Implementation of a Rotation-Latency-Sensitive Disk Scheduler

Lan Huang Tzi-cker Chiueh
Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
{lanhuang,chiueh}@cs.sunysb.edu

May 22, 2000

Abstract

Processor speed and memory capacity are increasing several times faster than disk speed. Disk I/O performance becomes an important bottleneck due to this disparity. Disk head seeking overhead has been reduced faster than rotational overhead and rotational overhead becomes a significant part in the overall overhead for a disk I/O operation. Traditional disk scheduling methods are targeted at reducing seeking overhead due to two reasons: device driver is lack of disk head information or rotational overhead is not a significant part of the overall overhead in early years. Although scheduling algorithms with rotation delay in concern are more promising for modern disks, not a single rotation-sensitive algorithm has ever been implemented due to lack of the knowledge of hard drive physical parameters and unawareness of disk head position.

In this paper, we present a pure software based disk head prediction mechanism which can be used to predict the current disk head position. We also describe certain heuristics to estimate the disk service time for a request. Based on these mechanisms, we implemented the first (to the best of our knowledge) rotation-sensitive disk scheduling algorithm on real disk storage systems. We show the stability and accuracy of our head prediction mechanism in this paper. Our rotation-sensitive scheduler is able to provide real-time guarantee to time-critical applications because of its full knowledge about disk head and service time. We also present the performance results for various rotation-sensitive disk scheduling algorithms against real and synthetic traces.

1 Introduction

Processor speed and memory capacity are increasing by over 40% per year. In contrast, disk speed is growing by only 7% per year [14]. Disk subsystem must be carefully managed to compensate the growing performance disparity between processing and storage components. In the past 25 years, various disk scheduling algorithms have been researched to improve the disk bandwidth utilization, to minimize the response time and to reduce the starvation. Among the most well-known disk scheduling algorithms are seek time optimized ones: SSF(shortest seek time first), SCAN, C-SCAN(cyclical SCAN) and rotation sensitive ones: STF(shortest service time first), GSTF(grouped STF), WSTF(weighted STF) [12].

SCAN restricts its search for minimum seek time request to one direction. It improves the fairness problem in SSF. When the disk head reaches one end of the disk, it merely reverses the direction and begins scanning towards the other end of the disk. One shortcoming of the SCAN algorithm is that requests on either end of the disks expereince worse response time than those in the middle of the disk. CSCAN alleviates this by paying one seek at the end of the disk to move the disk head to the other end. The disk head is thus always moving in one direction and the maximum observed response time is improved compared to SCAN, SSF.

STF searches for the request with the minimum disk service time, including seek time, rotation time and media access time. GSTF algorithm groups the requests into groups. It proceeds to the next group only when the requests in the current serviced group have been serviced. GSTF has better maximum response time than STF. WSTF is similar to STF except that in WSTF, the predicted disk service takes in the effect of the waiting time of the request.

Advances in disk technology have reduced seek time more than rotational latency. As this trend continues, we expect rotation-sensitive disk scheduling algorithms to be more useful.

However, disk remains a black box to disk driver in the sense that many physical disk parameters are unknown to disk drivers, such as disk head position, logical-to-physical mapping, on-board cache properties, seek curves. Most of the previous disk scheduling research can only demonstrate the performance through disk simulation. For example, most widely used disk scheduling algorithms in Linux disk driver is FCFS (first come first serve) algorithm. To the best of our knowledge, none of the existing disk controller exposes current disk head position to the device driver. Without disk head information, it is infeasible to build a rotation-sensitive disk scheduler in practice.

To accomplish the goal of comprehensive disk scheduling, we introduce a simple pure software-based disk head prediction scheme and demonstrate its validity and accuracy in this paper. We adjust the disk access time model to improve the accuracy of access time estimation in the context of a real disk. As illustrated in paper [16], disk physical parameters can be extracted once for all by using micro-benchmark and interrogative disk commands. With these extracted physical parameters, we implemented rotation-sensitive disk scheduling algorithms on real disks. Full knowledge about the servicing time of disk requests provides possibility of real-time scheduling for time-critical applications. We also show performance results for various rotation-sensitive scheduling algorithms against synthetic and real system traces.

The rest of the paper is organized as follows. In Section 2, we give a brief background on disk physical parameters extraction and existing disk scheduling algorithms. Section 3 shows how we do the software-based disk head prediction as well as the accuracy and stability of this scheme. We describe the heuristics we developed for estimating disk access time in Section 4. Section 5 presents various rotation-sensitive scheduling algorithms we implemented. We show the performance results in Section 6 and Section 7 concludes the paper.

2 Related Work

There are two main streams of disk scheduling algorithms: seek-time based and rotation-sensitive ones. FCFS has been illustrated as sub-optimal in many disk scheduling literature. In early years of the history of movable head disk, full-stroke seek time can be as long as 135 ms for IBM2314 while the full rotational delay is only one-eighth of that [5]. A wide range of seek time optimized algorithms have been studied. Hofri [8] shows the main drawback to SSF(shortest seek time first) is the larger variance in I/O response time. The SCAN algorithm restricts its search for the minimum seek time request to one radial direction. SCAN causes long waiting times for requests on the extreme of the disks. FSCAN addresses this by "freezing" the queue once the scan starts. Coffman, Klimko, and Ryan had an extensive analysis of the performance of FCFS, SCAN, SSTF in [3, 2]. Geist and Daniel have described a continuum of algorithms from SSF to SCAN differing only in the importance of maintaining current scanning direction [6].

As CPU power and memory capacity of computer systems increases dramatically, rotational delay is reduced slower than seeking overhead, rotation-sensitive disk scheduling will give better performance. Seltzer [12] evaluated the performance of rotation-sensitive scheduling algorithm and seek optimizing algorithms and found that rotation based algorithms performs better than seeking based ones(40% more). To reduce starvation, variations of rotation-sensitive scheduling schemes have also been pursued. Similar discussion about rotation-sensitive scheduling algorithms can be found at [9] by Jacobson and Wilkes. Bender, Andrews and Zhang [11] presents HeadSchedule algorithm which services all the requests in at most $\frac{3}{2}T_{opt} + a$ rotations, where T_{opt} is the number of rotations taken by an optimal algorithm. Disk scheduling problem is NP-hard, thus it is not practical to look for the optimum algorithm. They also give a CHAIN algorithm which potentially can be implemented as an online algorithm in disk scheduler. CHAIN tries to find the longest sequence of requests the disk head can service within one rotation from current disk head position.

However, due to the lack of detailed hard drive physical parameters, these rotation-based scheduling algorithms are evaluated only in theoretical analyses or simulation. In this paper, we present the software-based disk head prediction and disk access time prediction mechanisms, which is believed to be also extremely useful in write-where-diskhead-is type of file system proposed in [13, 4, 7, 1]. As known as eager-writing, to write near the disk head position in their cases requires the knowledge of disk head position at any time.

Ganger and Wilkes illustrated that disk physical parameters can be extracted once for all by using micro-benchmark and interrogative disk commands in [16]. Physical geometry data is permanent and won't be changed after the disk is manufactured. Rotation speed, seek time might be affected by the thermal variation and other conditions, but in a fixed period of time, they are stable. All these make it possible for us to develop software-based mechanisms to estimate the behavior of disk head.

3 Disk Head Prediction

Disk head position is critical information in many write-near-to-disk-head cases. Chiueh designed a track-based based logging disk which writes log record to the predicted nearest position to the disk head on the log disk [1]. R. Wang [13] presents a virtual log based programmable disk with the presumption that eager writing is available. The file system passes the target logical address to the low level driver, and the driver maps the received logical address to a nearest physical disk address, thus shortest disk service overhead is achieved.

To the best of our knowledge, none of the existing disk controller exposes this information to users. So a software-based prediction mechanism is needed. Disk physical parameters can be extracted once for all by using micro-benchmark and interrogative disk commands [16]. Rotation speed can be determined by performing a sequence of *writes* to the same location. The mean time between request completions is the average rotation period. According to our experience, rotation speed is pretty stable under a fixed temperature. By using interrogative disk commands (MODE SENSE, TRANSLATE ADDRESS etc.), one can retrieve the mapping between logical address and physical address. In some hard drives, these interrogative commands are not implemented and the following microbenchmark(GeoExtraction) is needed to extract out the data sector number per track[1]. The test is as shown in 1.

```
MAXLBA is the total sectors the tested drive has;
MINDISTANCE is a constant;
for(i = 0; i < MINDISTANCE ; i++){
    for(dest = i; dest < MAXLBA ; dest += MINDISTANCE)
        perform a *write* to address *dest*
}
```

Figure 1: *GeoExtraction* Algorithm. GeoExtraction Algorithm perform several(MINDISTANCE) sets of sequential write to the tested drive. MINDISTANCE is the number of sectors the hard drive passes by during the command processing period. To write MINDISTANCE away guarantees that the overhead for each write is minimal instead of containing the full rotation overhead. When the destination sector is crossing track boundary or cylinder boundary, we expects to see longer delay due to head switch or cylinder switch overhead. These extra overhead reflects as regular jumping pattern in the extracted overhead curve and we can tell the data sectors per track accurately.

GeoExtraction algorithm is tested on ST41601N and part of the curve is shown in Figure 2. MINDISTANCE is 10 in this case. The very first point of the jump is the starting point of the next track or cylinder. And this curve proves that 6 sectors at the end of this cylinder is spared for defect management and except the last track of the cylinder, the SPT number for other tracks is 85.

In order to estimate the disk head position, the driver takes a timestamp after every disk access in the interrupt handler code, and stores the timestamp in the variable T_0 and the logical block address to which the disk head moves in the variable LBA_0. For those cache hit reads, the timestamp and address is not updated since disk head does not move to the destination when disk on-board cache is able to service it.

With disk physical geometry information, the logical block address LBA_0 can be converted to CHS(cylinder, head, sector) address C_0, H_0, S_0. A disk driver can predicts the disk head position(LBA_1 or C_1, H_1, S_1) at time T_1 by using the following formula:

$$S_1 = (\frac{(T_1 - T_0)\ mod\ RotateTime}{RotateTime} * SPT + S_0 + \delta)\ mod\ SPT$$

$$H_1 = H_0, C_1 = C_0$$

$$LBA_1 = LBA_0 - S_0 + S_1$$

where SPT is the number of sectors in the current track, and *RotateTime* is the disk rotation period. δ is an empirical value to compensate the command processing overhead, some unknown fixed mechanical overhead [16] and other side effect from embedded servo system. Its value can be obtained by performing a series of single sector predicted write operations to the drive with different δ. Those set of tests with measured overhead for write operation less than half of the full rotation time are considered as predicted correctly. Among all δ values used in these correct tests, we choose the minimum one as the empirical value to be used in the prediction formula. If the value used is smaller than the extracted minimum empirical value, most of the prediction will be wrong

[1]Note that even in the same zone, the number of data sectors in differenent track may be different since usually in the last of one cylinder, several sectors are spared for defect management.

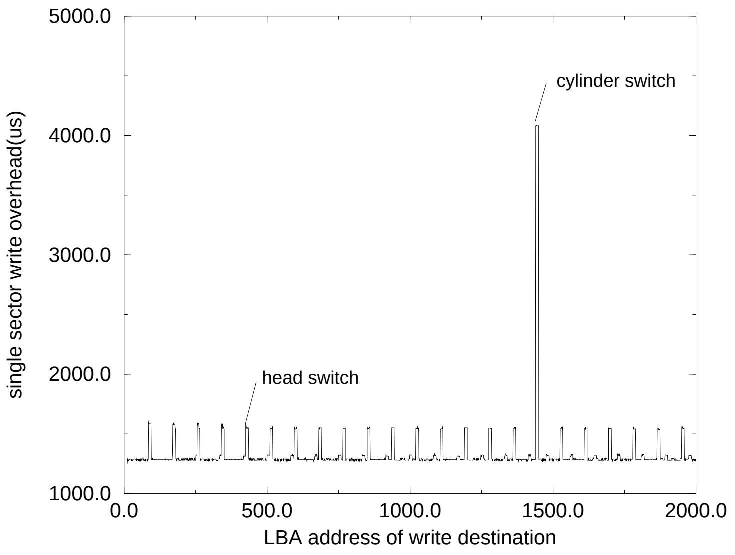


Figure 2: The resulting curve of applying GeoExtraction to ST41601N.

and a full rotation overhead can be seen in the measured overhead of write operation. But if the prediction module uses a little larger value, the prediction accuracy won't be affected much. For the dedicated servo system, one side of the platter is used to hold the servo information, thus no extra compensation is needed in the above formula to cover the servo sectors. The prediction module itself does not involve intensive computation, less than one microsecond is needed to get a timestamp and compute the prediction formula. Accuracy of $RotateTime$ is critical to the correctness of the head prediction. $RotateTime$ is a pretty stable value as we observed and we might only need to re-measure this value only weekly or even longer.

One recent invention in modern disks is called *wear leveling*, The idea is to remove the amount of time that the heads spend spinning over the same physical area of the disk. Every fifteen seconds, the drive controller moves the heads by one track, traversing the entire disk area, so that no area of the disk has the heads sitting over it for very long. The built in servo control mechanism automatically adjusts for shifts in the position of the media due to temperature variations. Many drives have implemented an additional thermal adjustment called thermal re-calibration. Every few minutes, the heads are moved and the distance between tracks measured. This information is recorded and used to aid in positioning the heads when reading or writing needs to be done. These periodic behavior potentially affects the accuracy of our disk head prediction. But as long as the reference point LBA_0, T_0 are updated after each disk access, given a normal access pattern to the disk, most of the prediction will be good ones.

To verify the accuracy of our prediction scheme, we performed a series of *writes* to the predicted address and measure the physical elapsed time from when the command is sent to the driver until the command is done. If the prediction is inaccurate, we expect to see that the average physical elapsed time has much longer delay than media access time for this write operation. Otherwise, we expect to see that most of the physical elapsed time to be media access time, which is proportional to the number of sectors read from or written into the disk. The tests on a ST41601N SCSI disk confirm that our prediction formula works accurately.

Due to periodic thermal re-calibration, wear leveling as well as minor fluctuation of rotation speed, our disk head prediction needs periodical re-calibration(update of reference point) to guarantee the accuracy. This re-calibration goes through when the disk is idle, otherwise it is unnecessary to do the re-calibration of reference point since it is updated anyway by the current disk operation. The following micro-benchmark is able to tell how long the disk head prediction can sustain before it needs a re-calibration.

```
for(i = 0; i < measurements ; i++){
    perform a write to address $t_0$
    wait for i * unit_interval time
    $t_1$ = HeadPredict();
    perform a write to address $t_1$
}
```

Figure 3: *Stability* Algorithm. The basic algorithm performs disk write to predicted position a named time interval (i**unit interval*) from the previous disk access. The **raw** disk is used to avoid file system optimization. *Flush* bit is turned on in SCSI command to avoid on-board cache effect. *HeadPredict* is our module for disk position prediction which returns the predicted value.

We apply *Stability* Algorithm to ST41601N disk and get the stability curve in Figure 4. The duration of correct prediction without re-calibration is 14.4 seconds. Any result for sustaining period longer than 14.4 seconds is affected by periodic behavior of hard drive. Is this value good enough? As long as the re-calibration period of *HeadPredict* module is longer than the average inter-arrival time, most of the disk head prediction will be accurate. As pointed out in [15, 10], disk access usually falls into bursty pattern. In bursty period, the reference point is updated frequently. In sparse period, re-calibration of *HeadPredict* updates the reference point timely before head prediction goes awry. So we conclude *HeadPredict* module works well in both cases.

HeadPredict module needs full knowledge of the properties of on-board disk cache so that it can skip the cache hit ones while updating the reference point in the memory. Most of the properties can be retrieved from Mode Page. And it can be verified by performing a series of read/write operations to certain specific position. Our measurements shows that ST41601N does track-based caching and the cache size is 192 Kbytes. More cache property data for ST41601N can be found at the appendix of [16].

4 Estimation of Disk Access Time

One criteria in rotation-sensitive disk scheduler is disk access time, i.e. given the current disk position, how long does it take to move the disk head to the destination? The service flow of a disk read requests contains: CPU prepares the command and data, CPU sends the command to the disk, disk actuator seeks to the destination cylinder, disk control system switches on the right head, disk head waits until platter rotates to the destination sector, disk head accesses the media, data are transfered to the kernel buffer.

All these steps can be summarized in the following service/access time estimation formulas, T_a is the access time, T_s is the service time, size is the request size in sectors, T_{seek} is the seek time,

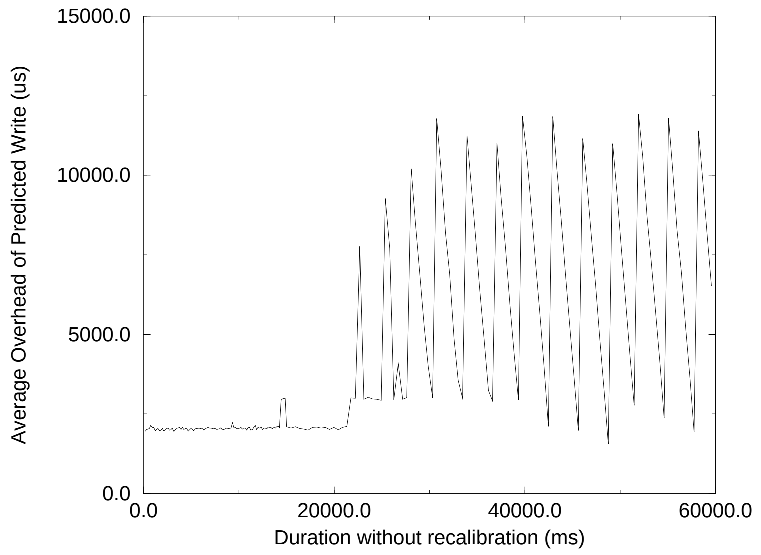


Figure 4: Stability of disk head prediction for ST41601N. This drive has a dedicated servo system. More physical parameters can be found in [16].

$T_{headswith}$ is the head switch overhead, T_{rotate} is the extra rotation time needed for positioning the disk head to the destination sector:

$$T_s = T_a + RotateTime * size/SPT$$

$$T_a = T_{seek} + T_{headswitch} + T_{rotate}$$

where $T_{headswitch}$ can be omitted in disks with dedicated servo system since their head switching is very fast and can be overlapped by T_{seek} time if the distance is not zero. T_{seek} can be estimated by a widely used disk seek time model [9, 12]. For short seeks, the disk arm accelerates to a halfway point, then decelerates and settles on the destination cylinder. For long seeks, the disk control system accelerates the disk arm to its full speed, then decelerates it back to zero at the desired cylinder.

$$T_{seek} = \begin{cases} c_1 + c_2 * \sqrt{s} & \text{if } s \leq threshhold \\ c_3 + c_4 * s & \text{if } s > threshhold \end{cases}$$

The seek time parameters for our tests on ST41601N can be found at the appendix.

T_{rotate} is decided by the seek time T_s and *physical* sector distance between source and destination. When the disk head arrives at the destination track, if the number of sectors passing by during the seeking period is larger than destination sector(S_1), the disk head has to wait for almost one full rotation period until the destination sector comes back. Due to the track skew and cylinder skew, the sectors on each track with minimal LBA address are not aligned. S_{rotate} is defined as the number of sectors to pass by after arrival at the destination track before meeting the destination sector. Skew effect must be added into formula for S_{rotate} too.

$$S_{rotate} = (S_1 + \sum Skew_i - T_s * SPT/RotateTime - S_0) \ mod \ SPT$$

However, we don't have accurate skew factors for our experimental disk. In order to predict the disk access time accurately, we perform extensive micro-benchmark over the disk and extract out the approximation curves to help us decide the access time. The following MinOffset Algorithm summarizes the microbenmark which extracts out, on arrival at destination track, how many sectors(S_{offset}) the disk head is away from the starting point(C_0, H_0, S_0)[2].

```
for(measurements = 0; measurements <SPT ; measurements++) {
    write to sector $s_1$ on track $c_1$
    write to sector ($s_1 + measurements$) on track $c_2$
}
    return $measurements$ with the minimal overhead for the second write;
```

Figure 5: MinOffset Algorithm. This algorithm finds the offset of the arrival point at destination track from the starting point. This offset reflects the combined effect of track/cylinder skew and seek time.

MinOffset algorithm is applied within each zone for every cylinders. And T_{rotate} can be computed from S_{offset} in the following way:

$$T_{rotate} = S_{rotate} * unittime$$

$$S_{rotate} = (S_1 - (S_{offset} + S_0) + SPT) \ mod \ SPT$$

where unit time is the time for the disk head to pass one sector on the current track. In certain boundary cases, the arrival point is close to the destination point and some deviation exists in S_{offset} estimation, which will make S_{rotate} deviates from real value far away (with a difference close to SPT). To be safe with this possibility, our prediction module chooses the maximum value between $((S_{rotate} - CMPS + SPT) \ mod \ SPT)$ and S_{rotate} as the estimated value for S_{rotate}. $CMPS$ is a predefined compensation value. Typical value is 5.

Figure 6 gives a graphical view of what the above formula represents.

This alternative saves us from lack of the accurate track/cylinder skew data. This approach requires exhaustively performing MinOffset over the whole disk. But since this is once for all, or at least for each disk format, so it is an affordable approach with accurate results. These data can be stored in a table for disk scheduler to look up when needed. To improve the space efficiency, we fit the data set with same starting cylinder into multiple linear curves. Only the slope and starting value of the curves are stored into a main memory table. Adjacent cylinder's curve feature can be merged to further reduce the memory space needed. When the disk scheduler needs to predict the access time for the given source, destination address, it has to compute the S_{offset} first from curve slopes and starting values we stored in the look up table, then applies the formulas above to get the S_{rotate}, T_a. Fortunately, the (slope, starting value) pair remains roughly the same within each zone. Experiments show that we can get as good accuracy using single slope per zone as using the comprehensive slope, starting value table. It also confirms that one only needs to perform MinOffset algorithm on small local areas in each zone to extract out the (slope, starting value) pair.

[2]In the rest of the paper, we use (C_0, H_0, S_0) to represent starting point position, and (C_1, H_1, S_1) as destination point

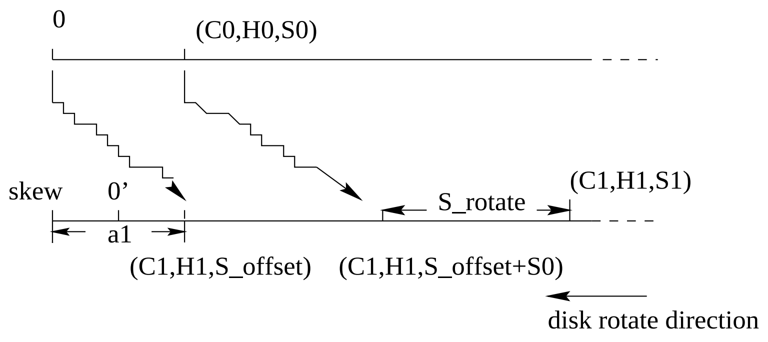


Figure 6: When disk head moves to the destination track, the disk platter passes by a_1 sectors during the seek period, plus the effect of track/cylinder skew, disk head will meet the sector with $S_{offset}= a_1+skew$ away from starting point. The disk head has to wait the remaining S_{rotate} sectors passing by before it can access the media.

5 Disk Scheduling Algorithms

A good scheduling algorithm must provide good performance in terms of response time as well as fairness to avoid starvation. In real time applications, real-time guarantee from storage system is necessary.

5.1 Seek Optimized Algorithms

Traditional seek optimized algorithms include SCAN, CSCAN etc. In CSCAN, the disk scheduler gives higher service priority to the request with minimal seek time in the current disk head moving direction. Also, in CSCAN, the scheduling decision can be made only based on the cylinder number of each candidate requests and the cylinder position of the current disk head. Potentially, the disk scheduler does not know exactly how long the next request takes although it decides it is the potentially shortest one. Indefiniteness comes from the extra rotation time for which the disk head has to wait until the destination sector meets the disk head. Without accurate disk head position(C, H, S) and an accurate access time prediction, CSCAN won't be able to serve the real-time applications.

5.2 Rotation-sensitive Scheduling Algorithms

One motivation for rotation-sensitive disk scheduler is the trend of modern disks. Seek time is reduced much faster than rotation time. Rotation sensitive disk scheduler could be more effective in servicing the requests than seek time based scheduler. One can argue that nowadays hardware is so fast that software does not has to waste time on complex scheduling. However, real-time application necessitates this work. A rotation-sensitive disk scheduler has the full knowledge about the current disk head position and the predicted servicing time of the next request. It can integrate real-time restrictions together with scheduling criteria and achieve real timing and high performance.

The basic version of rotation-sensitive scheduling algorithm is STF (shortest time first). In STF, the request which yields the shortest I/O time (including rotation time and seek time) is chosen as the next request to service. STF is expected to give the best throughput but the fairness is compromised sometime due to its always greedy algorithm. Another algorithm GSTF try to serve a group of request first before move the disk head to the other area of the disk. Within each group, STF algorithm is applied. GSTF is to give fast throughput as well as fairness. Similar to aging process scheduling in FreeBSD, WSTF applies aging to the access time computed for each candidate request in the queue. Each estimated physical I/O time is multiplied by a aging factor W(W < 1), the longer the request waits in the queue, the smaller W is and more chance it gets to be serviced [12]. And the formula to compute weighted disk access time is as following:

Let T_w be the weighted time
T_{real} be the actual I/O time
M be the maximum response time allowed (empirical value)
E be the waiting time in the queue up to now

$$T_w = T_{real} * \frac{M-E}{M}$$

6 Prototype Implementation and Performance Evaluation

6.1 Prototype Implementation

The prototype of our rotation-sensitive scheduler is implemented in Linux-2.2.5 kernel, on a SCSI general interface. Figure 7 shows the architecture of our prototype. The rotation sensitive scheduler searches for the next request to service and communicates with the general SCSI interface. The rotation sensitive scheduler contains the disk head prediction module and the disk service time prediction module.

6.2 Performance Evaluation

We test our rotation-sensitive scheduler against synthetic workload and real disk I/O traces. Synthetic workload has a read write rate of 1:3, which is the same as the number reported in [15]. The disk queue is always full except when the test is finishing off and no more than length requests are left in the queue. This model is first used in [12].

The real traces are collected in the PC boxes at Experimental Computer System Lab, SUNY Stony Brook. The machine we tested has single Pentium-II CPU, 323 Mega bytes DRAM memory, one ST32122A disk as data disk, one WDC AC29100D drive as log disk for trace data. The logging data are put into this dedicated disk so that disk I/O for these data do not affect the data disk accesses of the test system. The system is running Linux-2.2.5(kernel version). This machine is shared by several persons in the lab. We collected the traces on this machine for more than a month. Due to the regular schedule of the persons who use this machine, the daily disk traces are distinguished clearly as heavy-load traces and light-load traces for different days. During the busy period, people run tests/compilation on it. The light load traces are mainly contributed by web browsing, email editing, file system periodic writing back etc. Queue length determines how far the disk scheduler can look ahead to search the next request to service. Figure 8 shows the queue length distribution for two typical days traces we picked from this trace data set. In light load period, more than 50% requests arrive at an idle disk. But sometimes queue length can be very large as in heavy load case. The measured read write rate is 1:2 for both cases. The light load day has 129261 disk I/O totally and the heavy-load one contains 1176491 disk I/O requests.

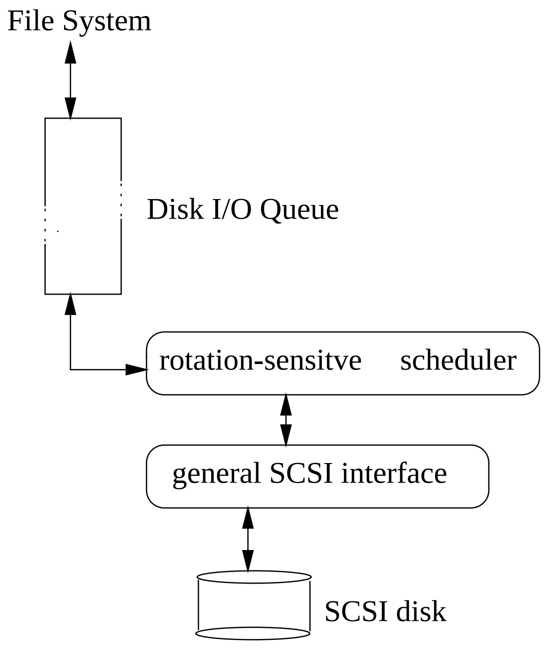


Figure 7: Prototype of Rotation-Sensitive Disk Scheduler.

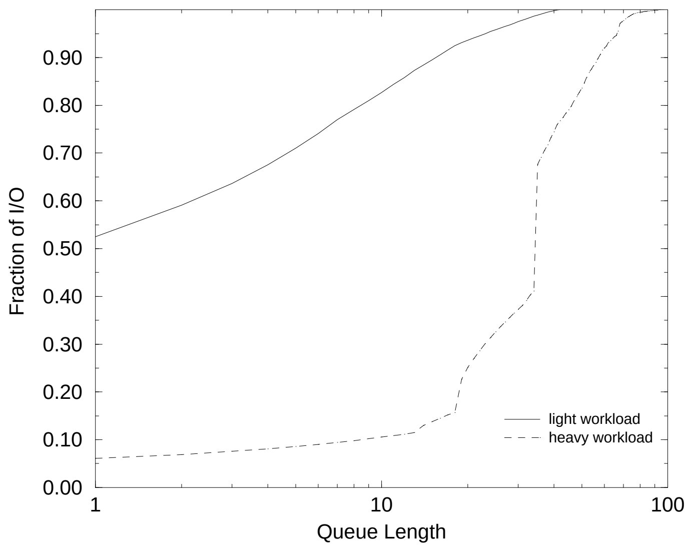
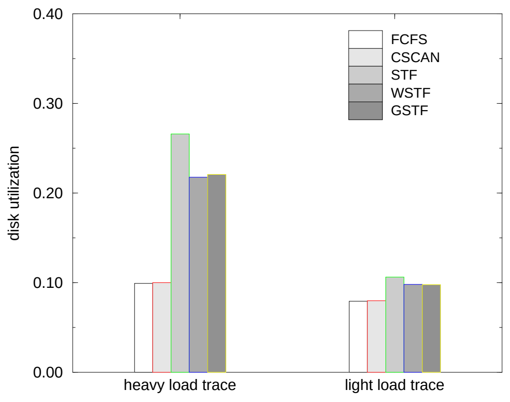


Figure 8: Queue length distribution for two days workload. Mean queue length is 5 for light load case and 33 for heavy load case.

The scheduling performance can serve as a verification for the correctness of the disk head prediction and other heuristics we introduced. One could also memorize the expected access time and real access time on the run and compare them afterwards. The misprediction rate is below 10% as measured this way. The trace-driven tests are performed on ST41601N SCSI disk, and the host machine has a PII 300MHz single CPU, 128M memory. The disk scheduler keeps on-board cache in concern when making the scheduling decision. The read/write size to disk is 4 Kbytes (i.e. 8 sectors), which is the file system block size.

Against real traces, we run WSTF, GSTF, STF, FCFS and CSCAN algorithms. Disk bandwidth utilization determines the fraction of time the disks are actually transferring data. For a disk spinning at 5400 RPM yields rotation time of 11.11 ms for 75 sectors of 512 byte each, providing a transfer rate of 4K / 1.12ms. The disk utilization for a 8 sectors write operation which takes 10 ms is $\frac{1.12}{10} = 11.2\%$. The maximum response time reflects the degree of request starvation. Figure 9 shows the result on real I/O traces. The rotation sensitive disk scheduler brings out larger performance difference over CSCAN and FCFS in the heavy load case than in light load case. The reason is in light load case, average queue length(5) is much smaller than in heavy load case(33). GSTF, WSTF reduce the maximum response time effectively.

Figure 10 shows the result for synthetic traces. Each run contains 100 times the number of queued items, which proved to be able to get a stable result (1-2% variance). GSTF and WSTF can efficiently solve the starvation of STF as shown in the figure, without compromising much of the performance. For both synthetic trace and real trace, the rotation sensitive scheduler has a 250% improvement in disk utilization over CSCAN and FCFS.



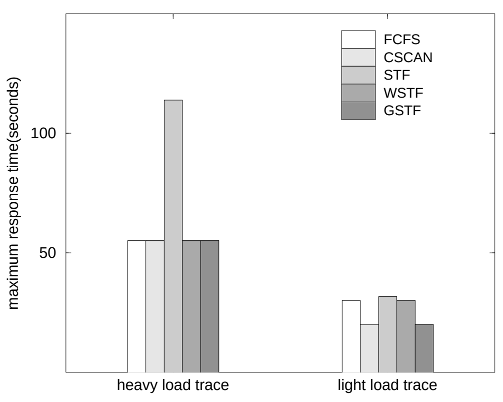


Figure 9: Performance difference in heavy load case is larger than in light load case due to smaller queue length in light load case. GSTF, WSTF reduces the starvation effectively. In heavy load case, disk utilization of STF class of schedulers has over 200% disk utilization improvement.

Note that in above test, the requests sent to disk are in the unit of 4K byte. If larger size is used(e.g. 256K), then most of the requests cover several consecutive tracks, and the distance between requests is far, so that there is no much space left for the rotation sensitive scheduler to improve over seek time based ones. Experiments proves these thoughts too.

The computation cost for complicated disk scheduling algorithms we discussed here is not an issue for medium queue length. Given a 300MHz CPU, the average overhead to predict one access

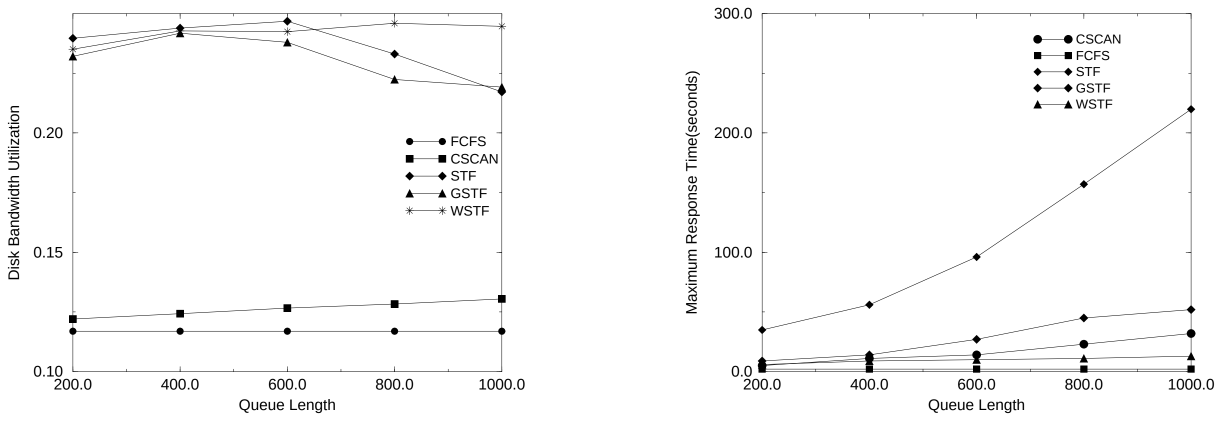


Figure 10: WSTF, GSTF, STF has over 200% improvement in disk utilization. STF is the most greedy algorithm and thus gives the worst starvation. WSTF, GSTF brings maximum response time down. In STF class of schedulers, disk utilization increases with the queue length to a point and then decreases due to the extra overhead it has to pay for longer queue analysis. Maximum response time increases with the queue length since more waiting are necessary in longer queue. GSTF uses the group size 700, which is the optimal value as measured.

time in our scheduler is around 8 microseconds.

7 Conclusion

In this paper, we describe the implementation of a rotation-sensitive disk scheduler. Disks remained a black box to the upper level modules such as device driver and file systems. Our disk head prediction mechanism and disk access time model could bring full knowledge of the disks to the upper level modules, which makes more advanced optimization and application possible, e.g. complex disk scheduler as illustrated in this paper and real-time storage system, which we did not discuss in this paper. And we use disk scheduling as one case study to prove the usage of these approaches. The future work of this project includes extending this technique to real-time file systems. We are working on building more general tools to extract detailed disk parameters like track skew, cylinder skew, defect sector position as well. Finally, this technique potentially can be embedded into a piece of customized IC chip on disk board, which will make disk more transparent as well as informative to programmers.

References

[1] T. Chiueh. Trail: a track-based logging disk architecture for zero-overhead writes. In *Proceedings of 1993 IEEE International Conference on Computer Design ICCD'93*, pages 339–43. Cambridge, MA, USA 3-6 Oct. 1993.

[2] E. G. Coffman, Jr and M. Hofri. On the expected performance of scanning disks. In *SIAM Journal of Computing*, February 1982.

[3] E. G. Coffman, L. Klimko, and B. Ryan. Analysis of scanning policies for reducing disk seek times. In *SIAM Journal of Computing*, September 1972.

[4] English, R. M., and Stepanov, A. A. Loge: a self-organizing disk controller. In *Proc. of the 1992 Winter USENIX*, January 1992.

[5] H. Frank. Analysis and optimization of disk storage devices for time-sharing systems. In *Journal of the ACM*, pages 16(4):602–20, October 1969.

[6] R. Geist and S. Daniel. A continuum of disk scheduling algorithms. In *ACM Transactions on systems*, pages 5(1):77–92, February 1987.

[7] R. Hagmann. Low latency logging. Technical Report CSL-91-1, Xerox Corporation. Palo Alto, CA, February 1991.

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

[9] D. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7, Hewlett-Packard Lab., Palo Alto, CA, Feb, 1991.

[10] K. K.Ramakrishnan, Prabuddha Biswas and Ramakrishna Karedla. Analysis of file I/O traces in commercial computing environments. In *Proceedings of 1992 ACM SIGMETRICS and PERFORMANCE92*, pages 78–89, June 1992.

[11] M. Andrews M. Bender and L. Zhang. New algorithms for the disk scheduling problem. In *Proceedings of 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 580–589, October 1996.

[12] M. Seltzer, P. Chen and J. Ousterhout. Disk scheduling revisited. In *Proceedings of the Winter 1990 USENIX Conference*, pages 313–24. Washington, DC, USA 22-26 Jan. 1990.

[13] R. Y. Wang, T. E. Anderson, D. A. Patterson. Virtual log based file systems for a programmable disk. In *Proc. Third Symposium on Operating Systems Design and Implementation*, February 1999.

[14] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. In *IEEE Computer*, pages 27(3):17–29, March 1994.

[15] Chris Ruemmler and John Wilkes. Unix disk access patterns. Technical Report HPL-92-152, Computer Systems Lab. HP Lab. Palo Alto, CA, December 1992.

[16] Worthington, Ganger, Patt Y. N., Wilkes J. On-line extraction of scsi disk drive parameters. In *Performance Evaluation Review*, pages vol.23, no.1, p. 146–56, May 1995.

A Extracted Parameters for Seagate ST41601N

Zone	First Cylinder	Last Cylinder	SPT	First Logical Sector	Track Skew	Cylinder Skew
0	0	625	85	0	3	23
1	626	700	83	46	3	23
2	701	800	82	77	3	22
3	801	925	79	23	3	22
4	926	1050	78	54	3	21
5	1051	1175	76	36	3	21
6	1176	1300	73	20	3	20
7	1301	1400	72	28	3	20
8	1401	1500	69	64	3	19
9	1501	1600	68	48	3	19
10	1601	1800	65	28	3	18
11	1801	1900	63	49	3	18
12	1901	2000	62	2	3	17
13	2001	2100	61	48	3	17

Table 1: Zone specifications. SPT denotes numbers of sectors per track. It is cited from [16].

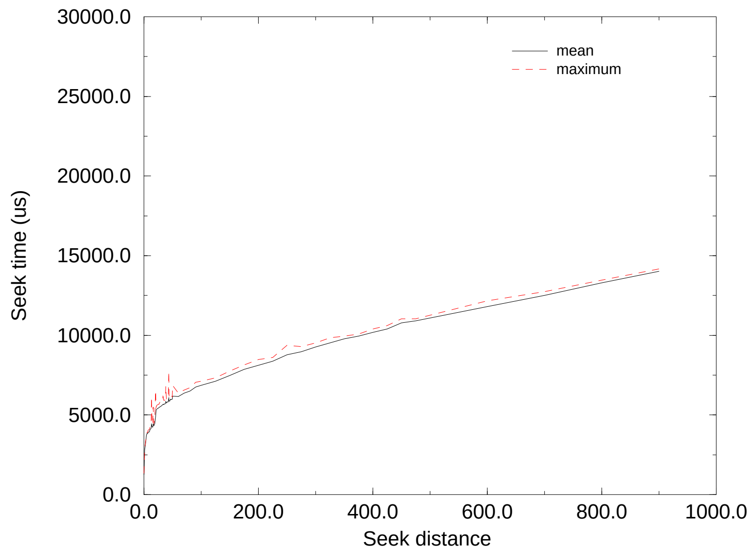


Figure 11: ST41601N Disk Seek Curve.

Seek Distance (cylinders)	**Delay(ms)**
1	1248
2	2782
3	3090
4	3456
5	3712
6	3833
7	3835
8	3917
9	3914
10	3941
11	4024
12	4110
13	4159
14	4401
15	4236
16	4242
17	4428
18	4321
19	4406
20	4473
21	4865
22	5327
23	5350
24	5393
25	5411
$25 < d < 400$	$3828.72 + 311.4 * \sqrt{d}$
$d >= 400$	7244.1 + 7.56*d

Table 2: Seek parameters for ST41601N. These values contain the SCSI command processing overhead. The parameters in the last two rows are approximated by fitting curves using *numerical least square analyses.*