Fast Reconfiguration of Data Placement in Parallel Disks *

Srinivas Kashyap and Samir Khuller †
Department of Computer Science
University of Maryland
College Park, MD 20742
Email: {raaghav, samir}@cs.umd.edu

Yung-Chun (Justin) Wan
Google

Leana Golubchik
Department of Computer Science and IMSC
University of Southern California
Los Angeles, CA 90089
Email: leana@cs.usc.edu

1 Introduction

The "How much information?" study produced by the school of information management and systems at the University of California at Berkeley [10], estimates that about 5 exabytes of new information was produced in 2002. It estimates that the amount of stored information doubled in the period between 1999 and 2002. It is believed that more data will be created in the next five years than in the history of the world. Clearly we live in an era of data explosion. This data explosion necessitates the use of large storage systems. Storage Area Networks (or SANs) are the leading [13] infrastructure for enterprise storage.

A SAN essentially allows multiple processors to access several storage devices. They typically access the storage medium as though it were one large shared repository. One crucial function of such a storage system is that of deciding the placement of data within the system. This data placement is dependent on the demand pattern for the data. For instance, if a particular data item is very popular the storage system might want to host it on a disk with high bandwidth or make multiple copies of the item. The storage system needs to be capable of handling flash crowds [8]. During events triggered by such flash crowds, the demand distribution becomes highly skewed and different from the normal demand distribution.

It is known that the problem of computing an optimal data placement[1] for a given demand pattern is NP-Hard [5]. However, polynomial time approximation schemes as well as efficient combinatorial algorithms that compute almost optimal solutions are known for this problem [12, 11, 5]. So we can assume that a near-optimal placement can be computed once a demand pattern is specified.

As the demand pattern changes over time and the popularity of items changes, the storage system will have to modify its internal placement accordingly. Such a modification in placement will typically involve movement of data items from one set of disks to another or requires changing the number of copies of a data item in the system. For such a modification to be effective it should be computed and applied quickly. In this work we are concerned with the problem of finding such a modification i.e., modifying the existing placement to efficiently deal with a new demand pattern for the data. This problem is referred to as the data migration problem and was considered in [9, 6]. The authors used a data placement algorithm to compute a new "target" layout. The goal was to "convert" the existing layout to the target layout as quickly as possible. The communication model that was assumed was a half-duplex model where a matching on the disks can be fixed, and for each matched pair one can transfer a single object in a round. The goal was to minimize the number of rounds taken. The paper developed

*This research was supported by NSF grant CCF-0430650. The research was also funded in part by the NSF grant EIA-0091474, the Okawa Research Award, and the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

†Department of Computer Science and UMIACS

[1] An optimal placement will allow a maximum number of users to access information of their interest.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

constant factor approximation algorithms for this NP-hard problem [9]. In practice these algorithms find solutions that are reasonably close to optimal. However, even when there is *no* drastic change in the demand distribution it can still take many rounds of migration to achieve the new target layout. This happens since the scheme completely disregards the existing placement in trying to compute the target placement.

In this paper we consider a new approach to dealing with the problem of changes in the demand pattern. We ask the following question:

In a given number of migration rounds, can we obtain a layout by making changes to the existing layout so that the resulting layout will be the best possible layout that we can obtain within the specified number of rounds?

Of course, such a layout is interesting only if it is significantly better than the existing layout for the new demand pattern.

We approach the problem of finding a good layout that can be obtained in a specified number of rounds by trying to find a sequence of layouts. Each layout in the sequence can be transformed to the next layout in the sequence by applying a small set of changes to the current layout. These changes are computed so that they can be applied within one round of migration (a disk may be involved in at most one transfer per round).

We show that by making these changes even for a small number of consecutive rounds, the existing placement that was computed for the old demand pattern can be transformed into one that is almost as good as the best layout for the new demand pattern.

Our method can therefore be used to *quickly* transform an existing placement to deal with changes in the demand pattern. We *do not* make any assumptions about the type of demand changes – hence the method can be used to quickly deal with any type of change in the demands. We also show that the problem of finding an optimal set of changes that can be applied in one round is NP-hard (see Appendix A.1 for the proof). The proof demonstrates that some unexpected data movement patterns can yield a high benefit.

In the remaining part of the introduction, we present the model and the assumptions made, and restate our result formally.

1.1 Model summary We consider the following model for our storage system. There are N parallel disks that form a *Storage Area Network*. Each disk has a storage capacity of K and has a load handling capacity (or bandwidth) of L.

The efficiency of the system depends crucially on the *data layout* pattern that is chosen for the disks. This data layout pattern or data placement specifies for each item, which set of disks it is stored on (note that the whole item is stored on each of the disks specified by the placement, so these are copies of the item). The next problem is that of mapping the demand for data to disks. Each disk has an upper bound on the total demand that can be mapped to that disk. A simple way to find an optimal assignment of demand to disks, is by running a single network flow computation in an appropriately defined graph (see Section 2.1).

Different communication models can be considered based on how the disks are connected. We use the same model as in [2, 7] where the disks may communicate on any matching; in other words, the underlying communication graph allows for communication between any pair of devices via a matching (e.g., as in a switched storage network with unbounded backplane bandwidth). *This model best captures an architecture of parallel storage devices that are connected on a switched network with sufficient bandwidth. This is most appropriate for our application.* This model is one of the most widely used in all the work related to gossiping and broadcasting. These algorithms can also be extended to models where the size of the matching in each round is constrained [9]. This can be done by a simple simulation, where we only choose a maximal subset of transfers to perform in each round.

Suppose we are given an initial demand pattern $\mathcal{I}$. We use this to create an initial layout $L_{\mathcal{I}}$. Over time, the demand pattern for the data may change. At some point of time the initial layout $L_{\mathcal{I}}$ may not be very efficient. At this point the storage manager may wish to re-compute a new layout pattern. Suppose the target demand pattern is determined to be $\mathcal{T}$ (this could be determined based on the recent demand for data, or based upon projections determined by previous historical trends). Our goal is to migrate data from the current layout to a new layout. We would like this migration to complete quickly since the system is running inefficiently in addition to using a part of its local bandwidth for migrating data. It is therefore desirable to complete the conversion of one layout to another layout quickly. However, note that previous methods completely ignored the current layout and fixed a target layout $L_{\mathcal{T}}$ based on the demand $\mathcal{T}$. *Is it possible that there are layouts $\mathcal{L}'$ with the property that they are almost as good as $L_{\mathcal{T}}$, however, at the same time we can "convert" the initial layout $L_{\mathcal{I}}$ to $\mathcal{L}'$ in very few rounds (say compared to the number of rounds required to convert $L_{\mathcal{I}}$ to $L_{\mathcal{T}}$)?* It is our objective to consider this central question in this paper. In fact, we answer the question in the affirmative by doing a large set of

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

experiments.

