submitted to the *First Conference on File and Storage Technologies* (FAST'02), Monterey, California, January 28-29, 2002.

Rotational Position Optimization (RPO) Disk Scheduling

Walter A. Burkhard* John D. Palmer°

*Gemini Storage Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114 USA

°Almaden Research Center
IBM Research Division
650 Harry Road
San Jose, CA 95120-6099 USA

July 16, 2001

Abstract

Rotational position optimization disk scheduling algorithms utilize seek distance versus rotational distance information implemented as rpo tables (arrays) which are stored in flashmemory within each disk drive. We consider a novel representation scheme for this information reducing the required flashmemory by a factor of more than thirty thereby reducing the manufacturing cost per drive. We present simulation results showing the throughput for conservative and aggressive versions of the scheme as well as comparative results with the standard production drives not using these results.

1 Introduction

Disk drive scheduling algorithms have been proposed and studied over the past 35 years in an attempt to improve performance. These algorithms utilize a stream of operation requests as input and arrange them as output. Many algorithms proposed to achieve better performance utilize disk internal state information; these algorithms are briefly presented in the first section. We consider the implementation of a good disk scheduling algorithm; our goal is to provide less costly disk drives by reducing the internal flash memory within a disk thereby reducing fabrication costs. We study the throughput achievable using a disk drive simulation model together with random uniformly distributed operations. We briefly overview various previous algorithms and then present our approach.

The paper is organized in seven sections as follows. We overview the state based disk scheduling algorithms, present our disk model and the IBM Ultrastar 18LZX command scheduling algorithm. The fifth section follows with a description of our approach reducing the rotational positioning algorithm tables. The sixth section contains an overview of the simulation as well as our results. Finally, we draw conclusions in section 7.

2 Classic Scheduling Algorithms

Numerous studies within the literature have considered command selection strategies such as *First Come First Served* (FCFS), *Shortest Seek Time First* (SSTF), and the SCAN scheduling algorithms [4, 3, 6, 11, 8, 2]. The SSTF approach reduces the average response time compared to FCFS over a wide range of workloads. The SCAN policies, including LOOK and C-SCAN, lower the variance in the response times; very recently, Worthington et al. combined LOOK

and C-SCAN and provide evidence of excellent performance for current disk drives [14]. Geist and Daniel proposed a continuum of algorithms between SSTF and LOOK that parametrically adjusts the propensity of the algorithm to change direction across the disk platter [5].

With the evolution of disk drives, standard disk form factors have shrunk and the full-stroke seek time has diminished considerably. Within modern disk drives, the full-stroke seek time is approximately twice the spindle full rotation time. This observation demands scheduling algorithms depending upon platter position as well as arm position.

Several studies within the literature have considered algorithms combining both the seek and rotational latencies. Jacobson and Wilkes consider the *Shortest Access Time First* (SATF) [10], Seltzer, Chen and Ousterhout consider *Shortest Time First* (STF) [13], Worthington, Ganger and Patt consider the *Shortest Positioning Time First* (SPTF) [14], and Heath, Pruett and Nguyen [7] consider an implementation of the these approaches.

More recently, Andrews, Bender and Quang consider the complexity of off-line disk scheduling which they show to be NP-complete. Moreover, they consider non-greedy on-line disk scheduling algorithms and claim to have contradicted a conjecture that the greedy algorithms mentioned above are optimal [1].

3 Disk Modeling

A disk drive is composed of several concentric annular *platters* mounted on a spindle motor. Platters are divided into concentric circular *tracks*; a *cylinder* is composed of the set of equal-radius tracks. The smallest unit that can be read or written to the disk is a *sector* which typically consists of 512 bytes of user data; individual sectors reside on a single track. Data is transferred to and from the disk via the set of read/write *heads* (one per platter surface.) The heads are mounted on ends of the actuator arms. The actuator moves all arms together and the heads will reside on the tracks of a cylinder.

