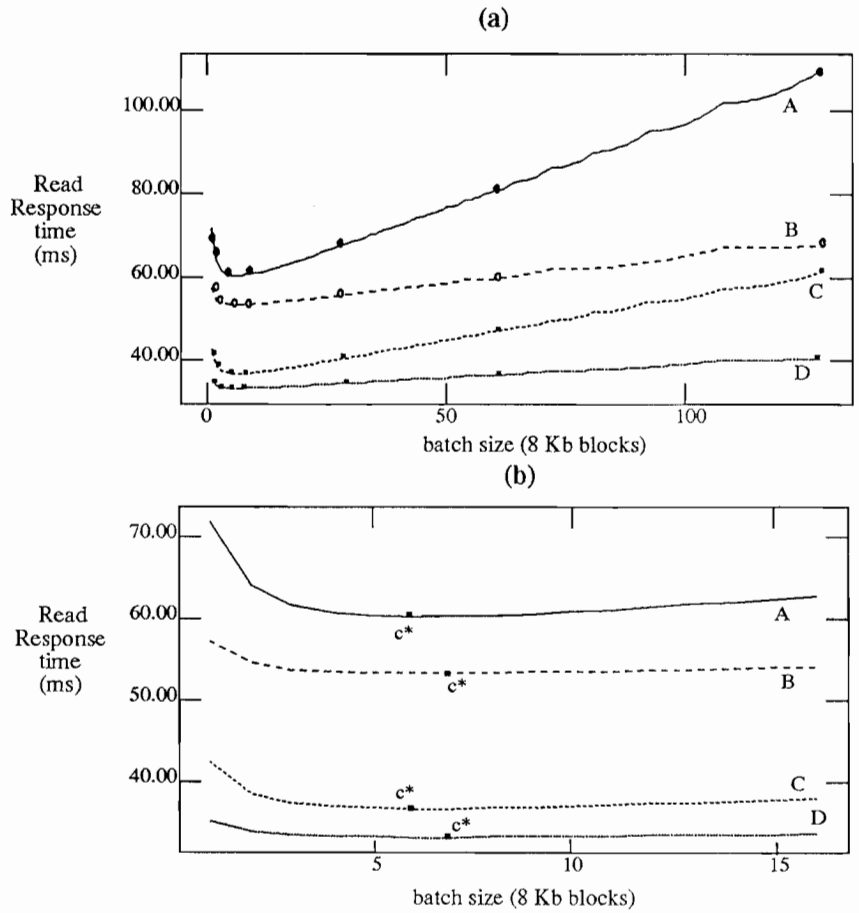

Figure 2. (a) Read Response time (in ms) versus batch size (in 8 Kb blocks) assuming Poisson read arrivals. Curves are from the analytical model: marked points are simulation results. A: $\rho_r = 0.6$, $\varrho_w = 0.6$; B: $\rho_r = 0.6$, $\varrho_w = 0.2$; C: $\rho_r = 0.2$, $\varrho_w = 0.6$; D: $\rho_r = 0.2$, $\varrho_w = 0.2$. (b) shows the same curves blown up for batch sizes less than 16. c^* denotes the batch size that minimizes read response time. The curves are linearly interpolated between integral batch-size values.

shows, our model predicts the read response time quite accurately. Note that the simulations and the model share all assumptions about read and write arrival patterns and service times. In the simulation, however, gathered statistics are based on read and write arrivals, not the mean number of reads and writes as used in the model.

As Figure 2(b) shows, the response time appears to have a minimum for a certain batch size, irrespective of the read and write loads. We now explore this observation analytically.

The batch size c can take on only integral values. Hence, Equation (4) for the read response time is only meaningful if c is an integer. For the moment, however, let us assume that c is a real number, and that there are no partial batch writes. Then Equation (4) can be rewritten as a continuous function of the batch size, c:

$$\overline{R}(c) = \overline{s}_r + \frac{\rho_r \overline{s}_r(1 + C_{s_r}^2) + \rho_w \overline{s}_w(1 + C_{s_w}^2)}{2(1 - \rho_r)} \tag{5}$$