To do this, we define the following *one round problem*. Given a layout $L_{\mathcal{P}}$ and a demand distribution $\mathcal{T}$, our goal is to find a one round migration (a matching), such that if we transfer data along this matching, we will get the maximum increase in utilization. In other words, we will "convert" the layout $L_{\mathcal{P}}$ to a new layout $L_{\mathcal{P}+1}$, such that we get the maximum utilization, and the new layout is obtainable from the current layout in one round of migration.

Now we can simply use an algorithm for the *one round problem* repeatedly by starting with the initial layout $L_{\mathcal{I}}$, and running ℓ iterations of the one round algorithm. We will obtain a layout $L_{\mathcal{I}+\ell}$, which could be almost as good is the target layout $L_{\mathcal{T}}$.

Of course there is no reason to assume that repeatedly solving the one round problem will actually yield an optimal solution for the ℓ round version of this problem. However, as we will see, this approach is very effective.

2 The problem

2.1 Example Since the formal definition of the problem will involve a lot of notation, we will first informally illustrate the problem and our approach using an example. In this example, we will show an initial demand distribution $\mathcal{I}$; an initial placement for this distribution $L_{\mathcal{I}}$; we will then show the changed demand distribution $\mathcal{T}$. We will show why the initial placement $L_{\mathcal{I}}$ is inadequate to handle the changed demand distribution $\mathcal{T}$. We will then show how a small change (a one-round migration) to the initial placement $L_{\mathcal{I}}$ results in a placement that is optimal for the new demand distribution.

In this toy example, we consider a storage system that consists of 4 identical disks. Each disk has storage capacity of 3 units and load capacity (or bandwidth) of 100 units. There are 9 data items that need to be stored in the system. The initial demand distribution $\mathcal{I}$ and the new demand distribution $\mathcal{T}$ are as follows:

Item	Initial demand	New Demand
A	130	55
B	90	55
C	40	20
D	30	60
E	25	5
F	25	10
G	25	15
H	22	70
I	13	110

The placement $L_{\mathcal{I}}$ (which in this case is also an optimal placement) obtained using the sliding window algorithm[2] for the demand distribution above is as follows (the numbers next to the items on disks indicates the mapping of demand to that copy of the item):

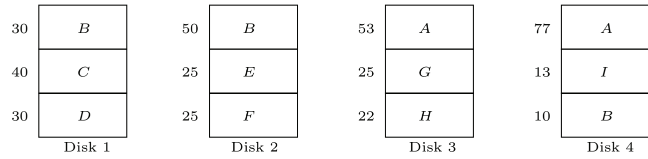

Figure 1: Optimal placement $L_{\mathcal{I}}$ for the initial demand distribution $\mathcal{I}$, satisfies all the demand. Storage capacity K=3, Bandwidth L=100. In addition to producing the layout the sliding window algorithm finds a mapping of demand to disks, which is optimal for the layout computed.

To determine the maximum amount of demand that the current placement $L_{\mathcal{I}}$ can satisfy for the new demand distribution $\mathcal{T}$, we compute the max-flow in a network constructed as follows. In this network we have a node corresponding to each item and a node corresponding to each disk. We also have a source and a sink vertex. We have edges from item vertices to disk vertices if in the placement $L_{\mathcal{I}}$, that item was put on the corresponding disk. Capacities of edges from the source to every item is equal to the demand for that item in the new distribution. The rest of the edges have capacity equal to the disk bandwidth. Using the flow network above, we can re-assign the demand $\mathcal{T}$ using the same placement $L_{\mathcal{I}}$ as given in Figure 3. Figure 2 shows the flow network obtained by applying the construction described above, corresponding to the initial placement $L_{\mathcal{I}}$ and new demand $\mathcal{T}$.

A small change can convert $L_{\mathcal{I}}$ to an optimal placement. In general, we would like to find changes that can be applied to the existing placement in a single round and get a placement that is close to an optimal placement for the new demand distribution. In a round a disk can either be the source or the target of a data transfer but not both. In fact, in this example a single change that involves copying an item from one disk to another is sufficient (and does not involve the other two disks in data transfers). This is illustrated in Figure 4.

We stress that we are not trying to minimize the total number of data transfers, but simply find the *best* set of changes that can be applied in parallel to modify the existing placement for the new demand distribution.

We compare this approach to that of previous works [9, 6] which completely disregard the existing placement

[2]The sliding window algorithm proposed by Shachnai and Tamir [12] is currently the best practical algorithm for this problem. For more on the sliding window algorithm and its performance, see [5].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

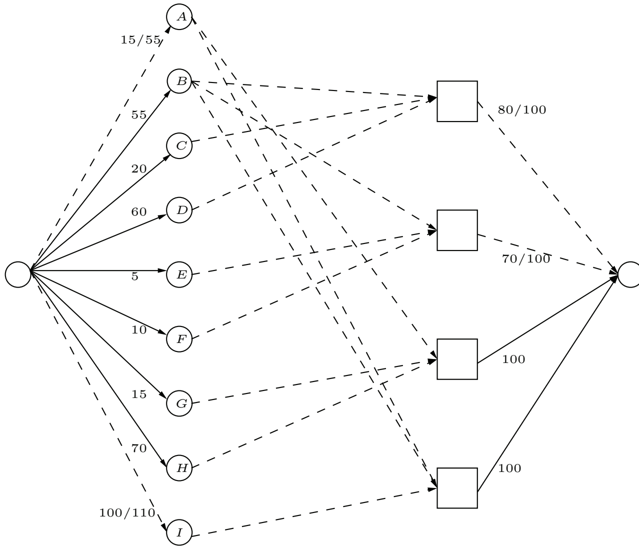

Figure 2: Flow network to determine maximum benefit of using placement $L_{\mathcal{I}}$ with demand distribution $\mathcal{T}$. $L_{\mathcal{I}}$ is sub-optimal for $\mathcal{T}$ and can only satisfy 350 out of a maximum of 400 units of demand. Saturated edges are show using solid lines.

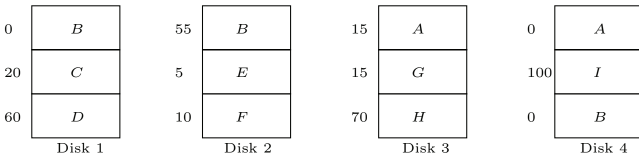

Figure 3: Maximum demand that placement $L_{\mathcal{I}}$ can satisfy for the new demand distribution $\mathcal{T}$. $L_{\mathcal{I}}$ is sub-optimal for $\mathcal{T}$ and can only satisfy 350 out of a maximum of 400 units of demand.

