Disk Scheduling Revisited

Margo Seltzer, Peter Chen, John Ousterhout

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

ABSTRACT

Since the invention of the movable head disk, people have improved I/O performance by intelligent scheduling of disk accesses. We have applied these techniques to systems with large memories and potentially long disk queues. By viewing the entire buffer cache as a write buffer, we can improve disk bandwidth utilization by applying some traditional disk scheduling techniques. We have analyzed these techniques, which attempt to optimize head movement and guarantee fairness in response time, in the presence of long disk queues. We then propose two algorithms which take rotational latency into account, achieving disk bandwidth utilizations of nearly four times a simple first come first serve algorithm. One of these two algorithms, a weighted shortest total time first, is particularly applicable to a file server environment because it guarantees that all requests get to disk within a specified time window.

1. Introduction

Present day magnetic disks are capable of providing I/O bandwidth on the order of two to three megabytes per second, yet a great deal of this bandwidth is lost during the time required to position the head over the requested sector. This study focuses on improving the effective throughput by using rotation and seek optimizing algorithms to schedule disk writes.

Since the introduction of the movable head disk, many people have undertaken similar efforts. However, most of these studies have assumed short queue lengths, and the performance improvement obtained under the various techniques is not substantial. Our approach was to consider a system, such as a file server, with a large main memory dedicated to disk buffering. We assumed that newly written data need not be transmitted to disk immediately; instead, it may be retained for a short period of time in a main memory buffer and transmitted to disk at a time that maximizes disk throughput. Given the ever increasing sizes of main memory (up to one hundred megabytes or more on some file servers), hundreds or thousands of blocks could be queued for writing at any given time. By careful ordering of these requests, it should be possible to reduce average head positioning time substantially. On the other hand, the potential for starvation of a request becomes more important and fairness becomes a requirement. To this end, we have developed two algorithms that attempt to avoid starvation yet provide very good disk utilization.

2. Previous Work

Most previous work has dealt with scheduling a small number (fewer than 50) of I/O requests. With small numbers of requests, research concentrated on first come first serve (FCFS), shortest seek time first (SSF), and the scanning algorithms which service requests in cylinder order scanning from one edge of the disk to the other.

Hofri shows that under nearly all loading conditions, SSF results in shorter mean waiting times than FCFS [HOFR80]. The main drawback he finds to SSF is the larger variance in I/O response time. He also alludes to more optimal scheduling which takes into account the number of requests in a given cylinder, but does not pursue this further. Hofri's results are a combination of theoretical analysis and simulation.

Coffman, Klimko, and Ryan also discuss FCFS and SSF [COFF72]. They add to their analysis two scheduling policies which are intended to control the high variance of SSF. These are called SCAN and FSCAN. SCAN restricts its search for the minimum seek time request to one direction (inward or outward). However, SCAN still causes long waiting times for requests on the extremes of the disk. FSCAN addresses this by "freezing" the queue once the scan starts--requests that arrive after the scan starts are serviced in the next scan. By pure theoretical analysis, Coffman et al. concludes that SCAN uniformly results in lower average response times than either FCFS or FSCAN, but higher average response times than SSF. Geist describes a continuum of algorithms from SSF to SCAN differing only in the importance attached to maintaining the current scanning direction [GEIS87].

In [TEOR72], FCFS, SSF, and SCAN are again analyzed. Similar conclusions are made that SSF yields shorter response times than SCAN, which yields shorter response times than FCFS. The Eschenbach scheme, which is similar to SCAN, schedules according to rotational position in addition to seek position. As a result, the Eschenbach scheme generates lower average response than any previous scheme as the queue length increases.

In all of these papers, no queue lengths averaging more than 50 are studied. This limitation is due in large part to the smaller memory sizes of the time and slower CPU's. Now, with exponentially growing memory sizes [MOOR65] and faster CPU's, more data may accumulate more quickly, and disk queues are no longer constrained to small lengths. With large queues, we are able to investigate previously impractical or unnecessary schemes. In particular, we continue the study of rotationally optimal scheduling algorithms.

3. The Test Environment

We chose to analyze the algorithms in three ways: theoretical model, simulation, and hardware tests. The theoretical model served as a first approximation of the potential performance gain. The simulation provided the most flexible testing platform, and the hardware tests verified the correctness of the simulator. After the validation of the simulator on some of the simpler algorithms, the remaining results were all derived from simulation.