Substituting $\rho_w = \frac{\lambda_w \overline{s}_w}{c}$ and for $\overline{s}_w$ and $C_{s_w}^2$ from Equations (1) and (3), respectively, we have

$$\overline{R}(c) = \overline{s}_r + \frac{\rho_r \overline{s}_r(1 + C_{s_r}^2)}{2(1 - \rho_r)} + \frac{\lambda_w}{2(1 - \rho_r)c}[\overline{s}_w^2 + \sigma^2] \tag{6}$$

$$= \overline{s}_r + \frac{\rho_r \overline{s}_r(1 + C_{s_r}^2)}{2(1 - \rho_r)} + \frac{\lambda_w}{2(1 - \rho_r)c}[(\overline{sk} + \overline{rot} + k \cdot c)^2 + \sigma^2] \tag{7}$$

$$= \overline{s}_r + \frac{\rho_r \overline{s}_r(1 + C_{s_r}^2)}{2(1 - \rho_r)} + \frac{\lambda_w}{2(1 - \rho_r)}[\frac{D^2 + \sigma^2}{c} + k^2 c + 2Dk] \tag{8}$$

where $D = \overline{sk} + \overline{rot}$.

Differentiating $\overline{R}(c)$ with respect to the batch size c, we have

$$\frac{\partial \overline{R}(c)}{\partial c} = \frac{\lambda_w}{2(1 - \rho_r)}[k^2 - \frac{D^2 + \sigma^2}{c^2}] \tag{9}$$

Setting the derivative equal to 0, we obtain the optimal batch size c^*

$$c^* = \left(\frac{D^2 + \sigma^2}{k^2}\right)^{0.5} \tag{10}$$

What is remarkable about the preceding result is that the batch size that minimizes the read response time is independent of the read and write loads and only depends on the characteristics of the disk. For the simulations plotted in Figure 2, $\sigma^2 = 44.2305$, $k = 4$ blocks/ms and $D = 23.75$, leading to $c^* = 6.17$. However,

as the batch size can take only on integral values, we have $c^* = 7$ or $c^* = 6$. Figure 2(b) shows that the read response time is minimized for a batch size of 6 or 7 for all combinations of read and write loads.

Although the write batch size that minimizes read response time is independent of the read and write loads, the *rate* at which the read response time increases as the batch size increases beyond c^* depends on these loads (see Equation 9). Thus, in Figures 2(a) and 2(b), we observe that the read response time grows at a much faster rate with increasing batch size for the cases A and C ($\varrho_w = 0.6$), than in the cases B and D ($\varrho_w = 0.2$). We note that under the disk service scheme modeled in this section, reads have priority over write batches, which is the best-case scenario from the point of view of read operations. Under disk service schemes in which reads do not have priority over write batches, the rate at which the read response time grows with increasing write batch size is always larger than under our scheme.

4. Relaxing the Assumptions

The model presented in the previous section has several simplifying assumptions that allow it to be analyzed. In particular, we have assumed that requests arrive in a Poisson stream and that successive requests are independent. Further, we assume that read requests are serviced in a first-come-first-serve fashion. If the results obtained through the model are to prove useful, we must be sure that the results will also be true for real systems. In this section, we relax several assumptions made in the previous section and examine their impact on the results of the last section. Because relaxing the assumptions makes our model intractable, the results in this section are obtained only through simulation.

All simulations reported in this section were obtained using a simple event-driven simulator. Read operations were generated according to either an exponential or a hyperexponential interarrival-time distribution. Writes were generated similarly but were "saved up" for 30 seconds and then "flushed" to the disk queue. Disk service times were generated by adding a uniformly distributed, random rotational latency, the seek time corresponding to a uniformly distributed, random "next cylinder" and the transfer time corresponding to either a single block (for reads) or a batch-sized block (for writes). Disk characteristics were those described in the previous section. Simulations were run long enough to provide 95 percent confidence intervals of 1 percent about the mean.

The first assumption we address is that of Poisson read and write arrivals to the disk cache, leading to Poisson read arrivals to the disk and to the number of blocks being written out to the disk at the sync pulse being Poisson distributed.

Most experimental studies of I/O traffic for UNIX-like file systems have observed that both read and write traffic is quite bursty [Ousterhout et al. 1985; Baker et al. 1991]. In order to model this burstiness, in our simulations we now assume that both reads and writes arrive according to a hyperexponential process, which has a coefficient of variation (C_a^2) > 1. This assumption has two effects: First, reads arrive in a much more bursty fashion (depending upon the coefficient of variation), and second, there is a larger variance in the number of blocks written to the disk at each sync pulse.