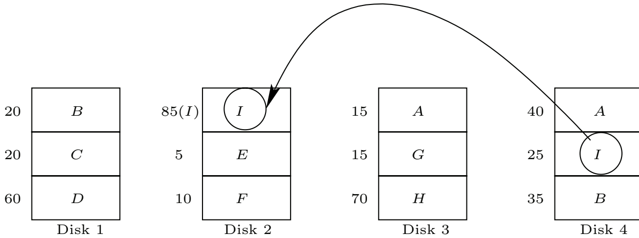

Figure 4: Removing item B from disk 2 and replacing it with a copy of item I from disk 4 converts $L_{\mathcal{I}}$ to an optimal placement $\mathcal{L}'$ for the new demand distribution $\mathcal{T}$. The placement shown above is optimal for $\mathcal{T}$ and satisfies all demand.

and simply try to minimize the number of parallel rounds needed to convert the existing placement to an optimal placement for the new demand distribution. In Fig. 6, we show that using the old approach, it takes 4 rounds of transfers to achieve what our approach did in a single round (and using just one transfer). In Figure 5 an optimal placement $L_{\mathcal{T}}$ is recomputed [3] for the new demand distribution $\mathcal{T}$. We show in Figure 6 the smallest set of transfers required to convert $L_{\mathcal{I}}$ to $L_{\mathcal{T}}$. *Note that both placement $\mathcal{L}'$ (obtained after the transfer shown in Figure 4 is applied) and placement $L_{\mathcal{T}}$ shown in Figure 5 are optimal placements for the new demand distribution $\mathcal{T}$.* Note that this is an optimal solution that also addresses the space constraint on the disk (this property is not actually maintained by the data migration algorithms developed earlier [9]).

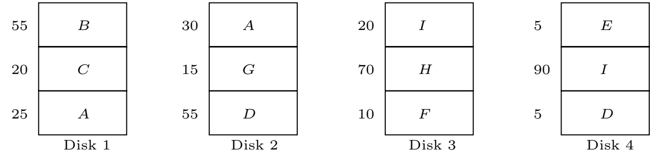

Figure 5: Placement $L_{\mathcal{T}}$. Output of the Sliding window algorithm for the new demand distribution $\mathcal{T}$.

2.2 Formal definition The storage system consists of N disks. Each disk has load-capacity of L and a storage-capacity of K. We have m items, each item j has size 1 and demand ℓ_j. This constitutes an m-dimensional demand distribution $\ell = (\ell_1, \ldots, \ell_m)$. An m-dimensional *placement* vector p_i for a disk i is $(p_{i1}, \ldots, p_{im})$ where p_{ij} are $0-1$ entries indicating that item j is on disk i. An m-dimensional *demand* vector d_i for a disk i is $(d_{i1}, \ldots, d_{im})$ where d_{ij} is the demand for item j assigned to disk i. Define $\mathcal{V}(\{d_i\}) = \sum_i \sum_j d_{ij}$ as the benefit of the set of demand vectors $\{d_i\}$. A set of placement and demand vectors that satisfy the following constraints is said to constitute a feasible placement and demand assignment:

1. $\sum_j p_{ij} \leq K$ for all disks i. This ensures that the storage-capacity is not violated.
2. $\sum_j d_{ij} \leq L$ for all disks i. This ensures that the load-capacity is not violated.
3. $d_{ij} \leq p_{ij}\ell_j$. This ensures that the demand for an item j is routed to disk i only if that item is present on disk i.
4. $\sum_i d_{ij} \leq \ell_j$ for all items j. This ensures that no more than the total demand for an item is packed.

A one-round-migration is essentially a matching on the set of disks. More formally, a one-round-migration is a 0-1 function $\Delta(s_d, s_i, t_d, t_i)$ where $s_d, t_d \in \{1, \ldots, N\}$

[3]Using the sliding window algorithm for computing a placement for a given demand.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

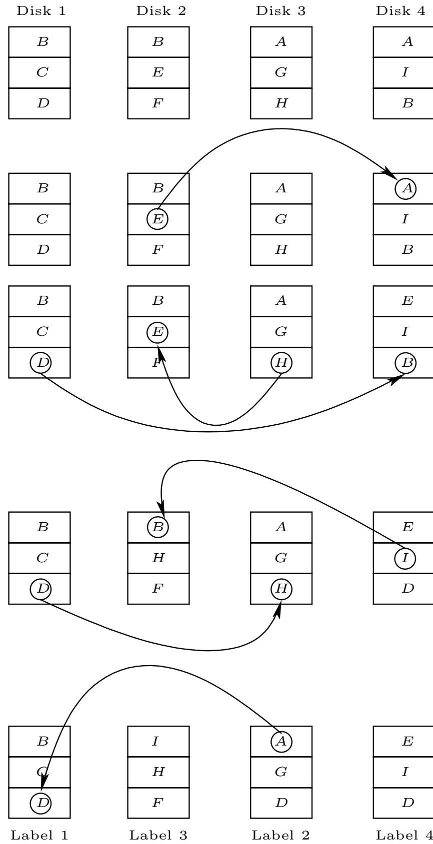

Figure 6: Transforming $L_{\mathcal{I}}$ to $L_{\mathcal{T}}$ takes 4 rounds. Note that the disks here will need to be renumbered to match the sliding window output. Final disk 2 corresponds to disk 3 in the sliding window output, final disk 3 corresponds to disk 2 in the sliding window output.

and $s_i, t_i \in \{1, \ldots, m\}$. Here s_d is the source disk, s_i is the source item, t_d is the target disk, t_i is the target item. Further, $\Delta(.)$ has to satisfy the following conditions:

1. $\sum_{t_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i) \leq 1$ for all disks s_d. This ensures that a disk can be the source for at most one transfer.

2. $\sum_{s_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i) \leq 1$ for all disks t_d. This ensures that a disk can be the target for at most one transfer.

3. $\left(\sum_{s_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i)\right) + \left(\sum_{t_d} \sum_{s_i} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i)\right) \leq 1$ for all disk pairs $s_d = t_d$. This ensures that a disk can simultaneously not be both a source and a target.

4. $(s_d = t_d) \Rightarrow \Delta(s_d, *, t_d, *) = 0$. This ensures that there are no self loops in the transfer graph.

5. $\sum_{t_d} \sum_{t_i} \Delta(s_d, s_i, t_d, t_i) \leq p_{s_d s_i}$ for all disks s_d and items s_i. This ensures that a disk s_d can be source of an item s_i only if that item is on that disk (i.e. $p_{s_d s_i} = 1$).

We can apply this function to an existing placement to obtain a new placement as follows. If $\Delta(s_d, s_i, t_d, t_i) = 1$, then set $p_{t_d t_i} = 0$ and $p_{t_d s_i} = 1$. We compute the optimal demand assignment for the new placement using max-flow.

