Towards Optimal I/O Scheduling for MEMS-based Storage

Hailing Yu Divyakant Agrawal Amr El Abbadi
Department of Computer Science, University of California,
Santa Barbara, CA, 93106
{*hailing,agrawal,amr*}*@cs.ucsb.edu*

Abstract

MEMS-based storage devices are currently being developed to narrow the gap between processor and disk speeds. MEMS-based storage devices have a different architecture from disk devices, thus algorithms, such as I/O scheduling and data placement, designed for disks need to be revisited. In this paper, we focus on developing an I/O scheduling algorithm for MEMS-based storage devices. Our theoretical analysis shows that this algorithm is guaranteed to perform within twice the optimal time for any workload.

1. Introduction

Although magnetic disks have dominated the persistent storage technology, recently the memory hierarchy has suffered from the problems of latency, bandwidth, and cost gap. In particular, due to advances in processor technology and semiconductor manufacturing, the processor-to-disk performance gap has been consistently growing. Currently this gap has widened to six-orders of magnitude and future trends indicate that unless a breakthrough occurs in the disk technology, this gap will continue to widen by about 50% annually.

Micro-ElectroMechanical Systems (MEMS) [1, 5] based storage systems are currently being developed as an alternative to conventional rotational disks for the non-volatile storage of large amounts of data. By using photolithographic processes similar to those employed in manufacturing traditional semiconductor devices, MEMS-based storage devices can be produced and manufactured at a very low cost.

MEMS-based storage has quite different characteristics from disks. In particular, they are two dimensional while disks have been approached always as one dimensional storage devices by dividing them into cylinders, sectors and tracks. Even though existing techniques developed for disks, such as disk scheduling algorithms and data placement schemes, can be adapted to MEMS-based storage devices, some characteristics of MEMS-based devices have not been considered adequately. In this paper, we first note that optimal scheduling for MEMS-based storage is NP-complete and then propose off-line and on-line scheduling algorithms that exploit the two-dimensional characteristics of these devices. We then show that these algorithms are guaranteed to perform within twice the optimal scheduling time.

2. Background

In this section, the architecture of MEMS-based storage devices is first described. Our development is based on the CMU's CHIPS project [1] and the IBM Millipede project [5]. Then we briefly review existing scheduling algorithms proposed for this new storage device.

2.1. Architecture for MEMS-based Storage

A MEMS-based storage device is composed of recording media heads and a recording media surface. The recording heads, usually called tips, are embedded in a semiconductor wafer arranged in a rectangular fashion. The recording media is another rectangular silicon wafer referred to as the media sled. There are different approaches for recording data. For example, IBM's Millipede [5] uses pits in the polymers made by tip heating, CMU's CHIPS [1] adopts the same techniques as data recording on disks. In this system, because it is very hard to rotate the unit on a microscopic scale, the media sled is suspended by springs above the wafer with probe tips. The movement in the Z direction is used to actuate the distance between the probe tips and the media sled. This design is shown in Figure 1 (Based on the CMU design [6]). The X and Y actuators provide the force for moving the media sled in X and Y directions while the spring supplies the restoring motion. These two actuators work independently.

The media sled is divided into rectangular regions as shown in Figure 2. Each of these rectangular regions contains an array of $M \times N$ bits and is serviced by one probe

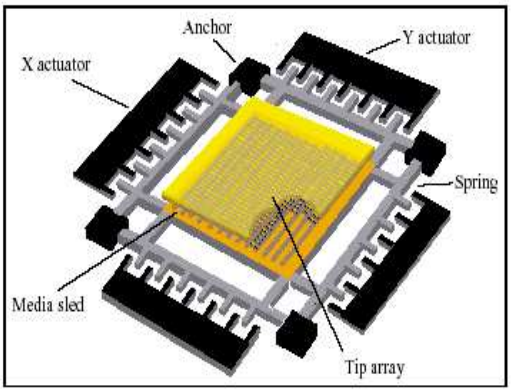

Figure 1. A design of MEMS-based storage devices

tip. The relation between the regions and tips is a one-to-one mapping; i.e., the number of regions is the same as the number of probe tips. In theory, all the probe tips can be activated simultaneously. For the CMU's CHIPS device, the system has 6400 tips, arranged in an array of 80×80 tips per rectangular region with each region having 2500×2500 $(M \times N)$ bits. Due to power and heat constraints, only 1280 tips can be activated simultaneously.