The results of the simulations are plotted in Figure 3. As the graph shows, the optimal batch size for various read and write load combinations is no longer the same. Thus, the result derived in the previous section that the optimal batch size is independent of the load is not true for non-Poisson read and write arrivals. However, what is noteworthy about the plots in Figure 3 is that the optimal batch size is still strongly dependent upon disk characteristics. For all the read and write load combinations in the figure, the optimal batch size is within two blocks of the optimal batch size for the model in the last section. The effect of the bursty read arrivals is to push the optimum toward a smaller batch size. As Figure 4 shows, the effect of the bursty writes is less pronounced (compare with Figure 2). This finding is understandable because the writes are aggregated by the cache and only delivered to the disk at the beginning of the update interval. These results show that the optimal batch size obtained from the model in the previous section can be used by file system designers to get an idea about the optimal batch size for real systems.

We now relax another assumption of our model. In the model and in the simulation just described, the disk service time for writing a batch is the sum of the seek time, rotational latency, and transfer time. Although this assumes that in writing out a batch, one always has to pay the price of a seek, in practice, it is only true when a read request(s) arrives during the time a batch is being written out. Because the read has higher priority, the read request(s) will be serviced next, thus necessitating a seek to get back to write the next batch of the segment. However, if no read request arrives while the write batch is being written, the next batch can be written out without a seek because the head already will be positioned on the right track. Thus, only a rotational latency and transfer time cost must be paid for writing the next batch.

The results of the simulation that take this effect into account are shown in Figures 5 and 6. We observe that the average read response time is not affected substantially. The optimal batch size, however, has been pushed even further to the left than in Figure 3 because the effect of the "zero-seeks" is to reduce the value of the constant D (as the average seek distance is reduced), thus leading to a

reduction in the optimal batch size. In Figure 6, the optimal batch size appears to be two or three blocks, depending on the read and write loads.

Finally, we consider another possible optimization in real systems. If we fix the batch size such that the batch fits exactly on a track or on multiple tracks, then while writing out a batch of the segment, we no longer have to pay a cost for rotational latency because as soon as the disk head is in position on the track to be

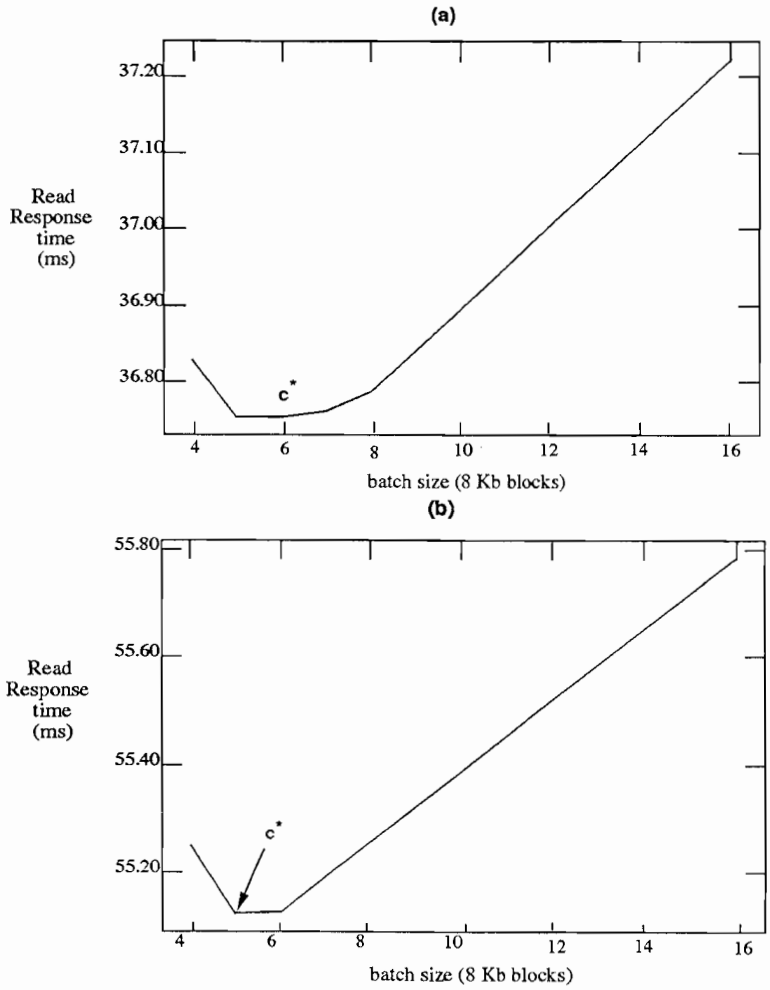

Figure 3. Read Response time (in ms) versus batch size (in 8 Kb blocks) for hyperexponential read arrivals with $C_a^2 = 5$ for (a) $\rho_r = 0.2, \varrho_w = 0.2$ and (b) $\rho_r = 0.4, \varrho_w = 0.2$. Note that the batch size can only have integral values. Curves are linearly interpolated between batch-size values.