3.1. The Simulator

The simulator modeled a Fujitsu M2361A Eagle described in Figure 1 and Table 1. In all the simulations, the CPU time required to calculate the next request was ignored on the basis that this computation could be overlapped with the actual I/O operation. Furthermore, we wished to focus on the potential of the algorithms themselves rather than optimizing their implementation.

Since we were most interested in viewing the behavior of the algorithms in the presence of many requests, we introduced an artificial model of request arrival. In order to examine behavior for a queue length of Q, we initialized the queue to contain Q events, each with a request time of 0. Whenever a request was serviced, it was replaced with a new request whose request time was equal to the completion time of the completed request. In this manner, we guaranteed that we always had a queue of length Q from which to select a request and our simulations were insensitive to the real arrival rate. In order to avoid skewing the response time results (by leaving unserviced requests in the queue at simulation completion), we completed the simulation by emptying the entire queue. That is, for the last Q requests, we did not generate any new requests, but

Fujitsu Eagle disk drive	
cylinders/disk	840
tracks/cylinder	20
sectors/track	67
bytes/sector	512 B
average seek	18 ms
average rotational latency	8.3 ms
time to transfer 4 KB	2 ms

Table 1: Specifications of Fujitsu disk drives.

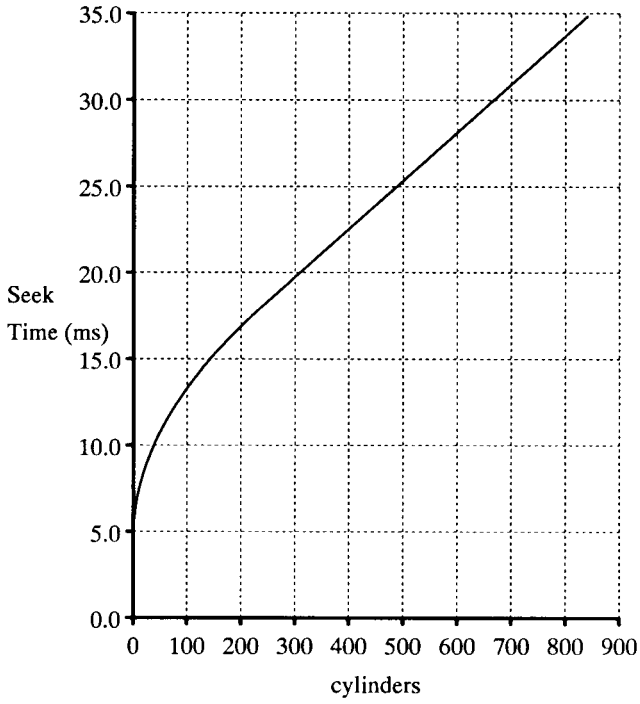

Figure 1: Seek Time Calculation Graphed above is seek time in ms as a function of seek distance in cylinders [FUJI84]. We model this as

$$seektime(x) = \begin{cases} 0 & \text{if } x = 0 \\ 4.6\,ms + .87\,ms\sqrt{x} & \text{if } x \le 239 \\ 18\,ms + .028\,ms\,(x - 239) & \text{if } x > 239 \end{cases}$$

serviced those remaining in the queue.

In order to guarantee that the averages obtained were statistically significant, we needed to determine an acceptable length for the simulation runs. Let Q be the length of the queue and B be the total number of blocks on the disk. At each point in time, there are Q objects selected from a set of B, in the queue. Therefore the probability of any particular set of Q objects being present is $\frac{1}{\binom{B}{Q}}$. For the drives we tested, there were 140,280 blocks on the disk. So, for a queue length of 10, there are on the order of 10^{44} combinations and for a queue of length 1000, there are more than 10^{100} combinations. Clearly, it is infeasible to actually examine a large portion of this space. Furthermore, in our simulations, each sample of size Q is not independent since the entries in the queue at time *t* differ from the Q entries in the queue at time *t-1* by precisely one event.

We define a test run as one simulation which generated a full queue of random I/O requests and serviced them. We analyzed the variance across N runs where N ranged from 2 to 200. After 100 runs of size Q, the variance had decreased to a small fraction (1-2%) of the mean response time and had stabilized. Therefore, we felt that tests with 100 times the number of queued items was a representative sample.

Our model of the I/O time was based on the information provided by the disk vendor as well as the results obtained in the disk tests described below. All requests to the disk subsystem were for 4K blocks[1] uniformly distributed over the entire disk. The seek time was computed as a function of the number of cylinders across which the head needed to move (see Figure 1). The rotational latency was calculated based on the time required to bring the data under the head once the seek was completed and then to read the data.