ONE-ROUND-MIGRATION: When given an initial demand distribution $\ell_{initial}$, a corresponding set of feasible placement vectors $\{p_i\}$, and demand vectors $\{d_i\}$ and a final demand distribution ℓ_{final}, the problem asks for a one-round-migration $\Delta(.)$ that when applied to the initial placement yields placement vectors $\{p_i^*\}$ and demand vectors $\{d_i^*\}$ such that $\mathcal{V}(\{d_i^*\})$ is maximized.

We show that this problem is NP-Hard (See Appendix A.1 for a proof).

3 Algorithm for one round migration

For any disk d, let $I(d)$ denote the items on that disk.

Corresponding to any placement $\{\mathbf{p_i}\}$ (a placement specifies for each item, which set of disks it is stored on), we define the corresponding flow graph $G_p(V, E)$ as follows. We add one node a_i to the graph for each item $i \in \{1 \ldots m\}$. We add one node d_j for each disk $j \in \{1 \ldots N\}$. We add one source vertex s and one sink vertex t. We add edges (s, a_i) for each item i. Each of these edges have capacity `demand`(i) (where `demand`(i) is the demand for item i). We also add edges (d_j, t) for each disk j. These edges have capacity L (where L is the load capacity of disk j). For every disk j and for every item $i \in I(j)$, we add an edge (a_i, d_j) with capacity L.

The algorithm starts with the initial placement and works in phases. At the end of each phase, it outputs a pair of disks and a transfer corresponding to that disk pair.

We determine the disk and transfer pair as follows. Consider a phase r. Let $\{\mathbf{p_i}\}_r$ be the current placement. For every pair of disks d_i and d_j, for every pair of items (a_i, a_j) in $I(d_i) \times I(d_j)$, modify the placement $\{\mathbf{p_i}\}_r$ to obtain $\{\mathbf{p_i'}\}_r$ by overwriting a_j on d_j with a_i. Compute the max-flow in the flow graph for the placement $\{\mathbf{p_i'}\}_r$. Note down the max-flow value and revert the placement back to $\{\mathbf{p_i}\}_r$. After we go through all pairs, pick the (a_i, a_j) transfer pair and the corresponding (d_i, d_j) disk pair that resulted in the flow-graph with the largest max-flow value. Apply the transfer (a_i, a_j) modifying placement $\{\mathbf{p_i}\}_r$ to obtain $\{\mathbf{p_i}\}_{r+1}$ - which will be the starting placement for the next phase. We can no longer use disks d_i and d_j in the next phase. Repeat until there is no pair that can increase the max-flow or till we run

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

out of disks.

4 Speeding up the algorithm

The algorithm described in Section 3 recomputes max-flow in the flow graph from scratch when evaluating each move. Recall that the algorithm proceeds in phases and at the end of each phase, it identifies a pair of disks (d_i, d_j) and a $(a_i \in I(d_i), a_j \in I(d_j))$ transfer for that pair of disks.

We can speed up the algorithm by observing that the max-flow value increases monotonically from one phase to the next and therefore we need not recompute max-flow from scratch for each phase. Rather, we compute the residual network for the flow graph once and then make incremental changes to this residual network for each max-flow computation. All max-flow computations in this version of the algorithm are computed using the Edmonds-Karp algorithm (see [1]). Let G_i denote the residual graph at the end of phase i. Let G_0 be the residual graph corresponding to the initial graph. All max-flow computations in phase $i+1$, we begin with the residual graph G_i and find augmenting paths (using BFS on the residual graph) to evaluate the max-flow. After each transfer pair in phase $i+1$ is considered, we undo the changes to the residual graph and revert back to G_i. At the end of phase $i+1$, we apply the best transfer found in that phase, recompute max-flow and use the corresponding residual graph as G_{i+1}.

Even with the speedup, the algorithm needs to perform around 415,000 max-flow computations even for one of the smallest instances (N=60, K=15) that we consider in our experiments. Since we want to quickly compute the one-round migration, too many flow computations are not acceptable. We therefore consider the following variants of our algorithm. In our experiments, we found these variants to yield solutions that are as good as the algorithm described above.

Variant 1: For every pair of disks d_i and d_j, let $I_+(d_i)$ be the set of items on disk d_i that have **unsatisfied demand**. For every pair of items (a_i, a_j) in $I_+(d_i) \times I(d_j)$, overwrite a_j on d_j with a_i, compute the max-flow. Pick the (a_i, a_j) pair that gives the largest increase in the max-flow value. Repeat till there is no pair that can increase the max-flow or until we run out of disks.

Variant 2: For every pair of disks d_i and d_j, let $I_+(d_i)$ be the set of items on disk d_i that have **unsatisfied demand** and $I_-(d_j)$ be the items with **lowest demand** on disk d_j. For every pair of items (a_i, a_j) in $I_+(d_i) \times I_-(d_j)$, overwrite a_j on d_j with a_i, compute the max-flow. Pick the (a_i, a_j) pair that gave the largest increase in the max-flow value. Repeat till there is no pair that can increase the max-flow or until we run out of disks.

All the experimental results that we present in Section 5 are obtained using the second variant (described above). To solve[4] even the largest instances in our experiments, a C (gcc 3.3) implementation of the second variant took only a couple of seconds while the brute force algorithm took on the order of several hours.

5 Experiments

In this section, we describe the experiments used to evaluate the performance of our heuristic and compare it to the old approach to data migration. The framework of our experiments is as follows:

1. (*Create an initial layout*) Run the sliding window algorithm [5], given the number of user requests for each data object.
2. (*Create a target layout*) To obtain a target layout, we take one of the following approaches.
   (a) Shuffle method 1: Initial demand distribution is chosen with Zipf (will be defined later in this section) parameter 0.0 (high-skew). To generate the target distribution, pick 20% of the items and promote them to become more popular items.
   (b) Shuffle method 2: Initial demand distribution is chosen with Zipf parameter 0.0 (high-skew). To generate the target distribution, the lowest popularity item is promoted to become the most popular item.
   (c) Shuffle method 3: The initial demand distribution is chosen with Zipf parameter 1.0 (uniform-distribution). The target distribution is chosen with Zipf parameter 0.0 (high-skew).
   (d) Shuffle method 4: The initial demand distribution is chosen with Zipf parameter 0.0 (high-skew). The target distribution is chosen with Zipf parameter 1.0 (uniform-distribution).
3. Record the number of rounds required by the old data migration scheme to migrate the initial layout to the target layout.
4. Record the layout obtained in each round of our heuristic. Run 10 successive rounds of our one round migration starting from the initial layout.

[4]Experiments were run on a 2.8Ghz Pentium 4C processor with 1GB RAM running Ubuntu Linux 5.04.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

The layout output after running these 10 successive rounds of our heuristic will be considered as the final layout output by our heuristic.