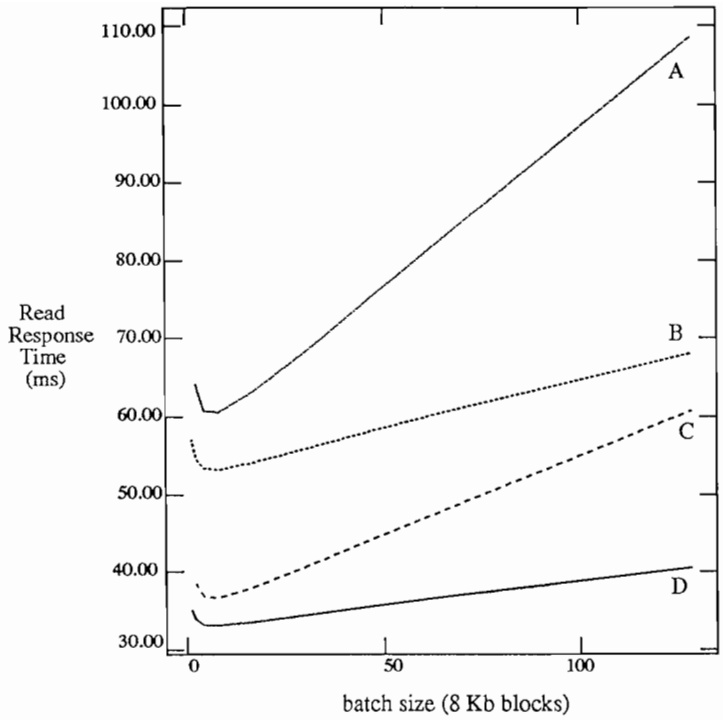

Figure 4. Read Response time (in ms) versus batch size (in 8 Kb blocks) for hyperexponential write arrivals with ($C_a^2 = 5$) to the cache. A: $\rho_r = 0.6$, $\varrho_w = 0.6$; B: $\rho_r = 0.6$, $\varrho_w = 0.2$; C: $\rho_r = 0.2$, $\varrho_w = 0.6$; D: $\rho_r = 0.2$, $\varrho_w = 0.2$. Curves are linearly interpolated between batch-size values.

written, it can initiate the disk transfer. Thus, in this case one has to pay only the cost of seek (if even that) and data transfer.

Simulation results for this optimization are shown in Figure 7. As expected, the simulations show that read response time is reduced for batch sizes that are multiples of four 8-Kbyte blocks but otherwise is unaffected. With this optimization in place, the optimal batch size may be a multiple of the track size, as it is in this case.

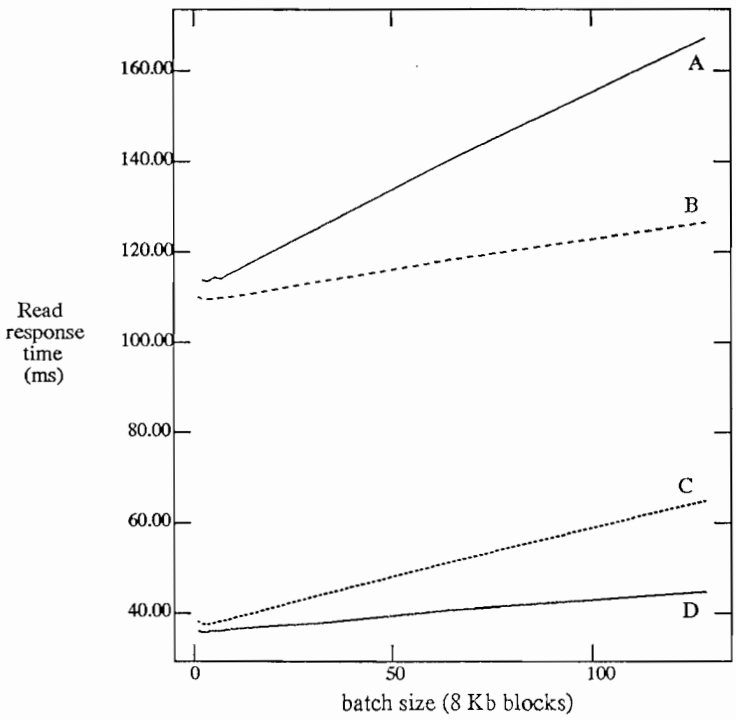

Figure 5. Read Response time (in ms) versus batch size (in 8 Kb blocks) taking "zero-seeks" into account. A: $\rho_r = 0.6$, $\varrho_w = 0.6$; B: $\rho_r = 0.6$, $\varrho_w = 0.2$; C: $\rho_r = 0.2$, $\varrho_w = 0.6$; D: $\rho_r = 0.2$, $\varrho_w = 0.2$. Curves are linearly interpolated between batch-size values.

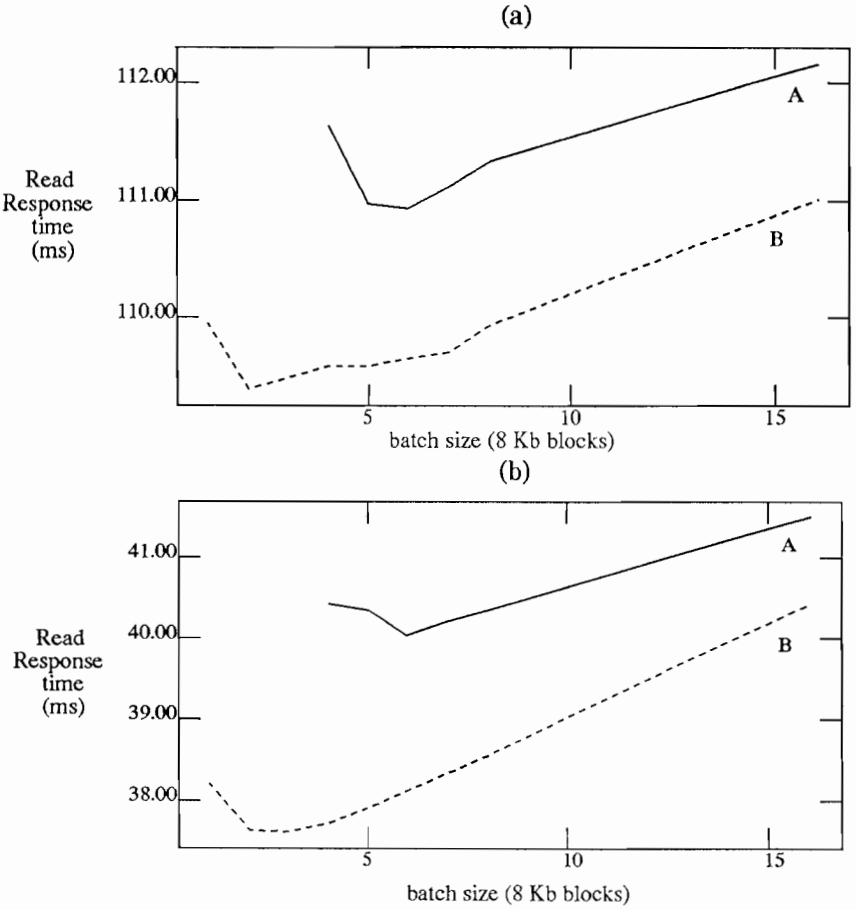

Figure 6. Read Response time (in ms) versus batch size (in 8 Kb blocks) taking "zero-seeks" into account for (a) $\rho_r = 0.6$, $\varrho_w = 0.2$ and (b) $\rho_r = 0.2$ $\varrho_w = 0.6$. Curves are linearly interpolated between batch-size values. A: hyperexponential read arrivals ($C_a^2 = 5$); B: hyperexponential read arrivals and "zero-seeks."