3.2. The Hardware Tests

To verify our theoretical models and our simulations, we ran tests on the disks that we were modeling, Fujitsu M2361A Eagles, described in Table 1 and Figure 1. We verified the simulation for the two basic scheduling algorithms: FCFS and SSF. Because of the difficulty in determining rotational position, we did not use hardware to verify the simulation for other algorithms. As in the simulation, we assume zero CPU time spent to process the I/O. In order to factor out the cpu time, we subtracted a constant 3 ms from each individual disk access[2].

4. Seek Optimizing Algorithms

We started with algorithms that optimize solely on seek distances: first come first serve, shortest seek first, and the scanning algorithms. For each algorithm we evaluated, we provide a brief description of the algorithm, an intuitive theoretical estimate of its performance (where feasible), the actual results, and a graph comparing the simulated versus theoretical results. Our metric for evaluating the algorithms was disk utilization, which we define as the fraction of time that the disk spends transferring data.

4.1. First Come First Serve

The simplest scheduling algorithm imaginable is first come first serve (FCFS). As one would expect, this model is independent of the queue length and we obtain an average I/O time equal to the predicted average seek plus the predicted average rotation. We also used these numbers to verify the other algorithms since any of the algorithms with queue length of 1 should equal FCFS.

The disks were spinning at 3600 RPM yielding a revolution time of 16.67 ms for 67 sectors of 512 bytes each, providing a transfer rate of 4K / 2.0 ms. A back-of-the-envelope calculation will show that for simple first come first serve scheduling policies, we can expect the average I/O time to be one half a rotation (8.3 ms) plus the time for an average seek (18 ms). This yields a disk utilization (the fraction of time the disks are actually transferring data) of $\frac{2}{2+8.3+18}$=7% As Figure 2 shows, this is very close to the hardware-derived and simulated utilizations.

---

[1]We chose a 4 KB block size as a common file system block size.

[2]We estimated the CPU time used in issuing an I/O from a user process by issuing two consecutive I/O's for two sectors on the same track. We found that after reading sector 0, the next sector we could read without missing an entire revolution was sector 12, 1/6 of a revolution later. This implied a 3 ms CPU turnaround time.

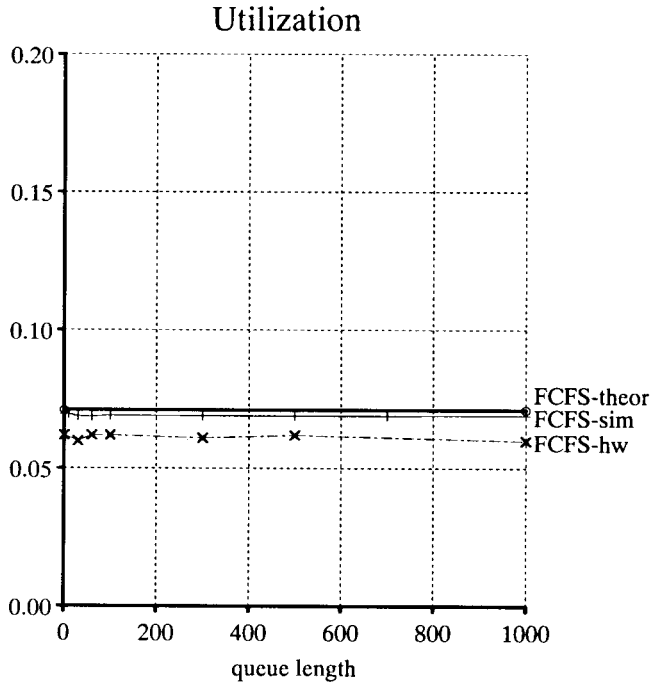

Figure 2: Comparing FCFS utilization derived from theoretical analysis (FCFS-theor), hardware measurements (FCFS-hw), and simulation (FCFS-sim).

4.2. Shortest Seek First

For this algorithm, we ignore the rotational latency and select requests based on the seek time required. On the average, we expect to see a half rotation (8.3 ms) and a seek to the closest cylinder with a request on it. This is a function of the total number of cylinders on the disk and the length of the queue. For example, the Eagles have 840 cylinders. With a queue length of 100, we expect the average seek to be 8.3 cylinders. Thus, for a queue length of 100, we expect average I/O time to be approximately 8.3 + 4.6 + .87 * sqrt(8.3) or 15.4 ms.[3] Figure 3 depicts these predicted values against the results actually obtained in simulation. The maximum queue length we used, 1000, corresponds to about 4 MB of dirty blocks. For example, consider a file server with 64 MB of main memory. It is reasonable to assume that 50% may be dedicated to a file cache, and of that, approximately 10-15% (3-5 MB) might be dirty.

