Towards Optimal I/O Scheduling for MEMS-based Storage

Hailing Yu, Divyakant Agrawal, and Amr El Abbadi

Department of Computer Science
University of California, Santa Barbara
{hailing,agrawal,amr}@cs.ucsb.edu

1. Introduction

Magnetic disks have been the dominant on-line storage technology for more than three decades. Disks have maintained this dominance in spite of other competing storage technologies such as bubble memory, holographic stores, and improved DRAMs. Although magnetic disks have enjoyed this position in the persistent storage arena, more recently the memory hierarchy has suffered from problems of latency, bandwidth, and cost gap. In particular, due to the advances in processor technology and semiconductor manufacturing, the processor-to-disk performance gap has been consistently growing. Fortunately, the processor-to-memory performance gap in the memory hierarchy has been partially mitigated by the integration of very fast caches. In spite of these efforts, the RAM-to-DISK gap has remained unfilled. 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. The RAM-to-Disk performance gap arises due to the physical characteristics of disk drives. Although the disk capacity growth has been phenomenal (about 60% per year), the mechanical positioning system in the disks limits the access times improvements to only about 7% per year.

Micro-ElectroMechanical Systems (MEMS) [11,12] based storage systems are being developed as an alternative to conventional rotational disks for the non-volatile storage of large amounts of data. MEMS are extremely small mechanical structures formed by the integration of mechanical elements, actuators, electronics and sensors. These are fabricated on silicon chips using photolithographic processes similar to those employed in manufacturing standard semiconductor devices. As a result, MEMS devices can be produced and manufactured at a very low cost. MEMS based systems can be used in a variety of applications due to their improved cost, size, capacity and power consumption. Specifically, MEMS based systems can be used in mobile application platforms such as PDA's, laptops and bio-medical monitoring systems.

Unlike traditional disks, MEMS based storage devices [2] do not make use of rotating platters due to the difficulty in manufacturing efficient and reliable rotating parts in silicon. The emerging paradigm for such systems is that of a large scale MEMS array which, like disk drives, has read/write heads and a recording media surface. The read/write heads are probe tips mounted on micro-cantilevers embedded in a semiconductor wafer and arranged in a rectangular fashion. The recording media is another rectangular silicon wafer (called the media sled) that can use conventional techniques for recording data. In general power consumption of MEMS devices is considerably lower than the conventional disks which means that when these devices are integrated as part of a system, designers do not need to worry about costly idle time prediction algorithms to conserve power. Preliminary studies have also shown that stand-alone MEMS based storage devices reduce I/O stall times by 4 to 74 times over disks and improves the overall application run times by 1.9X to 4.4X [8]. When used as on-board caches for disks, MEMS based storage improves I/O response time by up to 3.5X.

MEMS-based storage has quite different characteristics from disk. 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 scheme, can be adapted to MEMS-based storage devices, some characteristics of MEMS-based storage devices have not been considered adequately. In this paper, we first show 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 performance time.

The remainder of the paper is organized as follows. Section 2 presents the MEMS-based storage model and a brief review of existing scheduling algorithms. In Section 3, we show that optimal MEMS-based

storage scheduling is NP-complete and exploit some properties of minimum spanning trees in the infinity distance space. In Section 4, off-line algorithm and on-line algorithm are developed. In Section 5, we analyze the proposed algorithms by developing an upper bound on their performance and providing some preliminary experiment results. Section 6 concludes the paper.

2. Background

In this section, the architecture of MEMS-based storage devices is first described. Our development is based on the CMU CHIP project [9] and the IBM Millipede project [7]. Then we review some existing scheduling algorithms for this new storage device.

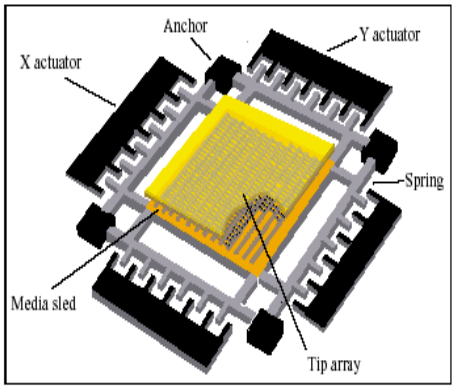

Figure 1: A design of MEMS-based storage devices

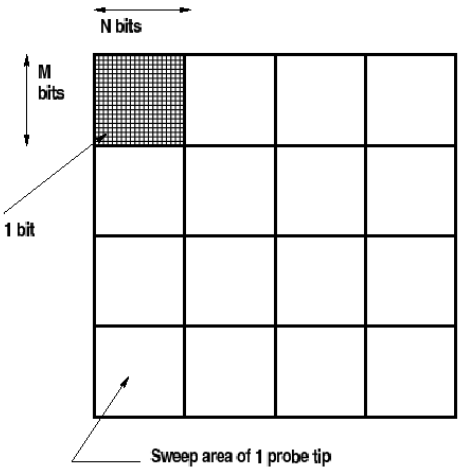

Figure 2: The media sled is divided into rectangular regions.

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 [7] uses pits in the polymers made by tip heating, CMU 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. Data is accessed by moving the media sled in X or Y directions over the stationary 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 the Figure 1 (It is derived from the CMU design [8]). The X and Y actuators provide the force for moving the media sled in the 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 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 CHIP device, the system has 6400 tips, arranged in an array of 80 x 80 tips per rectangular region with each region having 2500 x 2500 (M x N) bits. Due to power and heat constraints, only 1280 tips can be activated simultaneously.

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

(1) 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.

(2) 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_∞.

(3) 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. Based on observation (2), time T is the larger of these two values (time spent on the X (called T_x) and Y (called T_y) direction movement). For example, given points v and w, the time spent is T(v, w), where T (v, w) is a function of the L_∞ distance between v and w. Note that the time is symmetric, i.e. T(v, w) = T(w, v).

2.2 Existing scheduling algorithms for MEMS-based storage

Many different scheduling algorithms have been developed for conventional disks[6], 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 [1]. The CMU experimental results show that SPTF performs best in terms of average response time.

3. Theoretical Development.

This section describes the motivation to design a new scheduling algorithm for MEMS devices that is not adapted from disk-like devices. We first argue that finding an optimal scheduling algorithm for MEMS-based storage is NP-complete. Then we develop some properties of minimum spanning trees in the infinity distance domain. As described earlier, the scheduling performance in MEMS-based storage devices is a function of the infinity distance. Our scheduling algorithms are based on a minimum spanning tree in this domain.

3.1 Motivation.

MEMS-based storage characteristics are different from disks. Instead of plates rotating with a head moving back and forth, in MEM-based storage, the media sled can move in the X, Y and Z dimension. The seek time is dependent on the displacement in the X and Y dimension. Given these different parameters, MEMS-based storage devices require different request scheduling algorithms to fit in this environment. However finding the optimal solution is NP-complete.

In MEMS-based storage devices, for simplicity, a request can be denoted by vector (x, y), where x, y determine the position of the request in the active tip region. Because the tip's number is not related to the seek time, and scheduling algorithms find 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 to 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, a graph can be constructed. Requests can be denoted as vertices, edges are the time spent traveling from one vertex to another. The goal is to find a shortest path that visits each vertex exactly once in the graph. It is a Symmetric Traveling Salesman Problem, so finding the optimal path is NP-complete [3]. The symmetry arises from the fact that the traveling time from vertex v to vertex w is the same as the time traveling from w to v.

In the existing disk-based algorithms, SPTF performs best [1]. 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 R3 is the current request being served, because R2 is nearer to R3 than R4, then R2 is the next request to be served. Applying the same reasoning, the order of requests to be served is R3→R2→R4→R1→R5. 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 whole distribution of requests. In this example, if all requests are considered, the minimal order in terms of seek time would be R3→R2→R1→R4→R5.

Even though it is not practical to design an optimal algorithm, we develop an algorithm with guaranteed upper bound for any workload. We first introduce some basic concepts. A spanning tree of a graph is a cycle-free sub-graph that spans all the vertices. The cost of a spanning tree is the sum of the costs of edges in it. A minimum spanning tree (MST) is the smallest cost spanning tree of a graph, the cost for MST is referred to as T_{MST}. A double walk of a spanning tree means traversing all the vertices in pre-order, hence the cost of a double walk is equal to two times the cost of this spanning tree, as shown in Figure 4. The double walk is composed of all the arrow lines.

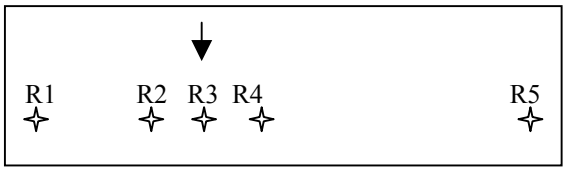

Figure 3: A setting that SPTF does not perform well

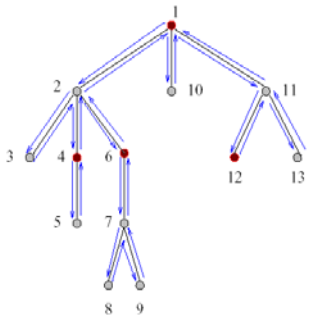

Figure 4: The double walk of a MST.

We propose a new scheduling algorithm based on serving requests in the order of the double walk of a minimum spanning tree. Because this algorithm is constructed on the minimum spanning tree (MST), before introducing the algorithms, we first present some properties of minimum spanning trees in the infinity distance space.

3.2 Properties of minimum spanning trees in infinity distance space

We start by defining region1 to region4 with respect to a vertex in the infinity distance space. Then we establish that the degree of any vertex in a MST is at most eight and can be reduced to four. It is interesting to note that in the Euclidian distance space, the bound on the degree is six and can be reduced to five [5]. In the following sections, without any specification, everything is in infinity distance space.

Consider a vertex is 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 $x_3 < x, y_3 \leq y$.

region4 is the subspace with any vertex (x_4, y_4) satisfying $x_4 \geq x, y_4 < y$.

All regions mentioned in the following refer to region1 to region4. We now establish several lemmas based on these regions.

Lemma 0: In any region of a vertex, there are at most two neighbors in the minimum spanning tree.

Proof: We first prove that a vertex v (x, y) in any region could have two neighbors. Then we prove that in each region, it is impossible for vertex v to have more than two neighbors.

Consider some region, say region1, and two neighbors v1 (x_1, y_1) and v2 (x_2, y_2). (A similar argument holds for any of the four regions.) There are four cases to consider depending on how v1 and v2 relate to a 45° line passing through v.

Case 1: Both v1 and v2 are on the 45° line, as shown in Figure 5.1. We have $x_1 = y_1$ and $x_2 = y_2$. Without loss of generality, assume $x_1 < x_2$. Then d(v, v1)<d(v, v2); d(v1, v2)<d(v, v2). Edge (v, v2) has to be replaced by (v1, v2) in the minimum spanning tree, so two neighbors of vertex v cannot be on the 45° line at the same time.

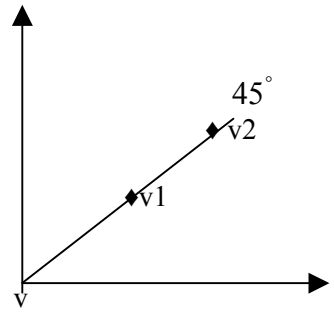

Figure 5.1

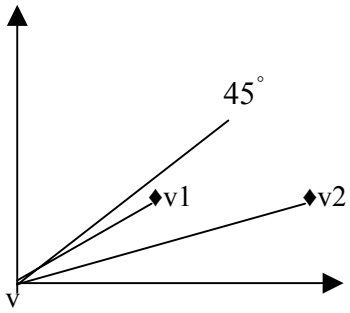

Figure 5.2

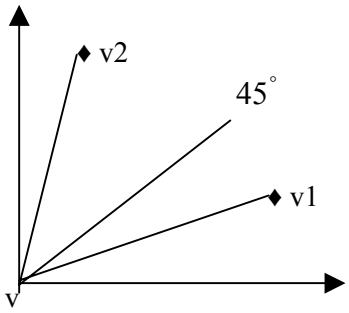

Figure 5.3

Case 2: Both v1 and v2 are either to the left or to the right of the 45° line. Without loss of generality, we analyze the case where v1 and v2 are on the right side of the 45° line, as shown in Figure 5.2. In this case, $y \le y_1$, $y \le y_2$, $x_1 > y_1, x_2 > y_2$. Thus d(v, v1) = $x_1 - x$; d(v, v2) = $x_2 - x$. Assume that $x < x_1 < x_2$ (the other case is symmetric). Since d(v1, v2) = $\max(|x_2 - x_1|, |y_2 - y_1|)$, there are four cases to consider.

If d(v1, v2)= $y_2 - y_1$, since $y \le y_1$, then $y_2 - y_1 \le y_2 - y$. However, since d(v, v2) = $x_2 - x$, it implies that $y_2 - y < x_2 - x$. Thus d(v1, v2) < d(v, v2).

If d(v1, v2)= $y_1 - y_2$, then since $y \le y_2$, we have $y_1 - y_2 \le y_1 - y$. However, since d(v, v1) = $x_1 - x$, it implies that $y_1 - y < x_1 - x$. Furthermore we know that $x_1 - x < x_2 - x$, since d(v, v1)<d(v, v2). Hence d(v1, v2) < d(v, v2).

If d(v1, v2) = $x_2 - x_1$, then since $x < x_1$, we get $x_2 - x_1 < x_2 - x$ = d(v, v2). Hence d(v1, v2) < d(v, v2).

If d(v1, v2)= $x_1 - x_2$, it is impossible since $x_1 < x_2$.

Thus swapping edge (v, v2) with (v1, v2) will reduce the cost of the spanning tree, i.e., both v1 and v2 connect to v, the spanning tree is not a MST.

Case 3: v1 and v2 are on different sides of the 45° line, as shown in Figure 5.3. In this case, we have $x_1 > y_1, x_2 < y_2, x < x_1, x < x_2, y \le y_1, y < y_2$. Thus d(v, v1) = $x_1 - x$; d(v, v2) = $y_2 - y$.

If d(v1, v2)= $y_2 - y_1$, because $y \le y_1$, then d(v1, v2)= $y_2 - y_1 \le y_2 - y$ =d(v, v2). The equality holds when $y = y_1$. So in this condition, both v1 and v2 can connect to v in a MST.

If d(v1, v2)= $y_1 - y_2$, because $y \le y_2$, then d(v1, v2)= $y_1 - y_2 \le y_1 - y < x_1 - x$ =d(v, v1).

If d(v1, v2)= $x_2 - x_1$, because $x < x_1$, then d(v1, v2)= $x_2 - x_1 < x_2 - x < y_2 - y$ =d(v, v2).

If d(v1, v2)= $x_1 - x_2$, because $x < x_2$, then d(v1, v2)= $x_1 - x_2 < x_1 - x$ =d(v, v1).

So in this case, v1 and v2 can connect to vertex v when $y = y_1$ and d(v1, v2)= $y_2 - y_1$.

Case 4: One of vertex v's neighbors is on the 45° line.

If one of them is on the 45° line and the other is on the right side of the 45° line. We have $x_1 > y_1, x_2 = y_2, x < x_1, x < x_2, y \le y_1, y < y_2$. Based on a proof similar to Case 3, we can show that v1 and v2 can connect to v if d(v, v2) = d(v1, v2).

If one of them is on the 45° line and the other is on the left side of the 45° line. We have $x_1 = y_1, x_2 < y_2, x < x_1, x < x_2, y < y_1, y < y_2$. Thus d(v, v1) = $y_1 - y$; d(v, v2) = $y_2 - y = x_2 - x$.

