Data Migration on Parallel Disks [*]

Leana Golubchik[†] Samir Khuller[‡] Yoo-Ah Kim [§] Svetlana Shargorodskaya[¶]
Yung-Chun (Justin) Wan [||]

Abstract

Our work is motivated by the problem of managing data on storage devices, typically a set of disks. Such storage servers are used as web servers or multimedia servers, for handling high demand for data. As the system is running, it needs to dynamically respond to changes in demand for different data items. There are known algorithms for mapping demand to a layout. When the demand changes, a new layout is computed. In this work we study the *data migration problem*, which arises when we need to quickly change one layout to another. This problem has been studied earlier when for each disk the new layout has been prescribed. However, to apply these algorithms effectively, we identify another problem that we refer to as the correspondence problem, whose solution has a significant impact on the solution for the data migration problem. We examine algorithms for the data migration problem in more detail and identify variations of the basic algorithm that seem to improve performance in practice, even though some of the variations have poor worst case behavior.

1 Introduction

To handle high demand, especially for multimedia data, a common approach is to replicate data objects within the storage system. Typically, a large storage server consists of several disks connected using a dedicated network, called a *Storage Area Network*. Disks typically have constraints on storage as well as the number of clients that can access data from a single disk simultaneously. The goal is to have the system automatically respond to changes in demand patterns and to recompute data layouts. Such systems and their applications are described and studied in, e.g., [5, 6, 19] and the references therein.

Approximation algorithms have been developed [13, 14, 15, 7, 10] to map known demand for data to a specific data layout pattern to maximize utilization[1]. In the layout, we compute not only how many copies of each item we need, but also a layout pattern that specifies the precise subset of items on each disk[2]. The problem is NP-hard, but there are polynomial time approximation schemes [7, 14, 10]. Given the relative demand for data, an almost optimal layout can be computed.

Over time as the demand pattern changes, the system needs to create *new* data layouts. The problem we are interested in is the problem of computing a data migration plan for the set of disks to convert an initial layout to a target layout. We assume that data objects have the same size (these could be data blocks or files) and that it takes the same amount of time to migrate a data item between any pair of disks. The crucial constraint is that each disk can participate in the transfer of only one item – either as a sender or as a receiver. Our goal is to find a migration schedule to minimize the time taken (i.e., number of rounds) to complete the migration (makespan) since the system is running inefficiently until the new layout has been obtained.

A special case of this was studied in [8]—they compute a movement schedule but *do not allow* the creation of new copies of any data object. It addresses only the data *movement* problem. (So for example, one cannot create extra copies of any data item, but can just change on which disks they are stored.) The problem studied in [8] is formally defined as follows: given a set of disks, with each storing a subset of items and a specified set of move operations (each move operation specifies which data object needs to be moved from one disk to another), how do we schedule these move operations? If there are no storage constraints, then this is exactly the problem of edge-coloring the following multigraph. Create a graph that has a node corresponding to each disk and a directed edge corresponding to each move operation that is specified. Algorithms for edge-coloring

[*] This research was supported by NSF Award CCR-0113192.

[†] Computer Science Department and IMSC and ISI, University of Southern California. E-mail : leana@cs.usc.edu.

[‡] Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. E-mail : samir@cs.umd.edu.

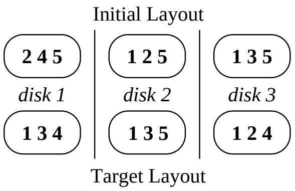
[§] Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : ykim@cs.umd.edu.

[¶] This work was done while this author was at the University of Maryland.

[||] Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : ycwan@cs.umd.edu.

[1] Utilization refers to the total number of clients that can be assigned to a disk that contains the data they want.

[2] This is not completely accurate and we will elaborate on this step later.



S_1={2,3}	D_1={1}
S_2={1,2}	D_2={3}
S_3={3}	D_3={1,2}
S_4={1}	D_4={3}
S_5={1,2,3}	D_5={}

Figure 1: An initial and target layouts as well as their corresponding S_i's and D_i's. Since item 5 has $D_5 = \emptyset$, we can just drop this item from consideration.

multigraphs can now be applied to produce a migration schedule since each color class represents a matching in the graph that can be scheduled simultaneously. Computing a solution with the minimum number of rounds is NP-hard, but several good approximation algorithms are available for edge coloring. With space constraints on the disk, the problem becomes challenging. In [8] it is shown that with the assumption that each disk has one spare unit of storage, very good constant factor approximations can be developed.