4.3. SCAN and CSCAN

The SCAN scheduling algorithm is oriented towards producing fairer response time. It orders the requests by cylinder number and services all the requests for a given cylinder before moving the head to the next cylinder. When the head reaches one end of the disk, it merely reverses direction and begins scanning towards the other end of the disk. It is important to notice that this

[3]This is not quite accurate, as we have used the time of an average seek, which is not the same as the average time. It is a close approximation because of the almost linear seek profile (Figure 1).

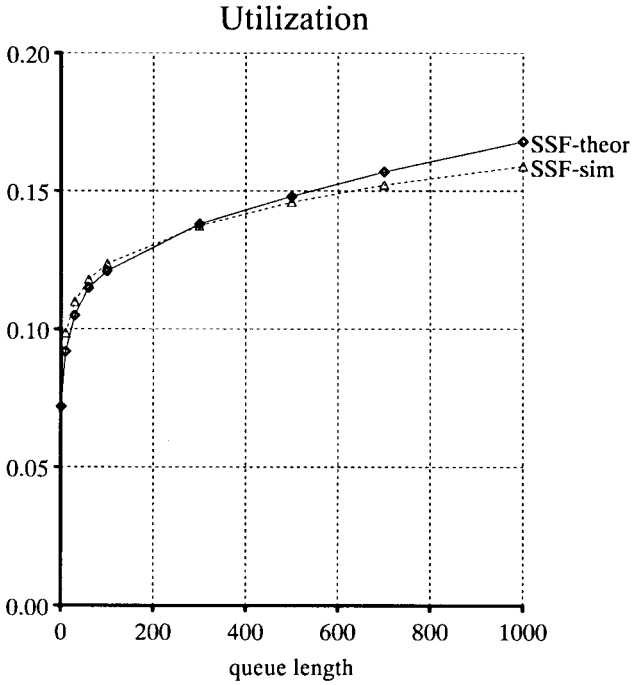

Figure 3: Comparing SSF utilization derived from theoretical analysis (SSF-theor) and our simulator (SSF-sim).

is actually very similar to the shortest seek first algorithm and we expect similar results.

One shortcoming of the SCAN algorithm is that requests on either end of the disks experience worse response time than those in the middle of the disk since those in the middle experience two passes of the head evenly spaced in time whereas the outermost cylinders delay for two full sweeps of all the cylinders before being revisited. Cyclical scan (CSCAN) alleviates this by paying one large seek at the end of the disk to move the head all the way to the other end. That is, the head always moves in one direction and we pay one very long seek at the end of each pass. This long seek is amortized over the requests, and the utilization is nearly the same as SCAN. The major difference is in the maximum observed response times and the variance of the response times. Figure 4 shows the utilizations for CSCAN which are essentially identical to SSF. However, as shown in Figure 5, CSCAN substantially improves the maximum observed response time.

5. Seek and Rotation Optimizing Algorithms

5.1. Shortest Time First

In SSF, we chose the request which yielded the fastest seek. In shortest time first (STF), we choose the request which yields the shortest I/O time, including both the seek time and the rotational latency. Advances in disk technology have reduced seek time more than rotational latency. As this trend continues, we expect rotational latency to account for a greater fraction of the total