We note that few large-scale measurement studies exist for the applications of interest here (e.g., video-on-demand systems), and hence below we are considering several potentially interesting distributions. Some of these correspond to existing measurement studies (as noted below) and others we consider in order to explore the performance characteristics of our algorithms and to further improve the understanding of such algorithms. For instance, a Zipf distribution is often used for characterizing people's preferences.

Zipf Distribution The Zipf distribution is defined as follows:

$$\text{Prob(request for item } i) = \frac{c}{i^{1-\theta}} \quad \begin{array}{c} \forall i = 1, \ldots, M \\ \text{and} \\ 0 \le \theta \le 1 \end{array}$$

$$\text{where} \quad c = \frac{1}{H_M^{1-\theta}} \quad \text{and} \quad H_M^{1-\theta} = \sum_{j=1}^{M} \frac{1}{j^{1-\theta}}$$

and θ determines the degree of skewness. For instance, $\theta = 1.0$ corresponds to the uniform distribution, whereas $\theta = 0.0$ corresponds to the skewness in access patterns often attributed to movies-on-demand type applications. See for instance the measurements performed in [3]. Flash crowds are also known to skew access patterns according to Zipf distribution [8]. In the experiments below, Zipf parameters are chosen according to the shuffle methods described earlier in the section.

We now describe the storage system parameters used in the experiments, namely the number of disks, space capacity, and load capacity (the maximum number of simultaneous user requests that a disk may serve).

In the first set of experiments, we used a value of 60 disks. We tried three different pairs of settings for space and load capacities, namely: (A) 15 and 40, (B) 30 and 35, and (C) 60 and 150.

In the second set of experiments, we varied the number of disks from 10 to 100 in steps of 10. We used a value of K=60, L=150 (this is the 3rd pair of L,K values used in the first set of experiments).

We obtained these numbers from the specifications of modern SCSI hard drives. For example, a 72GB 15,000 rpm disk can support a sustained transfer rate of 75MB/s with an average seek time of around 3.5ms. Considering MPEG-2 movies of 2 hours each with encoding rates of 6Mbps, and assuming the transfer rate under parallel load is 40% of the sustained rate, the disk can store 15 movies and support 40 streams. The space capacity 30 and the load capacity 35 are obtained from using a 150GB 10,000 rpm disk with a 72MB/s sustained transfer rate. The space capacity 60 and the load capacity 150 are obtained by assuming that movies are encoded using MPEG-4 format (instead of MPEG-2). So a disk is capable of storing more movies and supporting more streams. For each tuple of N,L,K and shuffle method we generated 10 instances. These instances were then solved using both our heuristic as well as the old data migration heuristic. The results for each N,L,K and shuffle method tuple were averaged out over these 10 runs.

5.1 Results and Discussion Figures 7, 8, 9, 10 and Tables 1, 2, 3 correspond to the first set of experiments. Figures 12, 13, 14, 15, 11 and Tables 4, 5, 6, 7 correspond to the second set of experiments.

Figures 8, 9, 10, 12, 13, 14, 15 compare the solution quality of our heuristic with that of the old approach. Tables 1, 2, 3, 4, 5, 6, 7 and Figure 7 compares the number of rounds taken by our approach with the number of rounds taken by the old approach to achieve similar solution quality.

We highlight the following observations supported by our experimental results:

- In all our experiments, our heuristic was able to get within 8% of the optimal solution using 10 rounds. This can be seen in all the figures and tables. For instance, see Figure 7.

- In comparison (see Figure 7 and Tables 1, 2, 3, 4, 5, 6, 7), the old scheme took a significantly larger number of rounds. For example, in the case of K=60, L=150 (corresponding to storing video as mpeg-4) the old scheme took over 100 rounds for every shuffle method and for every value of N we used, while our scheme was able to achieve similar solution quality within 10 rounds.

- Response to change in demand distribution: The experiments reveal an interesting behavior of the heuristic. When the target demand distribution is highly skewed, the heuristic's response or the amount of improvement made in successive rounds is linear. In contrast, when the demand is less skewed (i.e. the demand distribution is significantly different from the initial distribution but still the target distribution is not very skewed), the response is much sharper. For example in Figure 11, consider the response curve for shuffle methods 4 and 2 (low-skew) and contrast it with the flat response curves for shuffle methods 1 and 3 (high-skew).

    - Sharp response or diminishing returns: For a concrete example; in Figure 12 the improve-

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

ment obtained by our heuristic in the first round is almost as high as 10%, but successive improvements taper off quickly. This probably happens because we use a greedy algorithm and most of the gains are made in the first round and since this type of behavior is observed mainly when the demand is less skewed, there are presumably several items that need to replicated.

  - Flat response: For a concrete example; in Figure 14 the improvement obtained by our heuristic for N=100 in the first two rounds (1 and 2) is just about twice the benefit obtained in the last two rounds (9 and 10). This is probably because most of the load is concentrated on a few items and there is a large amount of unsatisfied demand. In each round we make more copies of these high popularity items and see almost the same benefit in each round.

- The case for this type of approach (that of making small changes to existing placement in consecutive rounds) is best supported by results from Table 7. This is an example of a case where the existing placement is already very good for the target distribution. The storage manager may wish to do a few rounds of migration to recover the amount of lost load. Our scheme lets the storage manager do such a quick adaptation. In contrast the old scheme takes over 150 rounds on average to achieve comparable results. This is especially unacceptable given that we already start off with a pretty good placement. In fact, shuffle method 4 seemed to consistently trigger expensive migrations in the old scheme while our scheme was able to get close to optimal within a couple of rounds. This is not surprising since the old scheme completely disregards the existing placement.

- Shuffle method 3 seemed to produce "harder" instances for our heuristic compared to the other shuffle methods we tried. This is not surprising since shuffle method 3 makes a drastic change to the demand distribution (moving it from uniform to highly skewed Zipf).

- It is very promising that our scheme performs particularly well for shuffle methods 1 and 2 (which is the type of demand change we expect to see in practice).

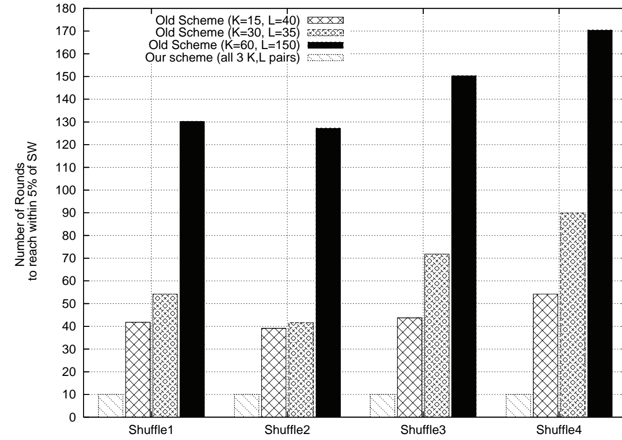

Figure 7: Plot compares the number of rounds that the old migration scheme took to reach within 5% of the optimal solution. We used N=60 and tried each of the shuffle methods for every pair of K and L shown in the plot. Every data point was obtained by averaging over 10 runs. In each of the experiments shown above, our scheme was set to run for 10 consecutive rounds.