On the other hand, to handle high demand for popular objects, new copies will have to be dynamically created and stored on different disks. This means that we crucially need the ability to have a "copy" operation in addition to "move" operations. In fact, one of the crucial lower bounds used in the work on data migration [8] is based on a degree property of the multigraph. For example, if the degree of a node is δ, then this is a lower bound on the number of rounds that are required, since in each round at most one transfer operation involving this node may be done. For copying operations, clearly this lower bound is not valid. For example, suppose we have a single copy of a data item on a disk. Suppose we wish to create δ copies of this data item on δ distinct disks. Using the transfer graph approach, we could specify a "copy" operation from the source disk to each of the δ disks. Notice that this would take at least δ rounds. However, by using newly created copies as additional sources we can create δ copies in $\lceil \log(\delta + 1) \rceil$ rounds, as in the classic problem of broadcasting by using newly created copies as sources for the data object. (Essentially each copy spawns a new copy in each round.)

The *most general problem* of interest is the **data migration problem with cloning [11]** when data item i resides in a specified (source) subset S_i of disks and needs to be moved to a (destination) subset D_i. In other words, each data item that initially belongs to a subset of disks, needs to be moved to another subset of disks. (We might need to create new copies of this data item and store it on an additional set of disks.) Figure 1 depicts an example.

Different communication models can be considered based on how the disks are connected. We use the same model as in the work by [8, 1] 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 (a switched storage network with unbounded backplane bandwidth). These algorithms can also be extended to models where the size of the matching for each round is constrained. This can be done by a simple simulation, where we only choose a maximal subset of transfers to perform in each round.

This model best captures an architecture of parallel storage devices that are connected on a switched network with sufficient bandwidth and is most appropriate for our application.

1.1 The Correspondence Problem

Given a set of data objects placed on disks, we shall assume that what is important is the grouping of the data objects and not their exact location on each disk. For example, we can represent a disk by the set $\{A, B, C\}$ indicating that data objects A, B, and C are stored on this disk. If we move the location of these objects on the same disk, it does not affect the set corresponding to the disk in any way.

Data layout algorithms (such as the ones in [13, 14, 10, 7]) take as input a demand distribution for a set of data objects and outputs a grouping $S_{1'}, S_{2'}, \ldots S_{N'}$ as a desired data layout pattern on disks $1', 2', \ldots, N'$. (The current layout is assumed to be $S_1, S_2 \ldots S_N$.) Note that we do not need the data corresponding to the set of items $S_{1'}$ to be on (original) disk 1. For example the algorithm simply requires that a new grouping be obtained where the items in set $S_{1'}$ be grouped together on a disk. For example, if $S_3 = S_{1'}$ then by simply "renaming" disk 3 as disk $1'$ we have obtained a disk with the set of items $S_{1'}$, assuming that these two disks are inter-changeable. We need to compute a perfect matching between the initial and final layout sets. An edge is present between S_i and $S_{j'}$ if disk i has the same capabilities as disk j'. The weight of this edge is obtained by the number of "new items" that need to be moved to S_i to obtain $S_{j'}$. A minimum weight perfect matching in this graph gives the *correspondence* that minimizes the total number of changes, but *not* the number of rounds. Once we fix the correspondence, we need to invoke an algorithm to compute a migration schedule to minimize the number of rounds. Since this step involves solving an NP-hard problem, we will use a polynomial time approximation algorithm for computing the migration. However, we still need to pick a certain correspondence before we can invoke a data migration algorithm.

There are two central questions in which we are interested; these will be answered in Section 7.3:

- **Which correspondence algorithm should we use?** We will explore algorithms based on computing a matching of minimum total weight and matchings where we minimize the weight of the maximum weight edge. Moreover, the weight function will be based on estimates of how easy or difficult it is to obtain copies of certain data.

- **How good are our data migration algorithms once we fix a certain correspondence?** Even though we have bounds on the worst case performance of the algorithm, we would like to find whether or not its performance is a lot better than the worst case bound. (We do not have any example showing that the bound is tight.) In fact, it is possible that other heuristics perform extremely well, even though they do not have good worst case bounds [11].

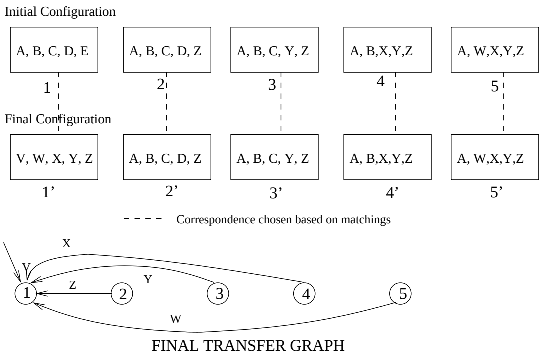


Figure 2: Figure to illustrate how a bad correspondence can yield an poor solution for data movement.