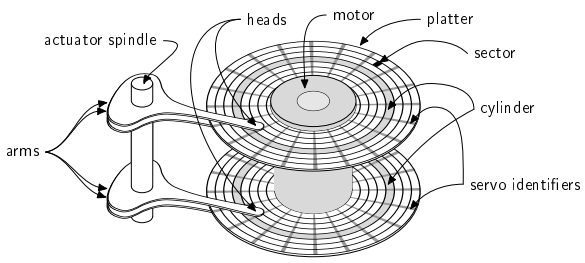

Figure 1. Disk drive internals

The *servo identifier* (sid) wedges consist of a unique synchronization pattern together with information identifying the cylinder and sector as well as centering information to position the head on the track; they are evenly spaced around the platter independent of the recording zone. All arm positioning as well as sector access is ultimately done via sids.

When the disk accesses a sector for reading or writing, the actuator must move the arms to the proper cylinder; then the desired sector must rotate under the head. A high-level pictorial description of a disk drive with two platters and four arms is presented in figure 1; table 1 contains typical times and disk drive architectural parameters circa 1999 for the IBM Ultrastar 18LZX drive [9].

form factor	3.5 inch
capacity	18.3 GB
platters	5
heads	10
rotation rate	10,000 rpm
servo identifiers per platter	90
average seek	4.9 ms
number of cylinders	11,712
number of zones	15
media data rate	23.3 ··· 44.3 MB/s

Table 1. IBM Ultrastar 18LZX Specifications

4 Rotational Position Optimization Command Scheduling

The IBM Ultrastar 18LZX command selection scheme, referred to as the *rotational position optimization algorithm,* is a greedy algorithm based upon the access time which depends upon both the seek and rotational latencies required to bring the desired physical block under the head. During the selection process, the access time for each command within the queue is determined and the command with the smallest time is selected.

Figure 2 presents the *reachable area* portion of the platter within the gray area; the disk rotates in the clockwise direction in the figure. Data in the reachable portion of the platter is accessible in one rotation from the current command and actuator position. The remaining data of the platter requires an additional rotation.

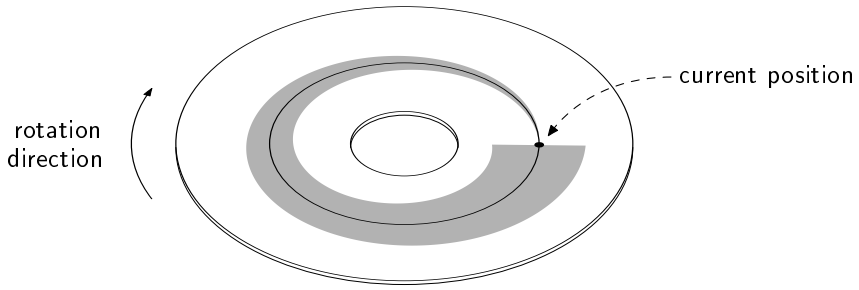

Figure 2. Platter reachable area

It is difficult in practice to determine precisely the boundary of the reachable area and a robust strategy for determining the relative location of the border is essential for access time calculation. The cost of making the incorrect decision will be an extra rotation, referred to as a *miss*, and such performance can be even worse than picking the next command at random. A conservative strategy is to consider only those commands far removed from the border; but, being too conservative reduces the benefits of the scheduling algorithm. However, if the strategy is too aggressive, the performance will also suffer from the additional misses.

Since the reachable area border is fuzzy around the edge, the IBM Ultrastar 18LZX drives use probabilistic data

regarding the edges of the reachable area. Rather than determining the access time for each command within the queue, the *expected access time* (EAT) is determined for each. The expected access time is the time necessary to perform a particular seek on the average. We discuss the calculation of a typical expected access time together with the internal tabular representation of the necessary data. Time is expressed in sids; each sid represents 6/90 ms = 44.44 μs as noted in table 1. Figure 3 presents the non-miss probability curve for a particular seek distance where the x-axis represents the rotational latency in sids from the current position and the y-axis represents the probability of reaching the desired cylinder within the rotational latency without a miss; these probabilities are referred to as *non-miss* probabilities.