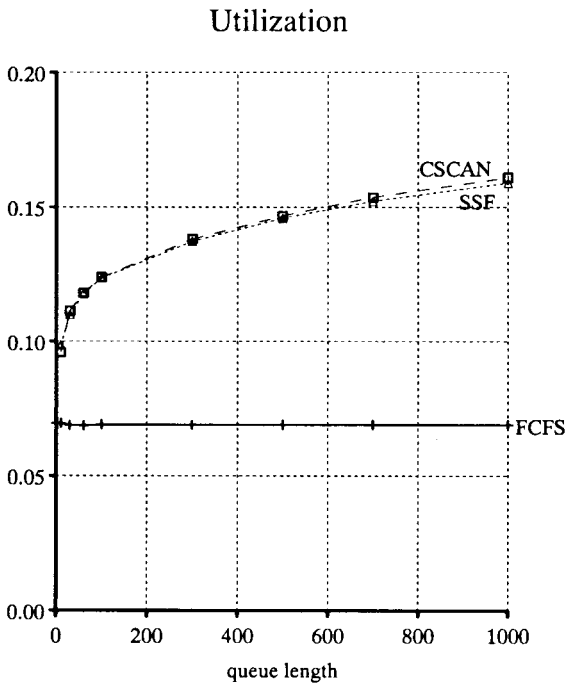

Figure 4: Disk Utilization from FCFS, SSF, and CSCAN. We graph the disk utilization derived from theoretical modeling, hardware verification, and simulation for three simple scheduling policies. The SCAN algorithm yields utilizations almost identical to CSCAN and is not shown.

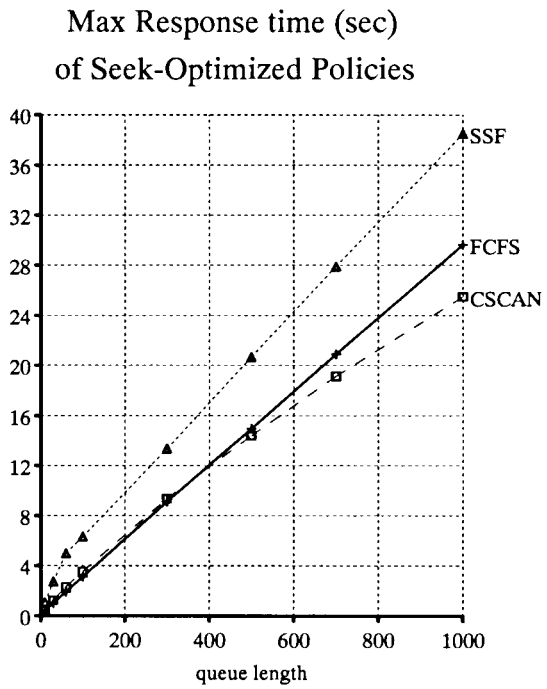

Figure 5: Maximum Observed Response Time for FCFS, SSF, and CSCAN. SSF has significantly worse maximum response time than FCFS, but CSCAN has roughly the same, or lower, maximum response time than FCFS. In some instances, CSCAN is able to have lower response time than FCFS because its average response time is lower.

I/O time, and rotation optimizing algorithms such as STF will become increasingly important. For example, saving half a rotation (8 ms) may cause an access 100 cylinders away to have a shorter total I/O time than an access 1 cylinder away.

STF is expected to yield the best throughput since we always select the fastest I/O. The algorithm scans the entire queue calculating how much time each request will take. It then selects that request with the shortest expected service time. For very long queue lengths (Q much greater than the number of cylinders), we expect to see the STF time approach 2.0 ms (the time to read a single 4K block). For very short queue lengths, we expect STF to approximate SSF since it is unlikely to have multiple requests on the same cylinder and adjacent requests are likely to be far enough apart so that seek time dominates rotation. Figure 6 shows the simulated results for STF. Note that, even at queue length of 1000, STF utilization is still rising. Preliminary runs at queue lengths of 5000 have utilizations of 40%.

Unfortunately, the scheduling algorithm is a function of both cylinder and rotational position, thus this algorithm is one of the most costly in terms of CPU utilization. In addition, STF

has the potential to starve requests, producing very bad response time (Figure 7). Note that, because the maximum observed response times in Figure 7 were empirically determined, maximum response times in a real system could be even worse.

The next two algorithms attempt to provide the utilization benefits offered by the shortest time first algorithm without paying a substantial penalty in response time.

5.1.1. Grouped Shortest Time First (GSTF)

In this algorithm, we combine scan techniques with shortest time first techniques. The disk is divided into some number of cylinder groups. Within each cylinder group, we apply a shortest time first algorithm, servicing requests within that group before advancing to the next group. This algorithm introduces two parameters, the queue length and the size of the cylinder group. Figure 8a shows the relationship between average I/O times as one holds the queue size at 1000