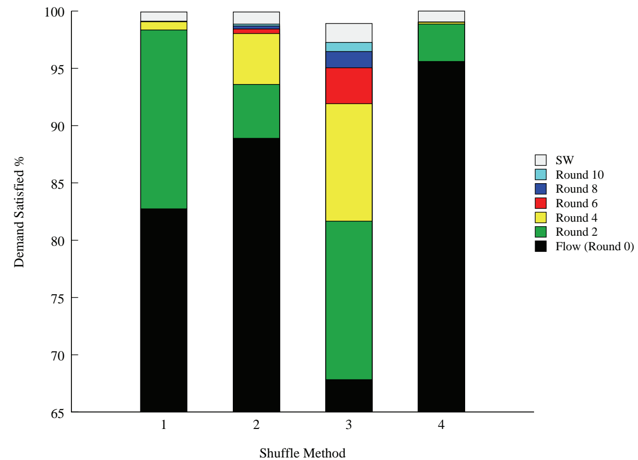

Figure 8: Plot shows improvement obtained by our scheme when presented with instances generated using the different shuffle methods. We used N=60, K=15, L=40 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

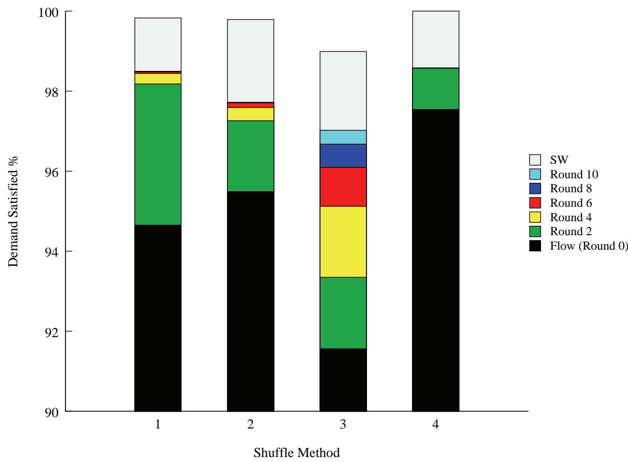

Figure 9: Plot shows improvement obtained by our scheme when presented with instances generated using the different shuffle methods. We used N=60, K=30, L=35 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

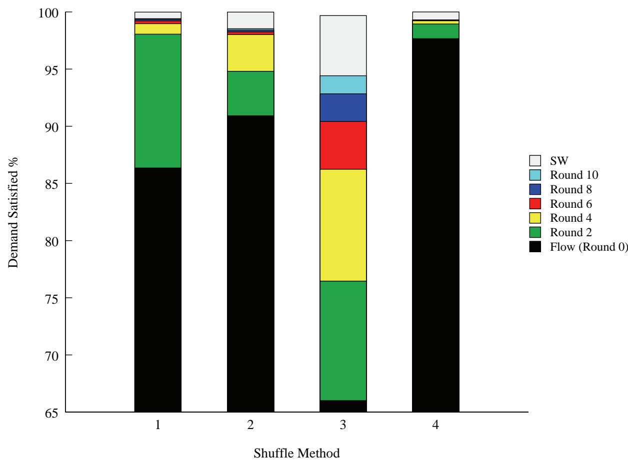

Figure 10: Plot shows improvement obtained by our scheme when presented with instances generated using the different shuffle methods. We used N=60, K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

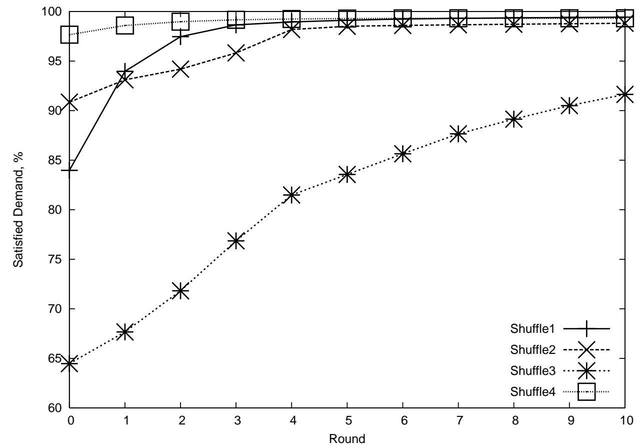

Figure 11: Plot comparing the response of our heuristic to the various shuffle methods. The response to shuffle 3 and shuffle 2 is much flatter than the diminishing returns type of response for shuffle 4 and shuffle 1. We used N=100, K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs.

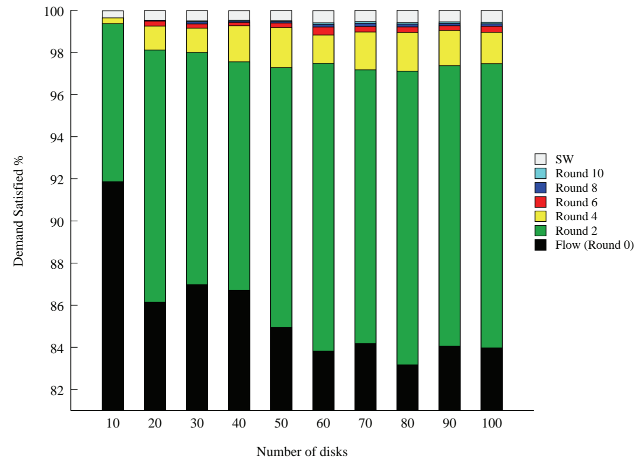

Figure 12: Performance of our scheme with varying number of disks for shuffle method 1. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

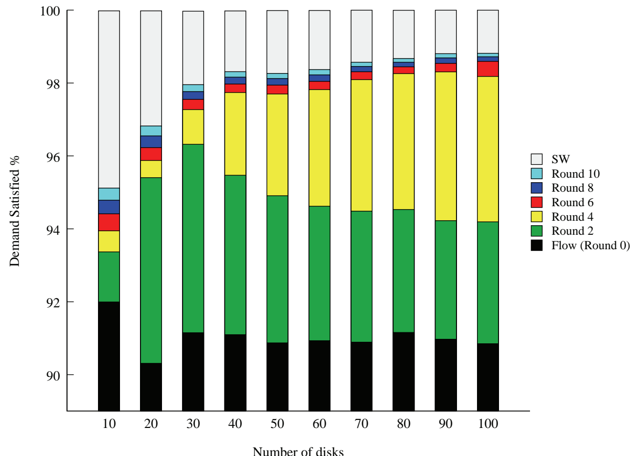

Figure 13: Performance of our scheme with varying number of disks for shuffle method 2. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