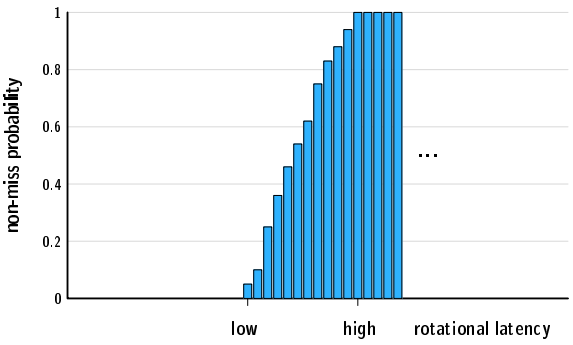

Figure 3. Typical non-miss probability curve for a particular seek distance

For a given seek distance d, there are two integral values low_d and high_d which specify a portion of *one* edge of the reachable area. As shown in figure 3, for any rotational latency rl at least as large as high_d, the probability of a non-miss is one and for rl less than low_d, the probability is zero. For each rotational latency rl, from low_d to high_d, a typical table will contain the expected access time which has been evaluated as

$$\text{EAT} \;=\; (1-p)\,(rl+90) \;+\; p \cdot rl$$

where p is the non-miss probability of reaching the desired cylinder within rl sids. The expected access time data for rotational latencies is stored within a table; these tables are referred to as the *rotational position optimization* (rpo) tables. For each seek distance d, the values high_d and low_d are stored together with the expected access times for rotational latencies between low_d and high_d. Since the platter is continuously rotating, the current location typically is the location at the termination of the current command. Determining the non-miss probabilities is an interesting question but beyond our immediate study; however we return to this topic within the our conclusion.

Accordingly, the expected access time rpo tabular data is pre-computed for the Ultrastar 18LZX drive family; it is stored on each disk drive in flash memory. Our approach ultimately will reduce the size of the necessary flash memory while maintaining acceptable throughput performance. Multiple tables are used, each describing a particular mode of operation, e.g. inward moving read operation. There are a number of factors impacting edge determination such as operation type: read or write, absolute head location whether near the inner or outer portions of the platter, seek direction: inward or outward, etc.

The general expected access time EAT calculation is approximated within the following expression where *table* is

an internal table. For seek distance d and rotational latency rl from the end of the current command first to the desired command

$$\text{EAT}\,(\,rl\,,\,d\,) = \begin{cases} rl + 90 & rl < \text{low}_d \\ rl & rl > \text{high}_d \\ \textit{table}\,[\,rl - \text{low}_d\,] & \text{otherwise} \end{cases}$$

The precise description of the EAT calculation follows the next paragraph where cylinder groups are introduced.

Each high, low, and EAT value can be made smaller than 256 and will reside within a byte. Accordingly as described, a table with 11712 slots will require approximately 350KB of flash memory. It is, however, not necessary to have a slot for each seek distance (in cylinders) because the time intervals being measured (sid intervals) are relatively coarse. It is only necessary to include enough slots to differentiate times over 2.5 revolutions of interest which is approximately 235 slots. Now each slot will correspond to a contiguous sequence of seek distances; each such sequence is referred to as a *cylinder group*. The ensemble of cylinder groups is non-overlapping and covers the 11712 seek distances. The first cylinder groups, corresponding to seeks where the actuator is moving very slowly, contain very few seek distances. The latter cylinder groups, corresponding to seeks where the actuator is moving rapidly at a constant rate, contain many more seek distances. The IBM Ultrastar 18LZX implements rpo tables each with 227 slots; accordingly, each table requires approximately 6.9KB of flash memory.

The expected access time for logical block (sector) address (LBA) N is calculated as follows:

1. Calculate the physical block cylinder, track and sid for N.
2. The absolute difference between the current cylinder and the desired cylinder
   specifies the cylinder group d which determines the table slot entry.
3. The difference between the sid of N and the current sid yields EAT.
   If EAT $<$ 0, then EAT := EAT + 90 .
4. if EAT $<$ minimum operation time, then EAT := EAT + 90 .
5. high := *table* [d].high low := *table* [d].low
6. if EAT $<$ low, then EAT := EAT + 90
7. if EAT $<$ high and EAT $\geq$ low, then EAT := *table* [d] .eat[EAT – low]
   return EAT