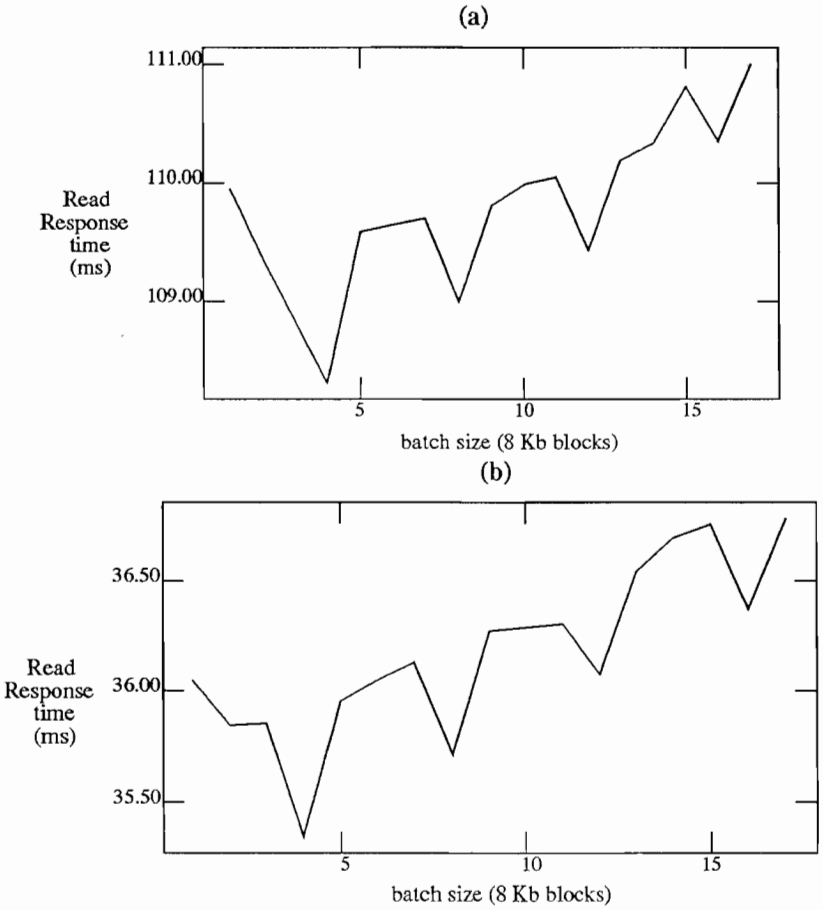

Figure 7. Read Response time (in ms) versus batch size (in 8 Kb blocks) assuming track-sized batches for (a) $\rho_r = 0.6$, $\varrho_w = 0.2$ and (b) $\rho_r = 0.2$ $\varrho_w = 0.2$. Curves are linearly interpolated between batch-size values.

5. *Discussion*

The results of this study can help to create a parameterized service policy for disk systems that provides the advantages of write batching while (nearly) minimizing average response time for read operations. This section considers several practical issues: use of the results, implementation, and example policy parameters.

We have considered several realistic situations: bursty read and write arrivals, taking advantage of disk head position when servicing writes, and writing complete tracks at a time. In each case, these conditions introduced some deviation from the analytic model; in particular, the actual response times were substantially different from those predicted by the model. However, the optimal batch size was similar to (actually, smaller than) the predicted value, thus suggesting that it is possible to derive general policy guidelines from the model.

The write-batching policy described in this paper is easy to implement. The policy is driven by a single parameter c^* that describes the optimal batch size. The log-structured file system does not need to know about this policy; instead, it is implemented in the disk driver itself. Suppose that a log-structured file system delivers groups of write operations to the disk queue. These groups of write operations can be of arbitrary size. The disk driver then repeatedly issues batches of writes to the disk, each batch containing n sectors. Between each write batch, however, the driver checks for the presence of read requests in the disk queue and services all reads before servicing the next write batch.

Since the nearly optimal value of n is fixed for a given disk, it is possible for the disk driver to use a table of batch sizes, indexed by disk type, to govern the policy. Table 1 shows c^* for three popular one-gigabyte disks.

The values for seek time mean and variance were calculated from manufacturers' specifications of minimum (track-to-track) and maximum (end-to-end) seek times, assuming a uniform, random seek-distance distribution and constant head acceleration/deceleration. The values for rotational latency mean and variance were calculated based on the disk rotation speed, assuming a uniform rotational latency distribution. Transfer rates were calculated based on disk rotation speed and track density. The optimal cluster size $c*$ is reported in kilobytes rather than 8-kilobyte blocks as in the previous sections.