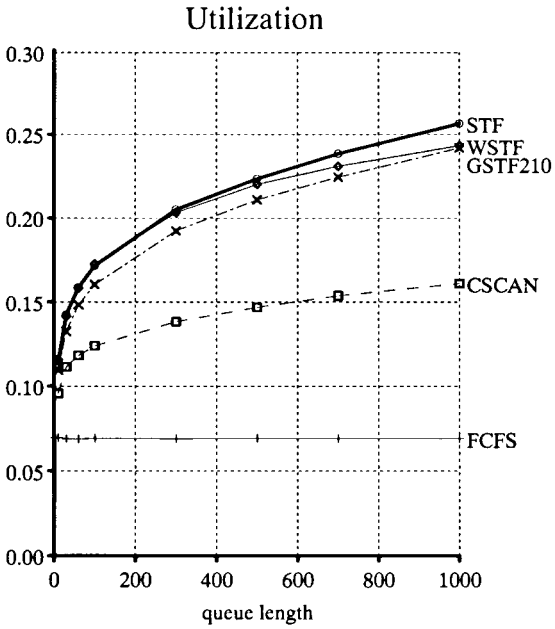

Figure 6: Disk Utilization for FCFS, CSCAN, STF, GSTF210, and WSTF. We graph the disk utilization for seek and rotation optimizing algorithms. FCFS and CSCAN are shown for comparison.

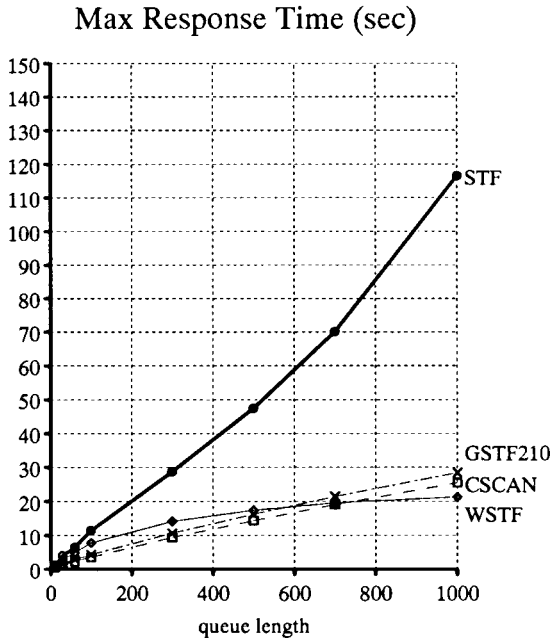

Figure 7: Maximum Response Time for CSCAN, STF, GSTF210, and WSTF. We graph the maximum response time for seek and rotation optimizing algorithms. Maximum response time for STF is much worse than other algorithms, but GSTF210 and WSTF bring maximum response time back down. Recall from Figure 5 that the maximum response time of FCFS is very close to that of CSCAN.