Step 1 requires knowledge of the zoned recording data layout. The seek distance d, referred to as the cylinder group, is determined next as is the seek direction. Then, in step 3, the diference in sid between N and the (termination of the) current operation is our first approximation to the expected access time. Since the platter rotates in one direction, EAT must be positive. One rotation corresponds to 90 sids as noted in table 1. Moveover, if EAT is less than a fixed operation dependent minimum, 16 sids for read and 24 sids for write, which corresponds to operation initialization, EAT is incremented by one rotation. Step 5, obtains the low and high values for the desired cylinder group. Then if EAT is less than low, EAT is incremented by 90. Finally, in step 7, the EAT value is obtained from the tabular data for sids within the range low to high. Within steps 4 and 6, increasing EAT by 90 sids represents an extra rotation. Of course within step 7, misses will occur but the expected value calculation includes this.

5 Our Approach

The discussion thus far describes the IBM Ultrastar 18LZX rpo scheduling algorithm. Now we consider an approach to further reduce the flash memory size while maintaining acceptable performance; our discussion centers on diminishing the data stored within an rpo table slot.

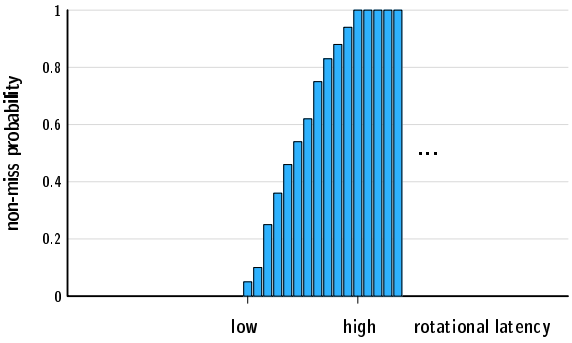

Figure 4. Typical non-miss probability curve for a particular cylinder group

For each cylinder group, the probability of a non-miss can be specified as in figure 4; these curves specify the non-miss probability as a function of the rotation latency in sids, relative to the termination sid of the current operation. Any rotational latency at least as big as high has probability one of non-miss access. Rather than storing all high – low values, we will consider storing only one value; the non-miss probability curve is represented as a *step function.* Either we have 100% chance of a miss or 0% chance within this approach. The reduction in space will be considerable and the throughput performance will remain acceptable.

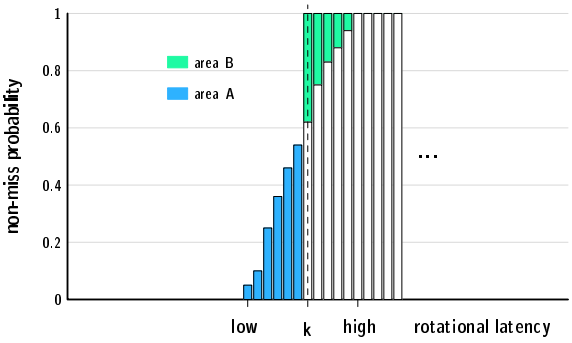

Figure 5. Typical non-miss probability curve for a particular cylinder group

Figure 5 contains, for a particular table and cylinder group, the prototypical non-miss probability curve showing the step at sid k. The shaded area to the left of k designated as area A indicates the fraction of the operations within this cylinder group with rotational latency between low and k that will be slated for an extra rotation even though there is a non-zero probability of success without a miss. Diminishing area A will yield a more aggressive scheduling algorithm. Similarly, the shaded complementary area above our probability curve to the right of k, designated as area

B indicates the fraction of the operations within the cylinder group with rotational latency between k and high that will be slated to be selected but will incur the runtime cost of a miss. Diminishing area B will also yield a more conservative scheduling algorithm. We use the ratio of the two areas designated α to select k for each cylinder group thereby maintaining a similar strategy throughout.

$$\alpha = \frac{\text{area A}}{\text{area B}} \qquad \text{where} \qquad \text{area A} = \sum_{i\,=\,\text{low}}^{\text{k}-1} p_i \qquad \text{and} \qquad \text{area B} = \sum_{i\,=\,\text{k}}^{\text{high}} 1 - p_i$$

where p_i is the probability an access occurs for the desired cylinder within i sids without a miss. Thus for a given α, the integral k can be determined; typical values for α range from 0.5 to 10 within our experiments. Large α represents conservative strategies with only operations with larger rotational latencies deemed plausible; small α represents aggressive strategies.