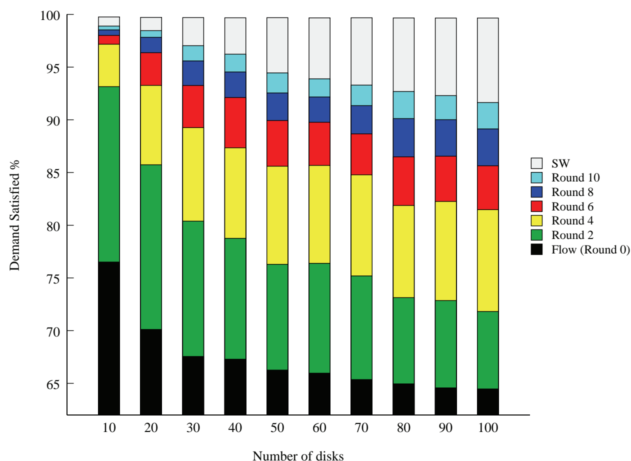

Figure 14: Performance of our scheme with varying number of disks for shuffle method 3. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

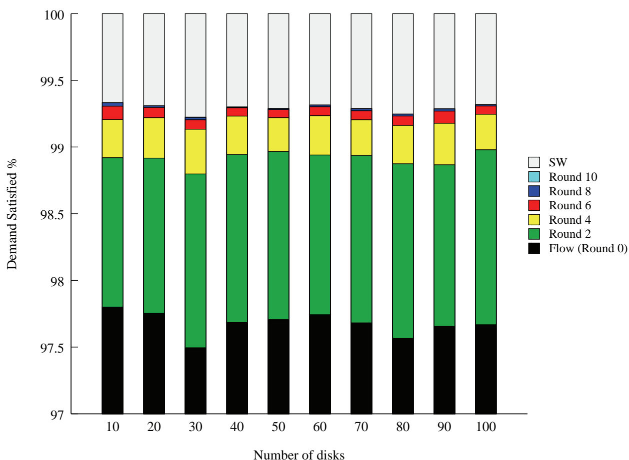

Figure 15: Performance of our scheme with varying number of disks for shuffle method 4. The number of disks N varied from 10 to 100. K=60, L=150 for each experiment. Every data point was obtained by averaging over 10 runs. SW is the solution value achieved by the old method.

Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	**10**	99.10	**41.8**	99.92
2	**10**	98.86	**39.1**	99.92
3	**10**	97.26	**43.7**	98.91
4	**10**	99.04	**54.2**	100

Table 1: Comparison of old scheme with our scheme for N=60, K=15, L=40

Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	**10**	98.50	**54.2**	99.83
2	**10**	97.72	**41.6**	99.79
3	**10**	97.02	**71.8**	98.99
4	**10**	98.57	**89.9**	100

Table 2: Comparison of old scheme with our scheme for N=60, K=30, L=35

Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	**10**	99.41	**130.3**	99.98
2	**10**	98.54	**127.3**	99.99
3	**10**	94.41	**150.4**	99.68
4	**10**	99.30	**170.5**	100

Table 3: Comparison of old scheme with our scheme for N=60, K=60, L=150

6 Conclusion

We proposed a new approach to deal with the problem of changing demand. We defined the one-round-migration problem to aid us in our effort. We showed that the one-round-migration problem is NP-Hard and

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	99.64	**104.8**	99.98
20	**10**	99.52	**111.8**	99.99
30	**10**	99.50	**121**	99.99
40	**10**	99.53	**121.9**	99.98
50	**10**	99.51	**125.9**	99.99
60	**10**	99.41	**128.2**	99.98
70	**10**	99.46	**128.5**	99.99
80	**10**	99.42	**135.4**	100
90	**10**	99.45	**138.6**	99.99
100	**10**	99.43	**136.2**	99.99

Table 4: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 1. K=60, L=150.

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	95.12	**103.3**	99.98
20	**10**	96.82	**109.4**	99.98
30	**10**	97.96	**119.4**	99.97
40	**10**	98.31	**119.7**	99.98
50	**10**	98.26	**124.7**	99.99
60	**10**	98.37	**127.4**	100
70	**10**	98.57	**129.3**	100
80	**10**	98.67	**135.3**	100
90	**10**	98.81	**134.4**	100
100	**10**	98.82	**137.5**	100

Table 5: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 2. K=60, L=150.

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	98.89	**126.2**	99.75
20	**10**	98.46	**133**	99.71
30	**10**	97.03	**138.8**	99.7
40	**10**	96.22	**144.1**	99.68
50	**10**	94.45	**146.5**	99.69
60	**10**	93.89	**149.4**	99.67
70	**10**	93.29	**149.8**	99.68
80	**10**	92.69	**154.6**	99.66
90	**10**	92.29	**152.8**	99.65
100	**10**	91.63	**155.9**	99.66

Table 6: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 3. K=60, L=150.

that unexpected data movement patterns can yield high benefit. We gave heuristics for the problem. We gave experimental evidence to suggest that our approach of doing a few rounds of one-round-migration consecutively performs very well in practice. In particular, in all our experiments they were able to quickly adapt the existing

N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	99.33	**128.3**	100
20	**10**	99.31	**139.6**	100
30	**10**	99.22	**148.9**	100
40	**10**	99.30	**159.2**	100
50	**10**	99.29	**166.6**	100
60	**10**	99.32	**170.8**	100
70	**10**	99.29	**178.7**	100
80	**10**	99.25	**183.6**	100
90	**10**	99.29	**190.3**	100
100	**10**	99.32	**196.1**	100

Table 7: Comparison of old scheme with our scheme for various number of disks N=10 to 100. Shuffle method 4. K=60, L=150.

placement to one that is close to the optimal solution for the changed demand pattern. We showed that, in contrast, previous approaches took many more rounds to achieve similar solution quality.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] Eric Anderson, Joseph Hall, Jason D. Hartline, Michael Hobbs, Anna R. Karlin, Jared Saia, Ram Swaminathan, and John Wilkes. An experimental study of data migration algorithms. In *WAE '01: Proceedings of the 5th International Workshop on Algorithm Engineering*, pages 145–158, London, UK, 2001. Springer-Verlag.

[3] Ann Louise Chervenak. *Tertiary storage: an evaluation of new applications*. PhD thesis, Berkeley, CA, USA, 1994.

[4] Shahram Ghandeharizadeh and Richard Muntz. Design and implementation of scalable continuous media servers. *Parallel Comput.*, 24(1):91–122, 1998.

[5] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 223–232, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[6] L. Golubchik, S. Khuller, Y. Kim, S. Shargorodskaya, and Y-C. Wan. Data migration on parallel disks. In *Proc. of European Symp. on Algorithms (2004). LNCS 3221*, pages 689–701. Springer, 2004.