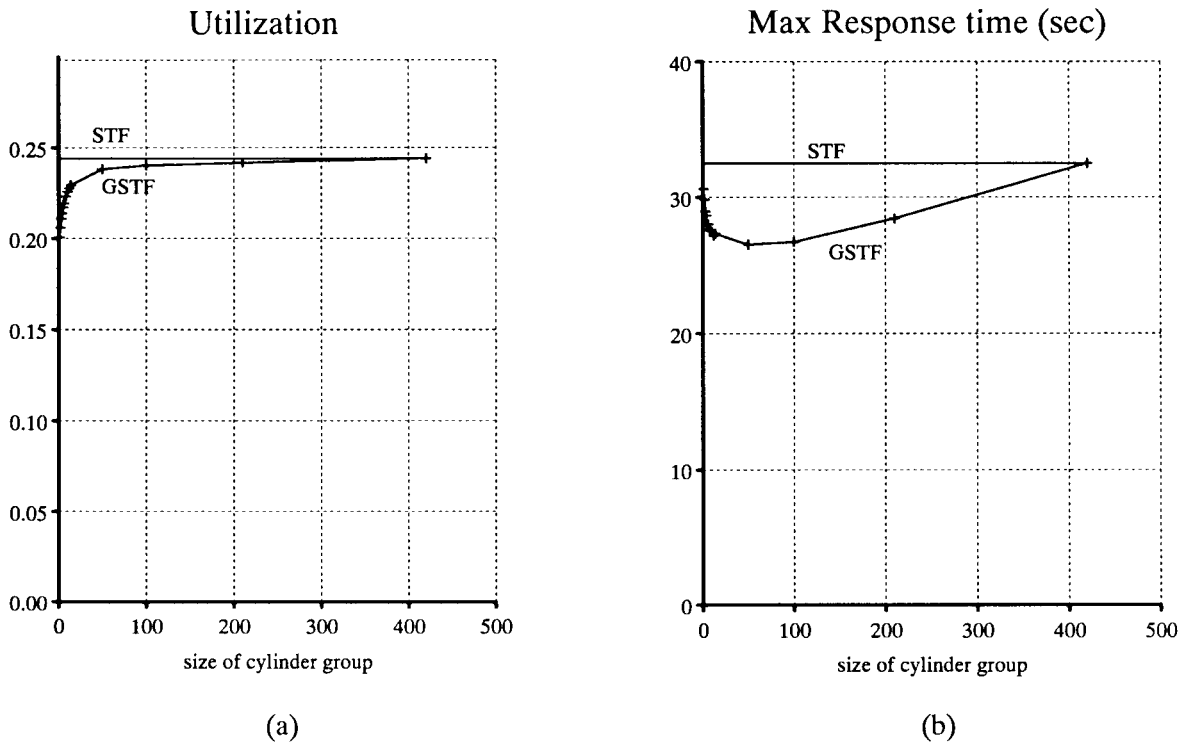

Figure 8: Utilization and Maximum Response Time of Shortest Seek First and Grouped Shortest Time First. Graphed above are the utilization and maximum response times as a function of the cylinder group size. The above results are for queue lengths of 1000.

and changes the cylinder group size. Figure 8b shows the maximum response times.

As the cylinder group size increases, the utilization of GSTF increases. Eventually, when the group size is 840 (the entire disk), GSTF is the same (by definition) as STF. Also, as the group size increases, the maximum response time becomes longer and longer, approaching STF. The early dip in the maximum response time curve is due to the interaction of utilization (average disk I/O time) and fairness. Although GSTF is more fair (less variation between response times) at small group sizes, the disk is being used less efficiently, and the average response time is larger. As utilization flattens out, the decreasing fairness causes increasing maximum response times. In Figures 6 and 7, we show the utilization and maximum response time for group size of 210 (4 cylinder groups per disk). We see that GSTF has utilization close to STF, but also succeeds in lowering the response time to close to the maximum response time of CSCAN.

GSTF services all requests for the current cylinder group before moving to the next cylinder group. If requests for the current cylinder group saturate the I/O system, it is possible to starve requests on other parts of the disk. A slight variation of GSTF freezes the queue of a cylinder group as soon as any requests to that cylinder group are serviced, guaranteeing that all the requests within that cylinder group are serviced before the head moves to the next cylinder group. Runs using this variation have 3%-4% lower disk utilizations and 15%-25% lower maximum

response times than the GSTF depicted in Figure 6.

5.1.2. Weighted Shortest Time First

This algorithm applies the standard shortest time first technique, but applies an aging function to the times computed. First, we assume a maximum acceptable delay between the time a write to the buffer cache is issued and when that data is written to disk (for these simulations, the time chosen was 30 seconds based on how frequently the UNIX kernel flushes its buffer cache) [MCKU84][4]. For each STF calculation, the actual I/O time is multiplied by a weighting value W. W is computed by calculating how much time is left before this request will exceed the maximum allowed response time. Thus, the weighted time is:

Let T_W *be the Weighted Time*
T_{real} *be the actual I/O Time*
M be the Max response time allowed
E be the elapsed time since this request arrived

$$T_W = T_{real} \frac{M-E}{M}$$

As the elapsed time increases, the weighting factor becomes smaller, the weighted time decreases, and the request is more likely to be serviced.

This algorithm displays remarkable performance. In most cases, the average I/O time is within 1-2% of the STF I/O, yet the maximum response time drops dramatically. Since WSTF has an enforced maximum response time, no I/O response time is allowed to take more than 30 seconds. In contrast, the STF "maximum" response time was empirically determined, and it did not guarantee that every request got serviced.

In trying to understand why WSTF performs so well, it is useful to observe that STF is a greedy algorithm. Always selecting the shortest time first means that regions of the disk get serviced first. However, as regions get cleaned off, there are fewer close requests to service. With WSTF, periodically, the arm is forced to do a "bad" seek, that is, one more costly than another. As a result, the head is in a new region providing the algorithm a better choice of requests from which to select. "Bad" seeks may also occur when a read or a forced write (i.e. a write that must go immediately to disk) is issued. Our results imply that these long seeks are unlikely to harm overall utilization.

6. Conclusion

There are two main conclusions from our work. First, substantial performance improvements (on the order of 3 to 4 times) can be gained by these scheduling mechanisms. As the queue from which one selects requests becomes larger, even more improvement can be realized. Second, there are algorithms which achieve this improved performance and still ensure fairness. However, note that at queue lengths of up to 1000, the best algorithms yield less than 40% disk utilization. This still leaves much room for improvement.