Implementing the tables using step functions reduces the size of each table by a factor of more than thirty to 227 bytes of flash memory since each slot contains only one value k each residing within a byte.

6 Simulation Study

Our simulation study involves only the command scheduling portion of the disk drive in which the Ultrastar 18LZX rpo algorithm, rewritten in C++ for ease of simulator system maintenance, is utilized; the simulator was configured using parameters provided within table 1. The simulator is detailed enough to read assembler files, of the variety used within the production drives, conveying the rpo tables; all rpo tables used in our experiments are expressed in this format.

The simulator system contains instances of the rpo algorithm configured with various rpo tables. The step-function rpo algorithm, referred to as the *step-function model,* uses the tables defined in section 5. The step-function model maintains a command queue of constant length Q; as each command is completed, a new command is selected from a uniformly distributed logical block (sector) address space to maintain the length. The runtime for individual commands is determined using an another instance of the rpo algorithm, referred to as the *timer model,* that contains the standard rpo tables. Once the step-function model selects its next command, the command is given to the timer model to obtain the command run time as well as the miss probability. The timer model runs only the commands selected by the step-function model in exactly the same order; accordingly, the current position at the end of each command will be identical in both the step-function and timer models.

For comparison purposes, we also have a *plant model* that also contains the standard rpo tables. This model also maintains a command queue of constant length Q adding a new command as each is completed; it will calculate its own run times and miss probabilities The same new command is included in both the plant model and the step-function model queues thereby maintaining identical queue lengths and similar command executions. Both command queues initially contain the same set of commands; accordingly, at the end of a simulation run, typically 10,000 commands, both the plant and the step-function models have processed the nearly the same set of commands but in possibly different orders. The only commands not processed by both are the commands remaining within exactly one of two queues of the step-function and plant models at the end of the run.

Our study considers uniformly distributed commands over four different ranges of the disk; the four ranges involve 0.7%, 11.1% 98.8% and 100% of the LBA space. This variety of test is typical of various large scale commerical disk drive consumers. The results for queue lengths Q = 1, 4, 8, 16, 32, and 64 are presented; typically the queue occupancy is bursty and very small [12]. Each of these twenty-four experiments processed 50,000 commands. We present the standard model as well as the step-function model results. We also present the best choice for α within each of the 24 LBA range and queue length combinations. Table 2 contains the four LBA ranges and percentages.

LBA id	LBA range	%
A	200000..400000	0.7
B	0..4000000	11.1
C	200000..35800000	98.8
D	0..36000000	100.

Table 2. Test suite LBA ranges

Table 3 contains the standard rpo table read throughput results; the 95% confidence interval is provided too.

LBA	Q = 1	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	232.6; 4.40	374.6; 7.22	438.7; 8.39	492.4; 9.35	533.2; 10.15	562.5; 10.76
B	178.9; 3.30	253.1; 4.70	301.1; 5.54	345.1; 6.31	390.5; 7.14	438.0; 8.05
C	120.4; 2.21	163.3; 2.97	188.4; 3.43	219.8; 4.02	253.9; 4.64	291.4; 5.31
D	119.9; 2.20	162.9; 2.96	188.2; 3.43	217.8; 3.98	253.0; 4.62	289.4; 5.27

Table 3. Standard rpo tables; read throughput with 95% confidence interval

Table 4 contains the standard rpo table write throughput results together with the 95% confidence interval.

LBA	Q = 1	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	207.2; 3.87	296.7; 5.64	338.4; 6.37	375.2; 6.98	408.4; 7.49	435.3; 7.91
B	157.4; 2.88	204.5; 3.77	236.0; 4.36	267.1; 4.92	298.0; 5.48	328.2; 6.03
C	108.1; 1.97	141.8; 2.57	158.5; 2.87	176.6; 3.21	197.3; 3.61	222.1; 4.08
D	107.8; 1.96	141.2; 2.56	158.1; 2.86	176.2; 3.20	196.8; 3.60	220.6; 4.06

Table 4. Standard rpo tables; write throughput with 95% confidence interval

Tables 5 and 6 contain read and write throughtput results for the believed model together with the associated α value yielding the highest throughput; twenty-one candidate α values were considered 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, and 9. When Q is one, the throughput is independent of the rpo tables.