[7] Joseph Hall, Jason Hartline, Anna R. Karlin, Jared Saia, and John Wilkes. On algorithms for efficient data migration. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 620–629, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[8] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites, 2002.

[9] Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan. Algorithms for data migration with cloning. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 27–36, New York, NY, USA, 2003. ACM Press.

[10] How much information? School of Information Management and Systems. University of California at Berkeley. http://www.sims.berkeley.edu/research/projects/how-much-info 2003/.

[11] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. In *Proceedings of Workshop on Approximation Algorithms (APPROX). LNCS 1913*, pages 238–249. Springer-Verlag, 2000.

[12] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29(3):442–467, 2001.

[13] Introduction to Storage Area Networks. IBM Redbook. http://www.redbooks.ibm.com/.

A Appendix

A.1 Hardness proof Recall that the *Subset-Sum* Problem is known to be NP-complete [4]. The *Subset-Sum* problem is defined as follows: Given a set $S = \{a_1, \ldots, a_n\}$ and a number b, where $a_i, b \in \mathcal{Z}^+$. Does there exist a subset $S' \subset S$ such that $\sum_{a_j \in S'} a_j = b$? Let $\text{sum}(S) = \sum_{a_i \in S} a_i$.

The *One-Round Migration* problem is defined as follows. We are given a collection of identical disks $D_1, \ldots D_N$. Each disk has a storage capacity of K, and a load capacity of L. We are also given a collection of data objects $M_1, \ldots M_M$, and a layout of the data objects on the disks. The layout specifies the subset of K data objects stored on each disk. Each data object M_i has demand u_i. The demand for any data object may be assigned to the set of disks containing that object (demand is splittable), without violating the load capacity of the disks. For a given layout, there may be no solution that satisfies all the demand. Is there a one-round migration to compute a new layout in which all the demand can be satisfied?

A one-round migration is a matching among the disks, such that for each edge in the matching, one source disk may send an item to a disk that it is matched to (half-duplex model).

We show that the One-Round Migration problem is NP-hard by reducing Subset-Sum to it. We will create a set of $N = 3n + 4$ disks, each having capacity two ($K = 2$). There are $4n + 6$ items in all. We will assume that L is very large. The current layout is shown in Figure 16.

The demand for various items is as follows: Demand for G_i is $L - a_i$. Demand for $C_i = \frac{L}{2} + a_i$. Demand for $E_i = \frac{L}{2}$. Demand for $F_i = L - a_i$.

Demand for $A = \text{sum}(A) + \frac{L}{2}$. Demand for $H = \text{sum}(A) + \frac{L}{2}$. Demand for $X = \frac{L}{2}$. Demand for $Y = L - b$. Demand for $Z = L - (\text{sum}(A) - b)$. Demand for $W = \frac{L}{2}$.

If we assume that the demand for C_i is $\frac{L}{2}$ then the assignment shown can satisfy all the demand. We will assume that all but two of the disks are load saturated (total assigned demand is exactly L). *If the demand for C_i increases by a_i*, then we have to re-assign some of this demand. The claim is that all of the demand can be handled after one round of migration if and only if there is a solution to the subset-sum instance. It is clear that a given solution (a matching) can be verified in polynomial time.

($\Rightarrow$) Suppose there is a subset $S' \subset S$ that adds

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

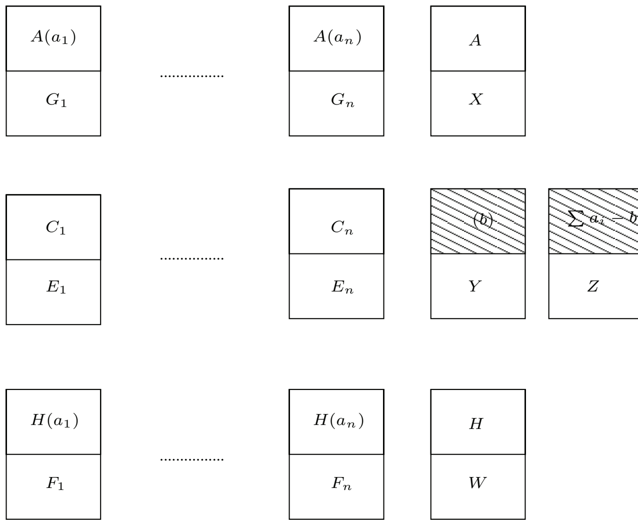

Figure 16: Reduction from SUBSET-SUM to ONE-ROUND-MIGRATION. Shaded portion indicates empty space. Number within brackets following item name indicates the amount of load assigned to the item.

exactly to b. We copy H (from the disk containing H and W) to the disk containing Z, and A (from the disk containing X and A) to the disk containing Y. If $a_i \in S'$ then we copy C_i to the disk containing G_i, and over-write the copy of A on that disk. All clients for A from this set of disks can be moved to the disk containing A and Y. If $a_i \notin S'$ then we copy C_i to the disk containing F_i and over-write the copy of H on that disk. All clients for H from this set of disks can be moved to the disk containing H and Z.

($\Leftarrow$) First note that the total demand is $3nL + 4L$. Since there are $3n + 4$ disks, all disks must be load saturated for a solution to exist. We leave it for the reader to verify that with the current layout there is no solution that meets all the demand. Suppose there exists a one-round migration that enables a solution where all of the demand can be assigned. A new copy has to be created for each C_i, or E_i since the total load for C_i and E_i is $L + a_i$, and exceeds L. Assume w.lo.g that a copy of C_i will be made to handle the excess demand of a_i on this disk. We also assume without loss of generality that $a_i < b$ so moving C_i to one of the disks containing Y or Z would not be of much use in load saturating those disks. The only choice is to decide whether this new copy is made at the expense of a copy of H or at the expense of a copy of A. Note that C_i cannot overwrite any of the other items since only a single copy of these items exists in the system. Since this is a one-round migration, we cannot move a single copy of an item to another disk, and then re-write it subsequently. Note that C_i has to overwrite the corresponding A disk or H disk, otherwise we will be unable to recover all the demand. Since the disks containing Y and Z are also load saturated, we will copy an item onto those disks. Moreover we have to move one item (either A or H) to the disk containing Y. Suppose that A is copied to the disk containing Y and H is copied to the disk containing Z. (The reverse case is similar.) When we shift b amount of demand of A to the disk containing Y, we have to completely remove the demand from a disk containing A, otherwise we will lose some demand. If C_i is moved to a disk containing A then $a_i \in S'$. If C_i is moved to a disk containing H then $a_i \notin S'$. Since C_i over-writes A (H), all of the demand of A (H) is moved out of the disk. Clearly, the total size of S' must be exactly b.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited