Efficient Disk Replacement and Data Migration Algorithms for Large Disk Subsystems

Roger Zimmermann and Beomjoo Seo
Integrated Media Systems Center
University of Southern California
Los Angeles, CA, U.S.A.

Abstract

Random data placement, efficient and scalable for large-scale storage systems, has recently emerged as an alternative to traditional data striping. In this study we address Disk Replacement Problem (DRP) of finding a sequence of disk additions and removals for a storage system while migrating the data and respecting the following constraints: (1) the data is initially balanced across the existing distributed disk configuration, (2) the data must again be balanced across the new configuration, and (3) the data migration cost must be minimized. In practice, migrating data from old disks to new devices is complicated by the fact that the total number of disks connected to the storage system is often limited by a fixed number of available slots and not all the old and new disks can be connected at the same time. We present solutions for both cases, where the number of disk slots is either unconstrained or constrained.

Keywords: Disk Replacement, Data Migration, Randomized Striping,

1 Introduction

To achieve high I/O performance with disk arrays the access load should be evenly balanced across all the devices. Most existing data placement techniques use some form of striping to decluster the data across the storage system. However, these conventional techniques lack the flexibility of easy hardware reconfiguration that is necessary with today's very large storage systems. Specifically, data placement schemes such as RAID often do not allow the efficient addition or removal of disks and hence are a hindrance to the scalability of a system. With striping almost every data item must be relocated if the number of disks is changed. Some of these problems can be somewhat alleviated with techniques such as spare disks or multiple, small RAID partitions.

The random distribution has great advantages when data needs to be reorganized: blocks to migrate can randomly be chosen and moved, resulting in a new random distribution after the reorganization. For example, the RIO multimedia object server demonstrated the applicability of random data placement for media-rich storage systems [1]. Next to flexibility and reliability, performance is often of paramount importance with storage systems. When data blocks are randomly distributed across all the storage devices, an efficient directory service is necessary to locate individual blocks. It has been shown that random data placement can achieve similar performance to traditional striping techniques [2].

On–line scalability is also closely related to a problem referred to as *data migration*. The general problem that a data migration algorithm addresses is to re-balance a storage system after it has become unbalanced. Data migration works in two steps to (a) compute a new, load-balanced data distribution of all the currently existing data items (or blocks) in the system, and (b) efficiently move all the data blocks from their previous placement to their new locations.

A number of studies have investigated data migration algorithms [3], [4], [5], [6]. Finding a migration plan can be mapped to a multi-graph edge-coloring problem to compute the minimum number of colors needed. Unfortunately, finding the optimal solution is either $\mathcal{NP}$-complete [3] or $\mathcal{NP}$-hard [6]; hence several approximation algorithms have been proposed in the literature. The data migration problem is very general in that it allows any data item to move from any device to any other device. In this paper, we address a problem that is slightly more constrained, but still covers many practical cases and very importantly allows an optimal solution to be computed in polynomial time.

In this study we generalize the example given above and term it the *disk replacement problem* (DRP), which describes the challenge of finding a sequence of disk drives removals and additions to obtain a final, target storage system while minimizing the data migration cost.

The contributions of this paper are twofold. First, we provide the mathematical models for DRP to find the optimal sequence of add/delete/replace operations. Second, we investigate a variation of the basic disk replacement problem and provide algorithmic results when the number of disk slots is constrained.

2 Disk Replacement Problem

Table 1 summarizes the symbolic notations used throughout this paper.

DRP should satisfy the following two pre-requisites to guarantee that the final configuration is load balanced and well randomized with high probability.

property 1: After every atomic operation the data is balanced across the devices involved, i.e., the space is uniformly utilized.

property 2: After every atomic operation all the block locations are well randomized, i.e., the workload imposed on every device is approximately equal.

Consider a *weighted direct acyclic graph* $G_c = (V, E)$, where each vertex v in V represents a tuple

This research has been funded in part by equipment gifts from Intel and Hewlett-Packard, unrestricted cash grants from the Lord Foundation and by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152.

TABLE 1. Symbolic notations used

Symbol	Meaning
R	disk bandwidth (read or write)
S	total amount of data stored in current system
C	set of disk slots. $c = \|C\|$
N	set of disks in use at current configuration. $n = \|N\|$
A	set of disks to add $A' \subset A, a = \|A\|, a' = \|A'\|$
D	set of disks to remove $D' \subset D, d = \|D\|, d' = \|D'\|$
$r(n, d, a, c)$	concise representation of disk scaling operation: start with n disks, then add a disks and delete d disks while using no more than c disk slots.
s	amount of data in each disk before scaling operation
s'	amount of data in each disk after scaling operation
A_n^a	a disk addition in a given n disks
D_n^d	d disk deletion in a given n disks
$\omega(r)$	cost of disk scaling sequence r (ω_s : space cost, ω_τ : time cost)

of four non-negative integer variables (n, d, a, c) – that means current number of disks in use is n and the next operation will remove d old disks and add a new disks. In any real storage system a constraint exists on how many disks can be attached. We refer to this maximum as the number of *slots* and denote it with c. For an unconstrained graph $c = \infty$, we omit the forth parameter in our notation, e.g., $(n, d, a, \infty) \equiv (n, d, a)$. The edge $(v_i(n_i, d_i, a_i, c),\ v_j(n_j, d_j, a_j, c)),\ v_i, v_j \in V$, represents the replacement operation that removes $(d_i - d_j)$ old disks and adds $(a_i - a_j)$ new disks. Each edge $e(v_i, v_j)$ is labelled with a non-negative weight $\omega(e)$, $\omega : E \rightarrow R^+$ that represents the data migration cost from v_i to v_j.(See Section 3). The weight of a complete path $p = \langle v_0, v_1, \cdots, v_k \rangle$ is the sum of the weights of its constituent edges. Consequently, the disk replacement problem can be formalized as finding the shortest path from the initial configuration $v_0(n_0, d_0, a_0, c)$ to the final configuration $v_t(n_t = n_0 - d_0 + a_0, d_t = 0, a_t = 0, c)$, possibly with a given constraint c that denotes the maximum number of disk slots which cannot be exceeded at any step of the algorithm.

If a vertex (n, d, a, c), termed *disk scaling request (state, condition)*, satisfies $n \leq c, (n - d + a) \leq c$, and $d \leq n$, it is said to be *valid*; otherwise, it is considered *invalid*. If a valid vertex satisfies $(n + a) \leq c$, it is said to be *unbounded*; otherwise, it is *bounded*. If an edge (v_i, v_j), termed *disk scaling operation (transition)*, which is incident to a valid pair of vertices satisfies $d_i \geq d_j, a_i \geq a_j, (d_i \neq d_j) \vee (a_i \neq a_j)$, and $(n_i - d_i + a_i) = (n_j - d_j + a_j)$, it is said to be *valid*; otherwise, it is *invalid*. If a valid edge satisfies $(n_i + a_i - a_j) \leq c$, it is *unbounded*; otherwise, it is *bounded*. A DRP graph G_c consisting of valid vertices and valid edges becomes directed and acyclic.

The running time of finding the single-pair shortest path in a given Direct Acyclic Graph (DAG) – based on topological sort – is known to be $O(V + E)$ (see [7]). Given the initial source $v_0(n_0, d_0, a_0, c)$, the maximum number of possible vertices is $O((a_0 + 1)(d_0 + 1)) = O(a_0 d_0)$. Thus, the running time is bounded by $O({a_0}^2 {d_0}^2)$ because $O(V+E) = O(V+V^2) = O(V^2)$. Hence, constrained or unconstrained DRP can be solved in polynomial time.

The complexity of building a DAG from a given source and destination is also dominated by $O(V^2) = O(E)$, which has the same order of magnitude of time complexity as the solution algorithm.

To simplify our further discussions we introduce the notation of an aggregation of a sequence of operations, $r(n, d, a, c)$. The aggregation r represents a complete sequence of d disk deletions and a disk additions starting with the current n disks:

$$r(n, d, a, c) \stackrel{def}{=} e(v_0(n, d_0, a_0, c), v_1(n - d + a, d_0 - d, a_0 - a, c)) \text{ where } \exists \omega(e(v_0, v_1)) < \infty \text{ in } G_c$$

We further introduce a simplified notation for single disk additions: $A_n^1 = A_n \equiv r(n, 0, 1, c)$. Consequently, we represent multiple *simultaneous* disk additions as A_n^a. Note however that A_n^a is not equivalent to $r(n, 0, a, c)$ because $r(n, 0, a, c)$ can be achieved through a number of different paths: $\langle A_n^1\ A_{n+1}^1\ A_{n+2}^1\ A_{n+3}^1\ \cdots\ A_{n+a-1}^1 \rangle$ or $\langle A_n^2 A_{n+2}^1 \cdots A_{n+a-3}^3 \rangle$, etc., or finally $\langle A_n^a \rangle$. A_n^a strictly represents the simultaneous addition of a disks. Analogous, we define a single disk deletion as $D_n \equiv r(n, 1, 0, c)$ and multiple disks deletions at a time as D_n^d.

3 Cost Models

We consider two types of data migration costs. The first one is based on the total amount of data being moved during the disk scaling operation and termed *space cost*, while the second one computes the total elapsed time and is correspondingly called *time cost*. Both cost models are based on the invariant that the total amount of data S stored in the initial disk configuration is unchanged after the disk scaling operation.(see Equation 1)

$$S = ns = (n - d + a)s' \tag{1}$$

3.1 Space Cost

The space cost ω_s is the amount of data moved during a disk scaling operation.

$$\omega_s(A_n^a) \models \frac{a}{n + a} S \tag{2}$$

Equation 2 quantifies the space cost of simultaneously adding a disk to the current n-disk configuration. The amount of data moved between the initial and the final state is as'. Substituting from Equation 1 with $d = 0$, we observe that the total amount of data moved is $as' = a(\frac{1}{n+a})S$. Analogous, Equation 3 quantifies the space cost of d simultaneous disk deletions from an n disk configuration. All data of D disks is moved to

$(C - D)$ disks, resulting in ds data being moved, i.e., $ds = d(\frac{S}{n})$.

$$\omega_s(D_n^d) \models \frac{d}{n}S \quad (3)$$

3.2 Time Cost

The total elapsed time can be expressed with three independent time variables t_{RD}, t_d, and t_{WR}. t_{RD} is the read time to load data from storage devices into the memory. Similarly, t_{WR} denotes the time of writing data from memory into the storage devices. Finally, t_d denotes the delay of moving data through a network. Hence, the edge weights are expressed as follows:

$$\omega_\tau(A_n^a) \text{ or } \omega_\tau(D_n^d) \models max(t_{RD} + t_d, t_{WR} + t_d) \quad (4)$$

In case of disk additions, the new disks are installed into empty disk slots and the data is moved from existing disks to the newly installed devices. Consequently, $(s - s')$ amount of data will be retrieved from N during t_{RD}. As soon as it is available in the system memory, the data will be transmitted over the network, requiring t_d amount of time. At the receiver side, the transferred data s' will be stored into the newly added disks. Therefore, the total elapsed time is the maximum of either the retrieval or the storage time. Similarly, in case of disk deletions, data is read from D disks and written to $(N - D)$ disks.

To simplify our model we assume that the disks are *fully connected and the network transmission time is negligible* and the network and system bus bandwidth do not present a bottleneck and that the reading and writing data rates of the disk devices is the same, i.e., $R = R_{RD} = R_{WR}$.

By adding a disks to the current disk configuration n, Equation 4 simplifies to $max(\frac{s-s'}{R}, \frac{s'}{R}) = max(\frac{aS}{n(n+a)R}, \frac{S}{(n+a)R})$. Thus, Equation 4 can be rewritten as follows.

$$\omega_\tau(A_n^a) = \frac{1}{min(n,a)R}\omega(A_n^a) \quad (5)$$

Intuitively, the elapsed time is primarily affected by the current number of disks or by the number of disk additions with the same space cost. The cost model for disk deletions is derived analogously. Data is read from D and transmitted to $(N - D)$.

$$\omega_\tau(D_n^d) = \frac{1}{(n + d - max(n, 2d))R}\omega(D_n^d) \quad (6)$$

4 Unconstrained DRP

We first consider a variation of the disk replacement problem where the number of disk slots in the system is not bounded, i.e., $r(n, d, a, \infty)$.

4.1 Non-Overlapping Approach - ABBD (Add–Balance–Balance–Delete)

If a disk scaling sequence consists of two non-overlapping disk scaling operations, disk additions and disk removals, the optimal sequence for minimizing both the space and the time cost is to add all the new disks first (including re-balancing the data) and then remove all the old disks to be deleted (including first re-balancing the data). We present the Lemmas 1 to 8 to outline our argument (detailed proofs are shown in [8]).

Lemma *1:* $\omega_s(A_n^{i+j}) \leq \omega_s(A_n^i A_{n+i}^j)$.

Lemma *2:* $\omega_s(D_n^{i+j}) \leq \omega_s(D_n^i D_{n-i}^j)$.

Lemma *3:* $\omega_s(A_n^i D_{n+i}^j) \leq \omega_s(D_n^j A_{n-j}^i)$.

Lemma *4:* The shortest path of $\omega_s(r(n, d, a))$ is $A_n^a D_{n+a}^d$.

Lemma *5:* $\omega_\tau(A_n^{i+j}) \leq \omega_\tau(A_n^i A_{n+i}^j)$.

Lemma *6:* $\omega_\tau(D_n^{i+j}) \leq \omega_\tau(D_n^i D_{n-i}^j)$.

Lemma *7:* $\omega_\tau(A_n^i D_{n+i}^j) \leq \omega_\tau(D_n^j A_{n-j}^i)$.

Lemma *8:* The shortest path of $\omega_\tau(r(n, d, a))$ is $A_n^a D_{n+a}^d$.

These lemmas suggest two additional findings, which we will use in several heuristics described in later sections:

1. *Disk additions are preferable to disk deletions.*
2. *Add as many disks as possible.*

The above observations result in algorithm $ABBD$ shown below, which is the algorithmic implementation of Lemmas 4 and 8.

Algorithm 1 $ABBD(N, D, A)$

1: add A
2: distribute the data evenly from N to $(N + A)$
3: distribute all data evenly from D to $(N - D + A)$
4: remove D from N

4.2 Merged Approach - ABD (Add–Balance–Delete)

Lines 2 and 3 of the $ABBD$ algorithm describe two non-overlapping data distribution operations, which result in wasteful data movement because some data stored on disk set D are moved to $(N-D)$ via A. Algorithm ABD is an enhancement of $ABBD$ and merges the two re-balance operations into one. Lines 3 through 7 of algorithm ABD disseminate the data probabilistically. Hence, ABD provides the optimal space cost for the unconstrained DRP. Lemma 9 shows the time cost of ABD (see detailed proof in [8]).

Lemma *9:* If $a > d$, then the time cost of algorithm ABD for $r(n, d, a)$ is $\frac{1}{n}\frac{S}{R}$, otherwise it is $\frac{1}{n-d+a}\frac{S}{R}$.

5 Constrained DRP

We will address a more realistic disk replacement problem, $r(n, d, a, c)$, where the number of disk slots in the system is limited to c while the current number of disks plus the new disks $(n + a)$ exceed c.

The algorithms that we present here are based on the following observation. If the number of current disks plus the new disks to be added exceed the maximum number of available disk slots, $(n + a) > c$, then we may break the problem into two sub-parts: adding up to $a' \leq (c - n)$ disks first and then considering the remaining problem of adding $(a - a')$ disks. Hence, we introduce a *divide and conquer* algorithm to find optimal path sequence.

Algorithm 2 $ABD(N, D, A)$

1: $a \leftarrow |A|$, $d \leftarrow |D|$, $n \leftarrow |N|$
2: install a disks
3: **if** $(a \geq d)$ **then**
4: move all data in D to A uniformly, and move $\frac{a-d}{n-d+a}$ faction of data from each disk in $(N-D)$ to A
5: **else**
6: move $\frac{(n-d)(d-a)}{d(n-d+a)}$ amount of data to $(N-D)$, and move $\frac{an}{d(n-d+a)}$ amount of data to A from each disk in D
7: **end if**
8: remove d disks

5.1 Divide-and-Conquer Approach (D&C)

A bounded disk scaling request can be divided into single unbounded disk scaling transition – using ABD for its solution – and its remaining disk scaling operation, because generally there is no direct transition from the bounded disk scaling request to the final termination state. This leads to a recursive algorithm where the subsequent disk scaling requests can be either bounded or unbounded. If a subsequent disk scaling request is unbounded, it has a direct transition sequence to reach the final disk configuration (using ABD). Otherwise, the problem is further subdivided until it resolves to an unbounded state.

Equation 7 formalizes the $D\&C$ algorithm. It can be viewed as a constrained DRP graph. Thus, its search cost is same as that of the shortest path algorithm. Note that both a' and d' in the equation cannot be zero at the same time. This solution finds both the optimal time and space cost with a constraint.

$$\omega(r(n,d,a,c)) =$$

$$\begin{cases} \omega(ABD(n,d,a)) & : c \geq (n+a) \\ \underset{\substack{0 \leq d' \leq d,\\ 0 \leq a' \leq (c-n)}}{\text{MIN}} \begin{pmatrix} \omega(ABD(n,d',a')+ \\ \omega(r(n-d'+a', d-d', a-a', c)) \end{pmatrix} & : \text{if bounded} \end{cases} \quad (7)$$

Memoization Technique The $D\&C$ algorithm may suffer from increased execution time because it recomputes the same subproblems multiple times. The well-known solution to avoid redundant computations is the memoization technique which stores intermediate results for future use [7].

5.2 Space Cost Minimization

Space cost minimization may be useful when the amount of data moved is crucial. We presently assume that the network or storage bus transmission time is negligible compared with the disk read and write times. If this assumption does not hold for a specific application (i.e., the data must be moved through a slow network), then minimizing the amount of data moved is desirable. We first suggest a slightly enhanced $D\&C$ algorithm termed EB and then introduce a linear solution with no memoization overhead.

Algorithm 3 $EB(N, D, A, C)$

1: $D'' \leftarrow D, A'' \leftarrow A$
2: **if** $|N| < |C|$ **then**
3: **if** $|A| \geq |D|$ **then**
4: **repeat**
5: $D' \leftarrow$ select $min(|D''|, |C| - |N|)$ number of disks from D''
6: $A' \leftarrow$ select $|D'|$ number of disks from A''
7: install A' to empty disk slots
8: move all data from D' to A' respectively
9: remove D' from disk slots
10: $D'' \leftarrow D'' - D'$
11: $A'' \leftarrow A'' - A'$
12: **until** $D'' = \emptyset$
13: $ABD(N, \emptyset, A'')$
14: **else**
15: **repeat**
16: $A' \leftarrow$ select $min(|A''|, |C| - |N|)$ number of disks from A''
17: $D' \leftarrow$ select $|A'|$ number of disks from D''
18: install A' to empty disk slots
19: move all the data from D' to A' respectively
20: remove D' from disk slots
21: $D'' \leftarrow D'' - D'$
22: $A'' \leftarrow A'' - A'$
23: **until** $A'' = \emptyset$
24: $ABD(N, D'', \emptyset)$
25: **end if**
26: **end if**

5.2.1 Exchange–Balance (EB)

The EB algorithm solves a bounded scaling request directly, which results in a smaller search space than with $D\&C$. Line 2 of algorithm EB examines whether the current disk scaling request includes at least one empty disk slot, regardless of its bounded or unbounded condition. If $a > d$, it adds as many disks A' as possible into the empty disk slots, selects $|A'|$ disks from the set D, copies their data to A', and removes the $|A'|$ selected disks. The complete operation is repeated until all D disks are removed. After the iterative exchange procedure, it installs the remaining new disks. When $a \leq d$ the procedure resembles that of the $a > d$ case. When $a \leq d$ the space cost computed by EB is exactly same as that of ABD. However, this algorithm is not applicable for minimizing the time cost.

As an extension, the algorithm $D\&C$ + EB uses the EB procedure only when the system has at least one empty slot and the number of new disks is smaller than or equal to the number of disks to be deleted. Otherwise, $D\&C$ + EB executes equivalently to $D\&C$.

5.2.2 Linear Heuristic Algorithm (SPACE)

The $SPACE$ algorithm is a linear solution for finding the minimum space cost for bounded disk scaling requests. Its computation time depends only on the number of disks to be removed, regardless of the constraint and current disk configuration. The $SPACE$ heuristic has its origins in the findings of Section 4.1: to add as many disks as possible to the remaining disk slots. This

Algorithm 4 $SPACE(N, D, A, C)$

1: $n \leftarrow |N|, d \leftarrow |D|, a \leftarrow |A|, e \leftarrow |C| - |N|$
2: **if** $(n - d + a) > |C|$ **then**
3: return ∞
4: **else if** $e \geq a$ **then**
5: return $\omega_s(ABD(N, D, A))$
6: **end if**
7: **if** $e = 0 \wedge d \geq a$ **then**
8: $D' \leftarrow$ select a disk from D
9: return $\omega_s(ABBD(N, D', \emptyset))$ + $\omega_s(EB(N - D', D - D', A, C))$
10: **else if** $d \geq a$ **then**
11: return $\omega_s(EB(N, D, A, C))$
12: **end if**
13: **if** $e > 0 \wedge a > d$ **then**
14: **if** $(a - d) = e \wedge a \neq d + 1$ **then**
15: $D' \leftarrow \emptyset$
16: $A' \leftarrow$ select $(a - d - 1)$ disks from A
17: **else if** $(a - d) < e$ **then**
18: $D' \leftarrow \emptyset$
19: $A' \leftarrow$ select $(a - d)$ disks from A
20: **end if**
21: **else**
22: $D' \leftarrow$ select a disk from D
23: $A' \leftarrow$ select e disks from A
24: **end if**
25: return $ABD(N, D', A')$ + $SPACE(N - D' + A', D - D', A - A', C)$

is illustrated in lines 16, 19, and 23 of the algorithm. Additionally, $SPACE$ attempts to perform disk additions before deletions. The complexity of the $SPACE$ algorithm primarily depends on the last line, which is executed at most D times. Therefore, the complexity of $SPACE(N, D, A, C)$ is $O(|D|)$.

5.3 Time Cost Minimization

The $D\&C$ algorithm is also applied to find an optimal time cost solution for bounded disk scaling requests. Its complexity is unchanged and equal to the $D\&C$ space cost algorithm.

5.3.1 Simple Heuristic Algorithm (TIME)

The $TIME$ algorithm selects a subset A' of the disks to be added A when attempting to divide a disk scaling request into an unbounded transition and its remainder. The set A' is chosen to be equal to the number of empty disk slots. As a result, the search space is halved. Furthermore, all possible state transitions are searched by varying the number of old disks to delete from 0 up to d disks. We call this search variable *pivot*.

The complexity of $TIME$ is determined by the number of times line 9 is executed. The memoization technique can be easily applied by checking for stored results before line 9 and memorizing sub-solutions after line 14.

5.3.2 Local Minima Algorithm (LMIN)

The $TIME$ algorithm examines all the possible result states after an unbounded transition by varying the pivot

Algorithm 5 $TIME(N, D, A, C)$

1: $n \leftarrow |N|, d \leftarrow |D|, a \leftarrow |A|, e \leftarrow |C| - n$
2: **if** $(n - d + a) > |C|$ **then**
3: return ∞
4: **else if** $e \geq a$ **then**
5: return $\omega_\tau(ABD(N, D, A))$
6: **end if**
7: $min \leftarrow \infty$
8: $A' \leftarrow$ select e disks from A
9: **for** $\forall_{(D' \subset D) \wedge (D' \neq \emptyset \wedge A' \neq \emptyset)}$ **do**
10: $cost \leftarrow \omega_\tau(ABD(N, D', A'))$+ $TIME(N - D' + A', D - D', A - A', C)$
11: **if** $cost < min$ **then**
12: $min \leftarrow cost$
13: **end if**
14: **end for**
15: return min

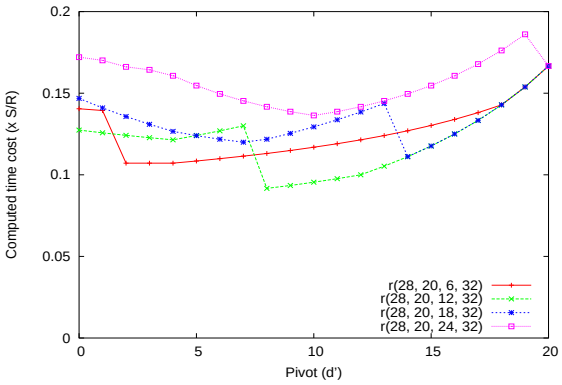

Fig. 1. Multiple bowl-shaped time cost curves as a function of pivot.

value ranging from 0 to d. The time cost function of this pivot value may show multiple bowl shapes as illustrated in Figure 1. In many cases, the optimal pivot value is its median value. The reason is that if many disks are involved in an unbounded transition, there are benefits from disk I/O parallelism. The optimal pivot never lies at either end of the pivot range, because the resulting states may also require as many disks for their transition. For this reason, we intuitively acknowledge that selecting the median value of the pivot range will be a promising start to find a near optimal solution. However, as shown in Figure 1, this intuition can be wrong.

The $LMIN$ algorithm extends the intuition by using a binary search to locate a local minima. Even though it is not guaranteed to find the global minima of the time cost curve, it may find a local minima. This decreases the search space significantly by the factor of $\log |D|$. If necessary, the memoization technique can be applied before line 8 and after line 10.

6 Experimental Evaluation

To evaluate the performance of the proposed DRP algorithms, we implemented them on a test system and compared their results. The set of algorithms are ABD for unconstrained situations, $D\&C$, $D\&C$ + EB, and $SPACE$ for constrained space cost optimizations and

Algorithm 6 $LMIN(N, D, A, C)$

$n \leftarrow |N|, d \leftarrow |D|, a \leftarrow |A|, e \leftarrow |C| - n$
if $(n - d + a) > |C|$ **then**
  return ∞
else if $e \geq a$ **then**
  return $\omega_\tau(ABD(N, D, A))$
end if
$A' \leftarrow$ select e disks from A
$pivot \leftarrow$ BINARY_SEARCH(N, D, A, C)
$D' \leftarrow$ select $pivot$ number of disks from D
return $\omega_\tau(ABD(N, D', A'))$+
$LMIN(N - D' + A', D - D', A - A', C)$

$D\&C$, $TIME$, and $LMIN$ for constrained time cost optimizations.

For the constrained experiments, we varied the disk slot constraint from 2 to 70. For all bounded disk scaling requests in a given constraint, we collected the computed cost, the elapsed time, and the number of items stored in memory. We also computed the cost of the unbounded algorithm for all bounded test cases to have a baseline in order to examine how badly the system works by the bounded situation.

All the algorithms use the memoization technique to reduce the computation time. We use a splay tree data structure to minimize the number of access operations over a period of time [9]. The binary search algorithm in $LMIN$ was implemented as follows. First, compute the costs of the two end points and the mid point. Discard the one point out of the three with the highest cost and use the other two as the end points of the next interval. Continue to subdivide the new intervals until a single point is reached.

6.1 Comparison of the Space Cost Minimization Algorithms

Figure 2 shows the ratio of the space costs incurred by different algorithms as a function of the disk slot constraint c. The results show that $D\&C$, $D\&C$ + EB, and $SPACE$ all compute the same space cost (average and maximum). The average ratio of $SPACE$ to ABD converges to one as the disk constraint c increases, and the maximum ratio does not exceed a factor of two. We conclude that the amount of data moved in a constrained environment is never more than twice the minimum data moved in the unconstrained case.

Figure 3 compares the average and maximum computation time measured for different algorithms. Note that the time scale uses milli-seconds and is logarithmic. The average measured time of the $SPACE$ algorithm is constant for all constraints, but the maximum time increases slowly for higher values of c. This is because the computational complexity of the $SPACE$ algorithm depends only on the number of disks to delete, which increases with a growing constraint c. As expected, the $D\&C$ and $D\&C$ with EB algorithms perform worse than $SPACE$. Even though $D\&C$ with EB shows a faster response time than that of $D\&C$ in the average case, there is no significant advantage of using $D\&C$ with EB over $D\&C$ alone. The former performs worse than the latter in some cases, which is

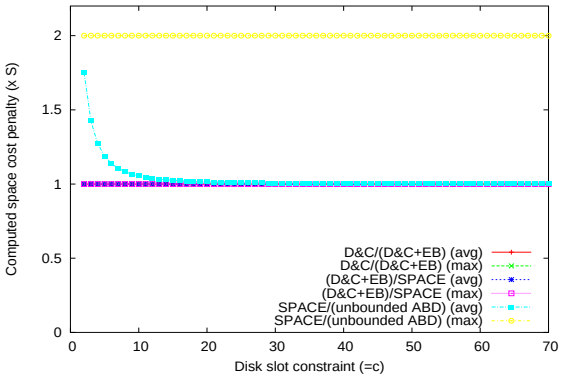

Fig. 2. Computed space cost ratio of six algorithms as a function of the disk slot constraint.

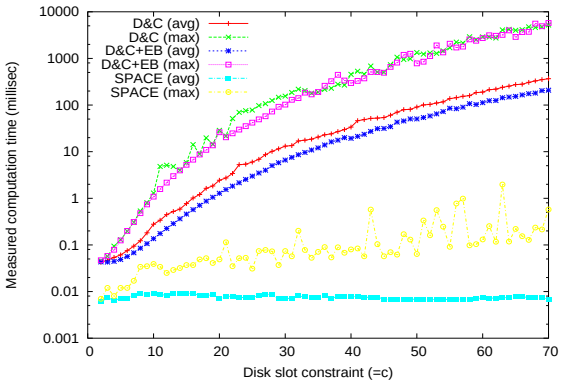

Fig. 3. Measured computation time to find optimal cost solution as a function of the disk slot constraint.

caused by the fact that different insertion and access sequences affects splay tree organization.

The $SPACE$ achieves both minimal computation time and finds the optimal space cost in linear time. The $D\&C$ with EB generally performs better than $D\&C$ alone, but under some conditions the two algorithms are the same.

6.2 Comparison of the Time Cost Minimization Algorithms

Figures 4 through 5 illustrate the same performance metrics as were used in the previous section. Here we compare the following four algorithms: $D\&C$, $TIME$, $LMIN$, and the unconstrained DRP algorithm, ABD.

In Figure 4 we see that the $D\&C$ and the $TIME$ algorithm perform the same within our test range and that the $LMIN$ solution produces a slightly worse result than $TIME$ solution does. The time costs computed by the constrained algorithms converge to a factor of two worse than that of the unconstrained algorithm on average. In the worst case, the time cost difference between the constrained and unconstrained algorithms are significant. This figure also shows that the worst time cost of the $LMIN$ algorithm increases slightly as the disk constraint increases. However, we expect that the gap between $LMIN$ and $TIME$ can be reduced if we adopt a more comprehensive search algorithm for

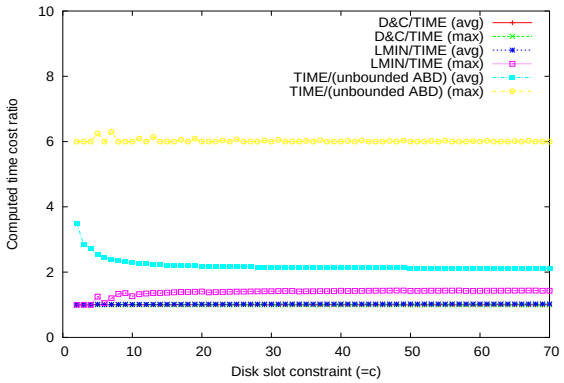

Fig. 4. Computed time cost ratio of algorithms as a function of the disk slot constraint.

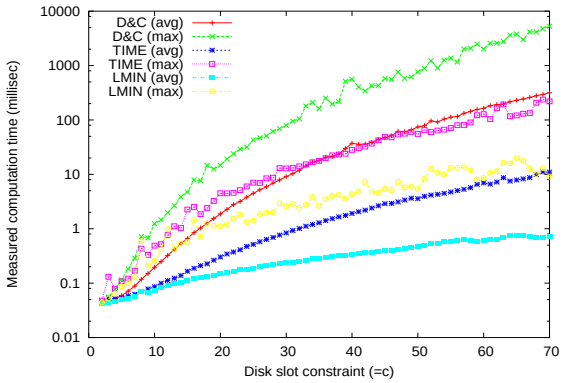

Fig. 5. Measured computation time to find the time cost solution as a function of the disk slot constraint.

$LMIN$ than the one currently implemented, with some expense to the search cost.

Figure 5 illustrates that the computation time of $LMIN$ increases more gradually than that of $TIME$. We also see that the average measured time of the $TIME$ algorithm is much longer than that of maximum measured time of $LMIN$. Furthermore, the average execution time of the $D\&C$ algorithm is much longer than the maximum time of $TIME$.

$TIME$ computes the same optimal time cost as the $D\&C$ algorithm, but with a shorter execution time. $LMIN$ has an obvious advantage on its computation time, but requires a more comprehensive local minima detection algorithm to reduce the time cost penalty.

7 Conclusions and Future Work

We presented the disk replacement problem, DRP, of finding a sequence of disk additions and removals for a storage system while migrating the data and respecting a constraint on the total number of available disk slots. If the system has enough empty disk slots to add all new disks the operation is termed *unbounded*. Otherwise, it will be *bounded*. We analytically modelled DRP and proved it to be a variation of the single-source shortest path problem. Thus, the upper bound of the complexity of finding the optimal sequence is polynomial.

For unbounded disk scaling requests, we proposed the $ABBD$ and ABD algorithms. $ABBD$ is the naive implementation of scaling sequences with disk additions and disk removals. The shortest sequence to minimize both the space and time cost is to add all new disks first and then to remove all disks to be deleted. The ABD algorithm improves upon $ABBD$ by combining two data re-balancing steps into one.

For bounded disk scaling requests, we proposed an exhaustive search algorithm based on a divide-and-conquer approach to minimize both the time and space costs. To reduce the its search space size, we introduced two space cost minimization heuristics, $D\&C$ + EB and $SPACE$, and two time cost minimization heuristics, $TIME$ and $LMIN$. The $D\&C$ + EB algorithm uses the EB component only when has system has at least one empty disk slot and the number of new disks is smaller than or equal to the number of disks to be removed. The $SPACE$ algorithm finds the optimal data moving sequence in linear time. $TIME$ reduces the $D\&C$ search space by filling all empty disk slots with new disks at once and then running the $D\&C$ algorithm. Finally, scaling sequences found by $LMIN$ are not guaranteed to be optimal but the search space is again reduced compared with $TIME$.

One aspect of data migration that we have not addressed in this paper is the limited storage of each disk. Hence, in some cases the presented algorithms may exceed the physically available storage on some devices temporarily. Such an overcommitment will happen only when the disk scaling operation involves excessive disk removals. These situations can be detected by keeping track of the disk storage capacities during computations and avoiding such cases. Another extension of this work will be to apply the proposed solutions to heterogeneous storage environments.

References

[1] R. Muntz, J. Santos, and S. Berson, RIO: A Real-time Multimedia Object Server, in *ACM Sigmetrics Performance Evaluation Review*, September 1997, vol. 25.

[2] J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto, Comparing Random Data Allocation and Data Striping in Multimedia Servers, in *Proceedings of the SIGMETRICS Conference*, Santa Clara, California, June 17-21 2000.

[3] Joseph Hall, Jason Hartline, Anna R. Karlin, Jared Saia, and John Wilkes, On Algorithms for Efficient Data Migration, in *12th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Washington DC, January 7-9, 2001.

[4] Eric Anderson, Joe Hall, Jason Hartline, Michael Hobbs, Anna R. Karlin, Jared Saia, Ram Swaminathan, and John Wilkes, An Experimental Study of Data Migration Algorithms, in *WAE: 5th International Workshop of Algorithm Engineering*, Aarhus, Denmark, August 28-30, 2001, LNCS.

[5] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence Mustafa Uysal, and Alistair Veitch, Hippodrome: Running Circles around Storage Administration, in *Conference on File and Storage Technologies (FAST'02)*, USENIX, Berkeley, CA, January 28-30, 2002, pp. 175–188.

[6] Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan, Algorithms for data migration with cloning, in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2003, pp. 27–36, ACM Press.

[7] T.H. Cormen et al., *Introduction to Algorithms*, The MIT Press, second edition, 2001.

[8] Roger Zimmermann and Beomjoo Seo, Efficient disk replacement and data migration algorithms for large disk subsystems," Tech. Rep., University of Southern California, 2004.

[9] D. D. Sleator and R. E. Tarjan, Self-adjusting binary search trees," *Journal of the ACM*, vol. 32, no. 3, pp. 652–686, 1985.