If d(v1, v2)= $y_2 - y_1$, because $y < y_1$, then d(v1, v2)= $y_2 - y_1 < y_2 - y$ =d(v, v2).

If d(v1, v2)= $y_1 - y_2$, because $y < y_2$, then d(v1, v2)= $y_1 - y_2 < y_1 - y$ =d(v, v1).

If d(v1, v2)= $x_2 - x_1$, because $x < x_1$, then d(v1, v2)= $x_2 - x_1 < x_2 - x$ =d(v, v2).

If d(v1, v2)= $x_1 - x_2$, because $x < x_1$, then d(v1, v2)= $x_1 - x_2 < x_1 - x < y_1 - y$ =d(v, v1).

So v1 and v2 cannot connect to v in a MST in this case.

We can conclude that vertex v in region1 can have two neighbors v1 (x_1, y_1) and v2 (x_2, y_2), where v1 is on the X axis, v2 is on the 45° line or on its left side, and d(v, v2) = d(v1, v2).
Without loss of generality, assume neighbor v3(x3, y3) is the furthest vertex among v1, v2, v3 to vertex v in region1. Based on the proof above, we have two cases.
Case 1: If vertex v1 is on the X axis and v2 is on the left side of the 45° line. It is impossible for vertex v3 to be on any side of the 45° line, the same reason as Case 2. If vertex v3 is on the 45° line, v2 and v3 cannot both connect to v based on Case 4. So it is impossible to add more neighbors.
Case 2: If one of v1 is on the X axis and v2 is on 45° line. If v3 is on the right side, it is the same situation as Case2; if v3 is on the 45° line, Case 1 can be applied; if v3 is on the left side of the 45° line, Case 4 can be applied. So under any condition, it is impossible for vertex v to have more than two neighbors in region1. □
Lemma 0 proves that in a minimum spanning tree, any vertex in any region can have at most two neighbors. We can easily derive Lemma 1 which state that the degree of any vertex is at most eight.

Lemma 1: The degree of any vertex in the MST is at most eight.
We now establish 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.

Lemma 2: There exists some MST, in which the degree of any vertex is at most four.
Proof: We claim that any MST with some vertices of degree larger than four can be transformed to a MST where no vertex has degree larger than four.
From Lemma 0, we know that in any of the four regions, at most two vertices may connect to a vertex in a MST. Without loss of generality, for a vertex v (x, y), its two neighbors in region1 are v1 (x_1, y_1) and v2 (x_2, y_2). From the proof of Lemma0, in all the possible positions of v1 and v2, d(v, v2) = d(v1, v2). By swapping edge(v, v2) with edge(v1, v2) would not increase the cost of MST, but the degree of vertex v is reduced to one in region1. The similar arguments can be achieved in region2 to region4.
By repeating this procedure in each vertex with more than four neighbors, we obtain a MST in which the degree of any vertex is at most four. □
We now explore ways to reduce the cost of traversing a MST.

Lemma 3: When traversing the MST, the cost of moving from one vertex to its sibling directly is no larger than the cost of passing through their parent vertex.
Proof: Because L_∞ satisfies the triangle inequality, vertex v, its siblings s and their parent p form a triangle, then the cost of moving from vertex v to s directly is no larger than the cost of moving from v to s by visiting p. □
Lemma 3 can be generalized as follows. The edge cost from one vertex to another vertex is less than or equal to the cost of the path between these two vertices in the MST.
In the following section, we develop scheduling algorithms based on these properties of MST in the infinity distance space.

4. Scheduling Algorithms

Our algorithm is based on building a minimum spanning tree of all requests and serving the requests in the tree order. An undirected graph (called cost graph) needs to be constructed in order to build a minimum spanning tree. In the cost graph, requests are treated as vertices, the edge cost from one request R_i to another request R_j is the seek time $T_{i,j}$. For MEMS-based storage, the following information is known:

1. Requests and their positions (x, y).

2. The equation to compute the seek time between two requests $R_i(x_i, y_i)$ and $R_j(x_j, y_j)$: $T_{i,j} = f(\max(|x_i - x_j|, |y_i - y_j|))$, which is the L_∞ distance between the two requests.

Since in a MEMS-based storage device, 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 $n(n-1)/2$, where n is the number of vertices. It is very inefficient to construct a MST based on a graph with $n(n-1)/2$ edges, because the time complexity of constructing MST is dependent on the number of edges and vertices. Both Prim's and Kruskal's algorithms for constructing MST have time complexity $O(m \log_2 n)$, where m is the number of edges. However, the number of edges in the cost graph can be reduced to at most 8n. From Lemma 2, the degree of each vertex in a 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. Thus the cost graph only needs to include all edges that are formed by connecting each vertex to its nearest neighbors in each region.

4.1 The cost graph

Based on Lemma 2, to build the cost graph for a given set of requests, we need to find at most four connecting vertices of every vertex which are distributed in region1 to region4 respectively. We now describe the method for finding the four connecting vertices for a given vertex $R_i(x_i, y_i)$.

Assume they are n vertices (or requests): $R_1(x_1, y_1), R_2(x_{2,} y_2), R_3(x_3, y_3), \cdots, R_n(x_n, y_n)$.

Two vectors are constructed: X-VECTOR and Y-VECTOR.

X-VECTOR contains the X dimension values in increasing order. Y-VECTOR contains the Y dimension values in increasing order.

With respect to $R_i(x_i, y_i)$, the infinity distance space is divided into region1 to region4. From another point of view, X-VECTOR is divided into two sub-vectors, X+ and X-, by x_i, Y-VECTOR is divided into two sub-vectors, Y+ and Y-, by y_i. The next task is to find the vertex nearest to $R_i(x_i, y_i)$ in each region.

We describe the procedure for finding the nearest vertex in the region1, which is equivalent to searching the X+ and Y+ sets. Assume the vertex nearest to $R_i(x_i, y_i)$ is $R_j(x_j, y_j)$.

Step 1: There are two pointers, X_Pointer and Y_Pointer, pointing to the current positions of X+ and Y+. Initially, they point to the first element in the X+ and Y+.

x is the value of the element pointed to by X_Pointer; y is the value of the element pointed to by Y_Pointer.

Step 2: If $R(x, y)$ is a vertex pair in $R_1(x_1, y_1), R_2(x_{2,} y_2), R_3(x_3, y_3), \cdots, R_n(x_n, y_n)$, then it is the vertex we are searching for. Connect $R_i(x_i, y_i)$ with $R(x, y)$.

Step 3: If $|x_i - x| \geq |y_i - y|$, move Y_Pointer to the next element in Y+,

$y =$ the value of element pointed by Y_Pointer, goto Step 2.

Step 4: Else $|x_i - x| < |y_i - y|$, move X_Pointer to the next element in X+,

$x =$ the value of element pointed by X_Pointer, gotoStep 2.

The procedure for finding the nearest vertices in the region 2 to region4 is similiar. The cost graph is built by repeating this procedure for all vertices. Since each vertex has four neighbors, in the cost graph, each vertex has between 4 and 8 neighbors. The resulting cost graph has all the necessary edges to build an MST.

4.2 Off-line scheduling Algorithm

Based on the theoretical development described above, we now present a scheduling algorithm for the case when all the requests are known a priori. In the off-line algorithm, the cost graph is built by the method mentioned above, and the MST is built by Prim's Algorithm. The requests are served in a pre-order traversal of the MST. The algorithm is composed of one main subroutine called Main Method and a function subroutine called Find_next.

Main Method:

1. Given n requests: $R_1(x_1, y_1), R_2(x_2, y_2), \cdots, R_i(x_i, y_i), \cdots, R_n(x_n, y_n)$;
2. Build the cost graph G for the n vertices (or requests);
3. Based on G, build the Minimum spanning tree MST for the n requests;
4. number_request_left = n;
5. next_request = root in the MST; Serve(next_request);
6. **While** (next_request != null) **do**
7. number_request_left = number_request_left – 1;
8. current_position = (x, y), where x, y are displacement of next_request;
9. next_request = find_next(next_request); Serve(next_request);
10. **END**.

Find_next (current_request)

1. **If** number_request_left = 0, **return** null;
2. **If** current_request has no children then save_parent = parent of current_request; Remove current_request from MST, and
3. **return** find_next(save_parent);
4. **else** choose minimum cost child of current_request, label it served,
5. **return** the minimum cost child.
6. **END.**

The variable number_request_left is used to record the number of requests that have yet not been served. The subroutine find_next (current_request) is used to find next request to serve. It also finds and returns the next request to serve. The main idea of finding the next request is primarily based on the tree preorder traversal. Even though as the algorithm is finding the next request, it traverses the tree, but the real distance moved is the distance between the current_request and the next_request based on Lemma 3. This algorithm starts from the root. After it is served, find one of its nearest children as the next request and label it as 'SERVED'. Otherwise recursively call find_next(parent of current_position) until find the next_request.

The overall cost of this algorithm is dominated by building the cost graph, which is $O(n^2)$. Because the cost of constructing a minimum spanning tree is $O(m \log_2 n)$ and m is at most 8n, the cost is reduced to $O(n \log_2 n)$.

4.3 On-line scheduling Algorithm

In this section, the algorithm for the on-line scheduling is presented where requests are continually arriving as prior requests are being served. The algorithm is different from the off-line one. When new requests arrive during the service of existing requests, the MST has to be updated to include the new requests. Current dynamic MST algorithms [4] only consider changes in the cost of edges and not the inclusion of new vertices.

Our on-line algorithm is composed of two parts. One serves existing requests in the current MST, the other updates the current MST with new incoming requests. The first part is the same as the off-line algorithm, so we primarily focus on describing the algorithm for updating the current MST. In the algorithm for updating the MST, according to the Lemma 2, the four nearest vertices in the different regions, if they exist, are found. The new vertex is connected to the MST by the smallest cost edge to one of these four vertices. The remaining three edges are checked to see if the MST cost can be reduced.

Update the MST with the new coming request $R_{new}(x_{new}, y_{new})$.

1. Insert x_{new} into the X-VECTOR, y_{new} into Y-VECTOR;
2. Search the requests which are nearest to the new request in region1 to region4 with respect to R_{new}. The procedure is the same as computing each vertex's neighbors in building up the cost graph. At most four requests exist, say R_1, R_2, R_3, R_4. Connect R_{new} to the nearest vertex in R_1, R_2, R_3, R_4. So that R_{new} is included in the MST.
3. After step 2, there are at most three edges left, if they are all larger than the max cost edge in the MST, DONE.
4. **for** any edge e of the three edges left with cost less than the max cost edge in the MST **do**
5. Insert e into the MST, it results in a cycle, remove the maximum cost edge from the cycle.
6. **end.**

Based on the Lemma 2, every vertex in MST connects to at most four other vertices, these four vertices are located in the region1, 2, 3, 4 respectively. Step 1 and 2 basically locate these four vertices if they exist. Because these four vertices are the nearest vertices to vertex v**,** connect v to the MST by the nearest neighbor in these four neighbors. The following steps are used for updating the current MST, because inserting new vertices may reduce the total cost of the MST. Step 3 is an optimization. If the remaining three edges are larger than the maximum cost edge in the MST, none of them will be in the MST. Step 4 and 5 update the MST. There must be a cycle c if an edge is added, according to the tree property of MST, remove the largest edge in this cycle.
The cost of updating the MST is dominated by finding cycles and finding four nearest vertices, both of them are $O(n)$, so the total cost of this algorithm is $O(n)$.

5. Analysis.

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

5.1 Upper bound.

Without the optimizations made by Lemma 3 & 4, our approach corresponds to a double walk of a minimum spanning tree. The cost of a double walk is equal to two times the cost of this spanning tree, as shown in Figure 4. The double walk is composed of all the arrow lines. If the spanning tree is a MST, the cost of a doubling 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 results can be obtained.