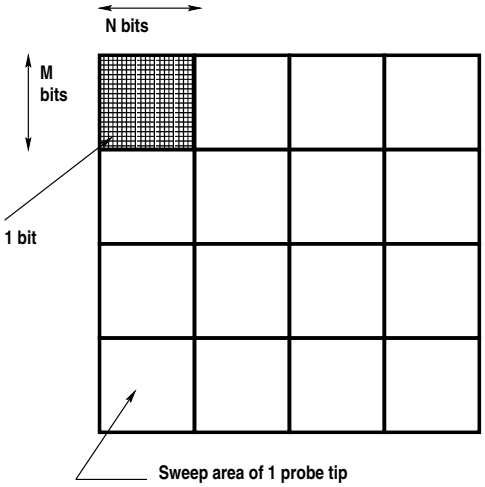

Figure 2. The media sled is divided into rectangular regions

Based on the above design considerations, some basic observations and assumptions can be made:

- Because the relation between the probe tips and regions is a one-to-one mapping, it can be assumed that the max distance the media sled can move in the $X(Y)$ direction is the edge length of regions in the $X(Y)$ direction.
- The infinity distance L_∞ between two points, (x_1, y_1) and (x_2, y_2), is the larger value of $|x_1 - x_2|$ and $|y_1 - y_2|$. Because the movement in the X and Y directions are independent, the distance between two points in one region is L_∞.
- The time T for the sled to move from one point to another is an increasing function of the distance in the X and Y directions [4]. Based on the above observation, time T is the larger of these two values (time spent on the X (called T_x) and Y (called T_y) direction movement. Note that time is symmetric, i.e. $T(v,w) = T(w,v)$, where v, w are two points.

2.2. Existing scheduling algorithms for MEMS-based storage

Many different scheduling algorithms have been developed for conventional disks [7], such as FCFS (first-come first-service), CLOOK (cyclical look), SSTF (shortest seek time first), SSTF_LBN (shortest seek time based on the Logical Block Number of the request), and SPTF (shortest position time first). The CMU group has adapted many of these disk scheduling algorithms in the context of MEMS-based storage by mapping these storage devices into a disk-like interface [3]. The CMU experimental results showed that SPTF performs best in terms of average response time.

3. Theoretical Development

In MEMS-based storage devices, requests can be mapped to (x, y) locations on a two-dimensional surface, where x, y determine the position of a request on the surface. Because the tip's number is not related to the seek time, and scheduling algorithms finding the shortest seek time to serve all requests, we can simply ignore the active tip's number. Requests for a MEMS-based storage can be viewed as points distributed in one rectangular area which is the same as one region. When the device serves requests, the media sled moves the request's position determined by the x and y, and the corresponding tips are activated, then the device accesses (reads or writes) data by the activated tips. The time spent moving the media sled from its current position to the next position is called seek time T, which is determined by the L_∞ distance between these two positions. After mapping requests into (x, y) locations in a two-dimensional surface, given a set of requests, a graph can be constructed: Requests are denoted as vertices, an edge represents the time required to move from one vertex to another. The optimal solution is to find the shortest path that visits each vertex exactly once in the graph. This corresponds to the Symmetric Traveling Salesman Problem, so finding the optimal path can be shown to be NP-complete.

In the existing disk-based algorithms, SPTF performs best [3]. However it is easy to show that it does not perform well in all settings. For example, consider the case where all requests have different x values, but the same y value, as shown in Figure 3. Assume request R_3 is the current request being served, because R_2 is nearer to R_3 than R_4, then using the scheduling criterion of SPTF, R_2 is the next request to be served. Applying the same reasoning, the order in which the requests will be served under SPTF is R_3, R_2, R_4, R_1, R_5. The problem is that this algorithm is greedy, and finds the next request to serve based on the shortest seek time and does not consider the entire distribution of requests. In this example, if all requests are considered, the minimal order in terms of seek time would be R_3, R_2, R_1, R_4, R_5.

Figure 3. A setting that SPTF does not perform well

Even though it is not practical to design an optimal algorithm, we develop an algorithm with guaranteed upper bound for any workload. The algorithm is based on serving requests in the order of the double walk of a minimum spanning tree(MST) for all requests. We have explored some interesting properties of MST in the infinity distance space to reduce its construction cost [8].

These properties are established based on the definitions of regions with respect to a vertex in the infinity distance space. Consider a vertex v with coordinates(x,y), we define the following four regions with respect to $v(x,y)$ as follows. Region1 is the subspace with any vertex (x_1,y_1) satisfying $x_1 > x, y_1 \geq y$; Region2 is the subspace with any vertex (x_2,y_2) satisfying $x_2 \leq x, y_2 > y$; Region3 is the subspace with any vertex (x_3,y_3) satisfying $x_3 < x, y_3 \leq y$; and Region4 is the subspace with any vertex (x_4,y_4) satisfying $x_4 \geq x, y_1 < y$.

We have shown that in an MST, any vertex in any region can have at most two neighbors [8]. Using this result, we can next establish that the degree of any vertex is at most eight. We then derive a stronger result, namely that in the L_∞ model and a set of vertices, there exists some MST where the degree of every vertex is at most four. Finally we derive a way to reduce the cost of traversing a MST, such that when traversing the MST, the cost of moving from one vertex u to its sibling v directly is no larger than the cost of passing through the parent vertex of u and v.

4. Scheduling Algorithms

Our approach is based on building a minimum spanning tree of all requests and serving the requests in the double walk order. An undirected graph (called cost graph) needs to be constructed in order to build a minimum spanning tree. Since it is possible to traverse from a request (x_i,y_i) to any other request (x_j,y_j), the number of edges in the cost graph will be $n(n-1)2$ edges. Both Prim's and Kruskal's algorithms for constructing MST have time complexity $O(mlogn)$, where m is the number of edges and n is the number of vertices. By applying the properties of MST in the infinity distance space, each vertex in an MST could have one nearest neighbor in each region. If these four nearest neighbors for each vertex are in a cost graph G, the MST can be built based on graph G. So we develop a procedure to construct the cost graph with at most $8n$ edges [8]. Thus the cost of constructing the MST reduces to $O(nlogn)$ when restricted to L_∞ distance.

We develop an off-line algorithm for the case when all the requests are known a priori. In the algorithm, a cost graph is first constructed with at most $8n$ edges. An MST is built by Prim's Algorithm. The requests are served in a preorder traversal of the MST. Based on the optimization described above, moving from the current request position to the next request position directly is no larger than the cost of passing through the parent vertex, thus during the preorder serving of requests, we move the media sled from the current request position to the next request position directly.

We also designed an on-line algorithm where requests are continually arriving as prior requests are being served. The procedure for serving requests is the same as the off-line algorithm. We have developed an algorithm to update the current MST dynamically. Based on the fact that there exists some MST where the degree of every vertex is at most four, we find the four nearest vertices of the new vertex in the different regions, if they are exist. The new vertex is connected to the MST by the smallest cost edge to one of these four vertices. The remaining edges are checked to see if the MST cost can be reduced. The time required to update the MST in the online case is $O(n)$.

5. Analysis

Our approach is based on the properties of MST. In Section 5.1, we show that for any workload, our approach can satisfy an upper bound. A preliminary performance study is presented in Section 5.2.

5.1. Upper bound

Without the optimization of moving from one vertex u to its sibling v directly, our approach corresponds to a double walk of an MST. The cost of a double walk is equal to two times the cost of this spanning tree. If the spanning tree is an MST, the cost of a double walk is equal to $2 * T_{MST}$ (T_{MST} is the cost of the MST). The optimal algorithm tries to find a path with the minimum cost. Actually the path is a spanning tree too. So the cost of the optimal path T_{opt} is no less than T_{MST}. If we serve the requests by the route of a double walk, the following can be established:

$T_{seek} = 2 * T_{MST}, T_{MST} \leq T_{opt}$.

Thus $T_{seek} \leq 2 * T_{opt}$, where T_{seek} is the total seek time for serving all requests. When considering moving from one vertex u to its sibling v directly instead of passing the parent of u and v, we get the following equation: $T_{seek} < 2 * T_{opt}$. Hence, our approach is guaranteed to have an upper bound of $2 * T_{opt}$ irrespective of the workload.

5.2. Preliminary Performance

A preliminary performance study was also performed. We set up the experiments by simulating a 100×100 region with uniformly distributed workloads. We implemented four scheduling algorithms: FCFS, SSTF, SPTF and our off-line algorithm. Because the seek time is a function of seek distance, we give simulation results based on the average seek distance instead of average seek time. The simulation results are shown in Figure 4.

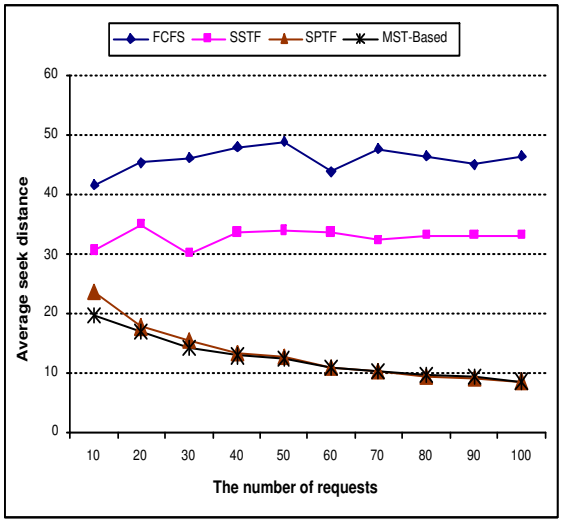

Figure 4. Average seek distance.

The data shows that SPTF and MST-based approaches always perform better than SSTF and FCFS. SSTF is better than FCFS. As the number of requests increases, the average seek distance of FCFS and SSTF is almost a constant, so these two algorithms do not scale well. However the average seek distance of SPTF and MST-based approach decreases. When the workload is not heavy, the MST-based approach achieves better average seek distance than SPTF. Under heavy workload, the result shows that their performance is comparable.

6. Conclusion

MEMS-based storage devices are being developed to alleviate the storage/processor bottleneck. Their unique characteristics differentiate them from the conventional disks. The I/O scheduling algorithms and data placement schemes developed for disks need to be revisited. In this paper, we developed an I/O scheduling algorithm based on the two-dimensional property of MEMS-based storage. Our theoretical analysis shows that the new scheduling algorithm can guarantee the upper bound of $2 * T_{opt}$ on seek time. The preliminary simulation results showed that the new algorithm and SPTF have comparable performance under uniformed workload. Our future work will conduct more experiments for different workloads, consider the starvation issue, and optimize this algorithm to achieve an upper bound of $1.5 * T_{opt}$ based on the algorithm designed by Christofides [2].

References

[1] CMU CHIPS project. 2002. http://www.lcs.ece.cmu.edu/research/MEMS.

[2] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. *Report 388, Grad School of industrial Administration, CMU*, 1976.

[3] J. Griffin, S. Schlosser, G. Ganger, and D. Nagle. Operating systems management of MEMS-based storage devices. *OSDI*, 2000. http://www.lcs.ece.cmu.edu/research/MEMS/.

[4] T. M. Madhyastha and K. P. Yang. Physical modeling of probe-based storage. *Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems*, 2001.

[5] P.Vettider, M.Despont, U.Durig, W.Haberle, M. Lutwyche, H.E.Rothuizen, R.Stuz, R.Widmer, and G.K.Binnig. The "millipede"-more than one thousand tips for future afm storage. *IBM Journal of Research and Development*, pages 44(3):323–340, May 2000.

[6] S. W. Schlosser, J. L. Griffin, D. F. Nagle, and G. R. Ganger. Designing computer systems with MEMS-based storage. *ASPLOS*, 2000. http://www.lcs.ece.cmu.edu/research/MEMS/.

[7] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. *USENIX Conference Proceedings (Washington, D.C.)*, pages 313–337, 1990.

[8] H. Yu, D. Agrawal, and A. E. Abbadi. Towards optimal I/O scheduling for MEMS-Based storage. *UCSB Technical Report 2002-22*.