Although these disks are similar enough that they might be considered interchangeable in practice, their characteristics produce distinct effects that agree with intuition about the optimal write-cluster size. The IBM disk, for example, seeks and rotates more quickly than the others. This leads to a smaller optimal cluster size, since the penalty for breaking up a larger cluster is smaller. The HP disk seeks and transfers more slowly than the Fujitsu disk, but because it rotates more

quickly, the penalty for breaking up a cluster is again smaller, leading to a smaller optimal cluster size. Finally, because the Fujitsu disk transfers data more quickly than the others, more data per disk operation must be transferred to make the seek and rotational penalties insignificant.

Table 1. Optimal Batch Sizes (in Kbytes) for Three Contemporary Disks.

	Drive Model		
	IBM 0663-H12	HP 97549	Fujitsu M2266S/H
Size (Gbytes)	1	1	1
Mean seek (ms)	14.40	24.90	21.1
Seek variance (ms^2)	18.90	61.60	39.6
Mean rotation (ms)	6.95	7.47	8.3
Rotational variance (ms^2)	16.10	18.60	23.2
Transfer rate (Mb/s)	2.40	2.20	2.5
c^* (Kbytes)	52	72	74
Realized write Transfer rate (Mb/s)	1.20	1.10	1.3

In practice, we can expect that the optimal cluster size will be slightly smaller than c^*, since c^* is calculated without the effects of seek optimization, track caching, and bursty arrivals. Although more investigation is needed before firm policy guidelines can be presented, the results of the previous section show that the deviation from c^* under realistic conditions will be small. However, we must also note that the larger the batch size, the more write load the system can sustain. Thus, it may be appropriate to ignore those effects that reduce the optimal batch size and use c^* instead. In addition, for systems that can take advantage of full-track writes, the best choice for the write batch size may be a multiple of the track size.

Another measure shown in Table 1 is the realized write transfer rate, which is calculated as the ratio of c^* to the sum of seek, rotational, and transfer delays. Because the realized write transfer rate is substantially smaller than the maximum transfer rate provided by the disk, systems that require sustained, high write rates may require a larger batch size than c^*. As pointed out in Section 3, the rate at which the read response time increases as the batch size increases beyond c^* depends on the read and write loads. By selecting a batch size greater than c^*, a

system designer can trade off an increase in the read response time for an increase in the write throughput. Depending on the load offered to the disk, a batch size greater than c^* can achieve a much higher write rate, while resulting in a relatively small increase in read response times.

Finally, we note that as disk technology evolves, optimal write batch sizes will change. Although it is difficult to predict the exact nature of future optimizations, we can expect disks to become increasingly dense and offer higher transfer rates. If seek and rotational latencies remain similar to their current values, then c^* will increase in the future. Likewise, synchronized multidisk systems, designed for high transfer rates, will require larger write batches than single-disk systems of similar technology.

6. Conclusions

The results of this paper demonstrate that if read response time is an important performance metric, we must take care in the design of write-batching file systems. Specifically, making the write-cluster size too large penalizes reads by making them wait for long write operations, whereas making the write-cluster size too small penalizes reads by wasting disk bandwidth. The optimal cluster size is dependent on load and access patterns but is dominated by a function of disk characteristics.

Although our results are preliminary, in that they have not been confirmed by measurement on a real system, they do suggest that it is possible to devise a batching policy that works well with statically determined parameters. Further, our results contrast sharply with some of the basic design objectives of log-structured file systems, such as maximizing the size of write batches to increase throughput. Although there are cases in which write throughput is the most important performance criterion, in most general-purpose environments read response time is more important. Our results indicate that in such cases batch sizes should be considerably smaller than they would be otherwise.

References

1. M. Baker, J. Hartman, M. Kupfer, K. Shirrif, and J. Ousterhout. Measurements of a Distributed File System. In *Proceedings of the 13th Symposium on Operating System Principles*, 198–212, 1991.
2. S. D. Carson and S. K. Setia. Analysis of the Periodic Update Policy for Disk Cache. *IEEE Transactions on Software Engineering*, 44–54, January 1992.

3. L. Kleinrock. *Queuing Theory II: Applications*. Vol. 2. New York: J. Wiley and Sons, 1976.
4. J. Ousterhout and F. Douglis. Beating the I/O Bottleneck: The Case for Log-Structured File Systems. *Operating Systems Review*, 11–28 January 1989.
5. J. Ousterhout, H. da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A Trace-Driven Analysis of the Unix 4.3 BSD File System. In *Proceedings of 10th Symposium on Operating System Principles*, 15–24, 1985.
6. M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM TOCS*, 10(1), 26–52, February 1992.