LBA	Q = 1	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	232.6	368.0; 1.00	429.8; 1.25	478.6; 1.50	509.1; 1.50	530.7; 2.00
B	178.8	249.8; 1.50	296.5; 2.00	337.7; 2.00	376.9; 2.00	420.8; 2.00
C	120.2	162.4; 1.25	186.5; 1.25	216.5; 1.75	249.1; 2.50	285.4; 3.00
D	119.6	162.0; 1.25	186.3; 1.25	215.5; 1.75	248.9; 2.00	283.8; 2.00

Table 5. Step function rpo tables: read throughput with optimal α

LBA	Q = 1	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	207.2	286.0; 1.25	322.7; 2.50	359.1; 5.00	388.8; 7.50	410.0; 8.00
B	157.3	199.2; 1.00	227.3; 1.25	255.8; 1.25	284.8; 1.75	316.2; 5.00
C	107.3	139.4; 1.00	155.0; 1.25	170.8; 1.25	189.6; 1.25	212.7; 1.25
D	107.0	138.8; 1.25	154.5; 1.25	170.2; 1.25	189.3; 1.25	212.9; 1.25

Table 6. Step function rpo tables: write throughput with optimal α

Table 7 presents the throughput percentage differences between the optimal α step-function rpo tables and the standard rpo tables. When Q is one, the difference is zero.

	read					*write*				
LBA	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	1.8	2.0	2.8	4.5	5.7	3.6	4.6	4.3	4.8	5.8
B	1.3	1.5	2.1	3.5	3.9	2.5	3.7	4.2	4.4	3.7
C	0.4	1.0	1.5	1.9	2.1	1.2	2.1	3.2	3.9	4.2
D	0.4	1.0	1.1	1.6	1.9	1.1	2.1	3.3	3.8	3.5

Table 7: Read & Write percentage difference: throughput for optimal α and standard rpo tables

We also present, in Tables 8 and 9, the throughput results when α is constant 1.25; this variety of rpo tables for constant α is practicable.

LBA	Q = 1	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	232.6; 4.40	368.0; 7.10	429.8; 8.35	472.6; 9.47	493.9; 10.39	500.5; 10.99
B	178.9; 3.30	249.8; 4.66	294.7; 5.51	331.7; 6.29	365.1; 7.13	400.8; 8.11
C	120.4; 2.21	162.7; 2.96	186.6; 3.41	215.8; 3.98	247.4; 4.61	280.5; 5.27
D	119.9; 2.20	162.2; 2.95	186.4; 3.41	214.9; 3.96	247.1; 4.60	279.7; 5.25

Table 8. Step function rpo tables when α is 1.25; read throughput with 95% confidence interval

LBA	Q = 1	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	207.2; 3.87	286.1; 5.49	318.9; 6.27	337.5; 6.90	347.8; 7.37	352.7; 7.68
B	157.4; 2.88	198.0; 3.64	227.3; 4.23	255.8; 4.83	282.3; 5.43	305.8; 6.01
C	108.1; 1.97	140.1; 2.53	155.2; 2.82	170.9; 3.13	189.6; 3.52	212.7; 4.01
D	107.8; 1.96	139.7; 2.53	154.8; 2.81	170.3; 3.12	189.3; 3.52	212.9; 4.00

Table 9. Step function rpo tables when α is 1.25; write throughput with 95% confidence interval

For short queue lengths, the constant α = 1.25 is a good choice; table 10 presents the throughput percentage difference between the optimal α and the α = 1.25 rpo tables. For queue length one, there is no difference. When the optimal α is 1.25, the throughputs in tables 8 and 9 may differ slightly from those in tables 5 and 6 because of statistical variation.

	read					*write*				
LBA	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64	Q = 4	Q = 8	Q = 16	Q = 32	Q = 64
A	0.0	0.0	1.3	3.0	5.7	0.0	1.2	6.0	10.5	14.0
B	0.0	0.6	1.8	3.1	4.8	0.7	0.0	0.0	0.9	3.3
C	0.0	0.0	0.3	0.6	0.7	0.5	0.0	0.0	0.0	0.0
D	0.0	0.0	0.3	0.7	0.7	0.0	0.0	0.0	0.0	0.0

Table 10: Read & Write throughput percentage difference: for optimal α and α = 1.25 rpo tables

7 Conclusions

Our results indicate the existence of workload domains where the step function rpo tables provide very acceptable performance especially when α is 1.25. Disk drives operating with a rather light workload (a non-server environment) would be ideal candidates for such cost reducing improvements. In such environments, the slight degradation in throughput would be tolerable. These modified drives will operate in a more aggressive mode in the domains characterized by longer queues or accessing a smaller portion of the disk with a throughput reduction caused by additional misses.

We mentioned the probability determination task at the end of section 4. For the Ultrastar family of drives, the values were determined en masse; that is to say, the values were determined for a few sample drives and these values became *the* probabilities to be used throughout the family. An interesting question would be how to efficiently determine these value during the manufacturing initialization phase dynamically for each disk. A second related issue is the notion of continuously upgrading the rpo tables as the disk ages, the environment changes, etc. Typically environmental changes include changes in temperature and gravity but this might also include queue length, preponderance of operation variety, the fraction and portion of the disk being accessed, as well as the command queue length.

A much longer term question is whether the greedy disk scheduling algorithms are the best possible. In at least one

study, the claim is that non-greedy ordering of the elements within the queue can obtain slightly better performance.

References

[1] Matthew Andrews, Michael A. Bender, and Lisa Zhang. New algorithms for the disk scheduling problem. In *Proceedings of the IEEE Foundations of Computer Science (FOCS'96)*, pages 550–559, 1996.

[2] Edward G. Coffman and Micha Hofri. On the expected performance of scanning disks. *SIAM Journal of Computing*, 11(1):60–70, February 1982.

[3] E.G. Coffman, L.A. Klimko, and B. Ryan. Analysis of scanning policies for reducing disk seek times. *SIAM Journal of Computing*, 1(3), September 1972.

[4] Peter J. Denning. Effects of scheduling on file memory operations. In *Proceedings of AFIPS Spring Joint Computer Conference*, pages 9–21, April 1967.

[5] Robert Geist and Stephan Daniel. A continuum of disk drive scheduling algorithms. *ACM Transactions on Computer Systems*, 5(1):77–92, February 1987.

[6] C.C. Gotlieb and G.H. MacEwen. Performance of movable-head disk storage systems. *Journal of the ACM*, 20(4):604–623, October 1973.

[7] Mark A. Heath, D. Christopher Pruett, and Bang C. Nguyen. Method for reducing rotational latency in a disc drive. United Stated Patent number 5,570,332, October 1996.

[8] Micha Hofri. Disk scheduling: FCFS vs. SSTF revisited. *Communications of the ACM*, 23(11):645–653, 1980.

[9] IBM. IBM Ultrastar 18LZX and 36ZX hard disk drive. Product sheet: `www.storage.ibm.com/hardsoft/diskdrdl/us18lzx36zx.htm`, 1999.

[10] David M. Jacobson and John Wilkes. Disk scheduling algorithms based on rotational position. Technical report, Hewlett-Packard Companry, Technical report HPL-CSP-91-7rev1, 1995.

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

[12] Chris Ruemmler and John Wilkes. UNIX disk access patterns. In *Proceedings of the Winter 1993 USENIX Conference*, pages 405–420, January 1993.

[13] Margo Seltzer, Peter Chen, and John Ousterhout. Disk scheduling revisited. In *Proceedings of the Winter 1990 USENIX*, pages 313–324, January 1990.

[14] Bruce J. Worthington, Gregory R. Ganger, and Yale N. Patt. Scheduling algorithms for modern disk drives. In *Proceedings of the ACM SIGMETRICS Conference*, pages 241–251, May 1994.