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

Hence $T_{seek} \le 2 * T_{opt}$, where T_{seek} is the total seek time for serving all requests.
Our approach made some optimizations based on Lemma 2 & 3. So we have the following equation:

$$T_{seek} < 2 * T_{opt}.$$

Hence, our approach is guaranteed has an upper bound of $2*T_{opt}$ irrespective of the workload.

5.2 Preliminary performance.

We set up the experiments by simulating 100×100 region with uniformly distributed workloads. We implemented four scheduling algorithms: FCFS, SSTF, SPTF and Tree-based 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 6.

The data shows that SPTF and Tree-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 are not scalable. However the average seek distance of SPTF and Tree-based approach decrease. When the workload is not heavy, the Tree-based approach achieves better average seek distance than SPTF. Under heavy workload, their performance is comparable.

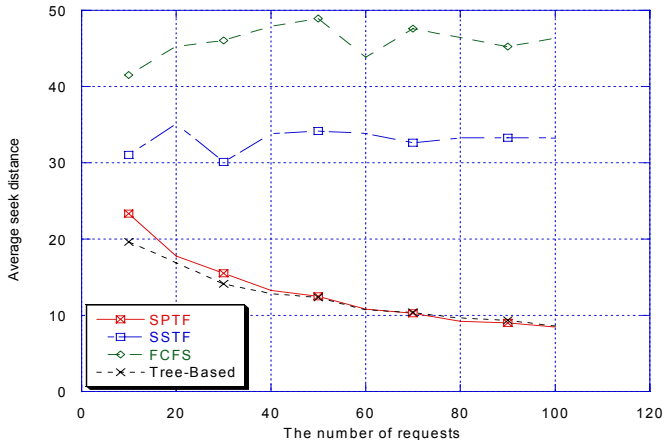

Figure 6: Average seek distance.

6. Conclusion.

In this paper, based on the characteristics of MEMS-based storage devices, we developed two dimensional scheduling algorithms which guarantee the upper bound $2T_{OPT}$ on seek time. The cost of serving a request is $O(n)$. The cost of updating the MST is also $O(n)$. Based on the algorithm designed by Christofides [10], we could achieve an upper bound of $1.5T_{OPT}$, but the cost for the online case would be more expensive. Our future research will explore tradeoffs among these different approaches.

References:

[1] Griffin, J., Schlosser, S., Ganger, G., Nagle, D., Operating Systems Management of MEMS-based Storage Devices. In OSDI 2000, October 23-25, 2000.

[2] L. Richard Carley, Gregory R. Ganger, and David F. Nagle, MEMS-Based Integrated-Circuit Mass-Storage Systems. in COMMUNICATIONS OF THE ACM November 2000, Vol.43, No.11.

[3] M. Andrews, M. A. Bender, and L. Zhang. New algorithms for the disk scheduling problem. In Proc. 37th IEEE Sympos. Found. Comp. Sci., pp. 580 589, Oct. 1996.

[4] David Eppstein. Offline algorithms for dynamic minimum spanning tree problems. Journal of Algorithms, 17(2):237-250, September 1994.

[5] C. Monma and S. Suri, "Transitions in Geometric Minimum Spanning Trees," Discrete & Computational Geometry, Vol. 8, No. 3, pp. 265--293, 1992.

[6] Seltzer, M., Chen, P. and Ousterhout, J. Disk Scheduling Revisited. In Proceedings USENIX of the Winter 1999 Conference.

[7] P.Vettiger, M.Despont, U.Drechsler, U.Durig, W.Haberle, M.I. Lutwyche, H.E.Rothuizen, R.Stutz, R.Widmer, and G.K.Binnig. The “Millipede”- More than one thousand tips for future AFM storage. IBM Journal of Research and Development, 44(3):323-340, May 2000.

[8] Schlosser, S., Griffin, J., Nagle, D., Ganger, G. Designing Computer Systems with MEMS-based Storage. In ASPLOS 2000, November 13-15, 2000.

[9] CMU CHIP project: http://www.ece.cmu.edu/research/chips.

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