The implication is that greater utilization of disk bandwidth is achievable by viewing most of main memory as a large write buffer. In systems where the order of writes is unconstrained, one can take advantage of this unordered nature of writes to minimize the disk seek overhead. Therefore, larger file caches may be used not only to minimize I/O's but to make the necessary I/O's individually cheaper.

[4]One could also adjust the acceptable response time based on the number of requests in the queue (e.g. 150% * average I/O time of FCFS * queue length). Runs with these response time limits yielded results within a few percent of runs with the 30 second limit.

7. Acknowledgements

We would like to thank Brent Welch and Keith Bostic for instrumenting Sprite and BSD machines to collect statistics on real queue lengths. These numbers convinced us that the burstiness one expects on a file server does, in fact, result in a large number of disk writes to be scheduled.

8. References

[COFF72] Coffman, E. G., Klimko, L. A., and Ryan, B., "Analysis of Scanning Policies for Reducing Disk Seek Times", SIAM Journal of Computing, September 1972, Vol 1. No 3.

[FUJI84] M2361A Mini-Disk Drive Engineering Specifications, Fujitsu Limited, 1984.

[GEIS87] Geist, Robert, and Daniel, Stephen, "A Continuum of Disk Scheduling Algorithms", *ACM Transactions on Computer Systems*, February 1987, Vol 5. No. 1.

[GOTL77] Gotlieb, C. C. and MacEwen, H., "Performance of Movable-Head Disk Storage Devices", *Journal of the ACM*, October 1983, Vol 20. No. 4.

[HOFR80] Hofri, Micha, "Disk Scheduling: FCFS vs SSTF Revisited", *Communications of the ACM*, November 1980, Vol 23, No. 11.

[MCKU84] Marshall Kirk McKusick, William Joy, Sam Leffler, and R. S. Fabry, "A Fast File System for UNIX", *ACM Transactions on Computer Systems*, Vol. 2, No. 3, August 1984, pp. 181-197.

[MOOR75] G. Moore, ''Progress in Digital Integrated Electronics,'' *Proceedings IEEE Digital Integrated Electronic Device Meeting*, 1975, p. 11.

[ONEY75] Oney, Walter C., "Queuing Analysis of the Scan Policy for Moving-Head Disks", *Journal of the ACM*, July 1975, Vol 22. No. 3.

[SMIT81] Smith, A. J., "Input/Output Optimization and Disk Architectures: A Survey", *Performance and Evaluation I* (1981), pp. 104-117.

[TEOR72] Teorey, Toby J. and Pinkerton, Tad B., "A Comparative Analysis of Disk Scheduling Policies," *Communications of the ACM*, March 1972, Vol 15. No. 3.

[WILH76] Wilhelm, Neil C., "An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling Using an Empirical Model for Disk Accesses", *Communications of the ACM*, January 1976, Volume 9, No. 1.

Margo I. Seltzer
UC Berkeley

Peter M. Chen
UC Berkeley

Margo I. Seltzer is a Ph.D. student in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. Her research interests include file systems, databases, and transaction processing systems. She spent several years working at startup companies designing and implementing file systems for transaction processing and designing a full custom VLSI CPU for a multiprocessor architecture. Ms. Seltzer received her AB in Applied Mathematics from Harvard/Radcliffe College in 1983.

Peter M. Chen graduated from the Penn State University (go Lions!) in 1987 with a BS in Electrical Engineering. He received his MS in Computer Science from the University of California at Berkeley in 1989 and is currently a Ph.D. student working with the RAID (Redundant Arrays of Inexpensive Disks) project. His interests are in I/O system design and performance evaluation.

John Ousterhout
UC Berkeley

John K. Ousterhout is a Professor in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. His interests include operating systems, distributed systems, user interfaces, and computer-aided design. He and his students have developed several widely-used programs for computer-aided design, including Magic, Caesar, and Crystal. Ousterhout is now leading the development of Sprite, a network operating system for high-performance workstations. Ousterhout is a recipient of the ACM Grace Murray Hopper Award, the National Science Foundation Presidential Young Investigator Award, the National Academy of Sciences Award for Initiatives in Research, the IEEE Browder J. Thompson Award, and the UCB Distinguished Teaching Award. He received a B.S. degree in Physics from Yale University in 1975 and a Ph.D. in Computer Science from Carnegie Mellon University in 1980.