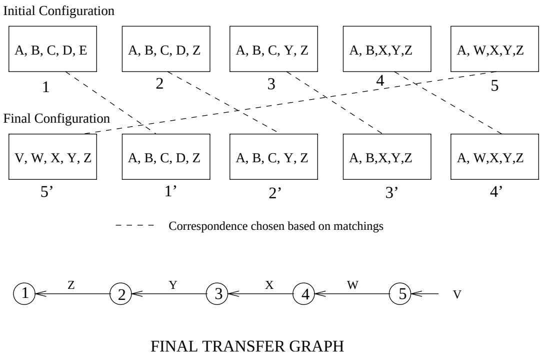


Figure 3: Figure to illustrate how a good correspondence can yield significantly better solutions for data movement.

For example, in Figure 2 we illustrate a situation where we have 5 disks with the initial and final configurations as shown. By picking the correspondence as shown, we end up with a situation where all the data on the first disk needs to be changed. We have shown the possible edges that can be chosen in the transfer graph along with the labels indicating the data items that we could choose to transfer from the source disk to the destination disk. The final transfer graph shown is a possible output of a data migration algorithm. This will take 5 rounds since all the data is coming to a single disk; node 1 will have a high in-degree. Item V can be obtained from tertiary storage, for example (or another device). Clearly, this set of copy operations will be slow and will take many rounds.

On the other hand, if we use the correspondence as shown by the dashed edges in Figure 3, we obtain a transfer graph where each disk needs only one new data item and such a transfer can be achieved in two rounds in parallel. (The set of transfers performed by the data migration algorithm are shown.)

1.2 Contributions

In recent work [11], it was shown that the data migration with cloning problem is NP-hard and has a polynomial time approximation algorithm with a worst case guarantee of 9.5. Moreover, the work also explored a few simple data migration algorithms. Some of the algorithms cannot provide constant approximation guarantee, while for some of the algorithms no approximation guarantee is known. In this paper, we conduct an extensive study of these data migration algorithms' performance under different changes in user access patterns. We also show that a good correspondence solution can improve the performance of the data migration algorithms by a factor of 2, relative to a bad solution. A more detailed observations and results of the study is given in Section 7.

2 Models and Definitions

In the *data migration problem*, we have N disks and Δ data items. For each item i, there is a subset of disks S_i and D_i. Initially only the disks in S_i have item i, and all disks in D_i want to receive i. Note that after a disk in D_i receives item i, it can be a source of item i for other disks in D_i which have not received the item yet. Our goal is to find a migration schedule using a minimum number of rounds, that is, to minimize the total amount of time to finish the data migration schedule. Our algorithms make use of known results on edge coloring of multigraphs. Given a graph G with max degree Δ_G and multiplicity μ, it is known that it can be colored with at most $\Delta_G + \mu$ colors.

3 Different Algorithms for the Correspondence Problem

To match disks in the initial layout with disks in the target layout, we tried the following methods:

Create a bipartite graph with two copies of disks.

1. (*Simple min max matching*) The weight of matching disk p in the initial layout with disk q in the target layout is the number of new items that disk q needs to get from other disks (because disks p does not have these items). Find a perfect matching that minimizes the maximum weight of the edges in the matching. Effectively it pairs disks in the initial layout to disks in the target layout, such that the number of items a disk needs to receive is

minimized.

2. (*Simple min sum matching*) Minimum weighted perfect matching using the weight function defined in (1). This method minimizes the total number of transfer operations.

3. (*Complex min sum matching*) Minimum weighted perfect matching with another weight function that takes the ease of obtaining an item into account. Note that the larger the ratio of $|D_i|$ to $|S_i|$ the more copying is required. Suppose disk p in the initial layout is matched with disk q in the target layout, and let S be the set of items that disk q needs which are not on disk p. The weight for matching these two disks is $\sum_{i \in S} \max(\log \frac{|D_i|}{|S_i|}, 1)$.

4. Direct correspondence. Disk i in the initial layout is always matched with disk i in the target layout.

5. Random permutation.

We found that using a matching-based correspondence method can improve the performance of all data migration algorithms by a factor of 2 if a bad correspondence is chosen. Moreover, we found that all matching-based correspondence methods are comparable to one another, while Direct Correspondence method performs well only when the initial layout and the target layout are similar. Therefore we think matching-based methods should be used, even though Direct correspondence method and Random permutation method run much faster than the matching-based methods. More detailed description of the results can be found in Section 7.1.

4 The Data Migration Algorithm

Define β_j as $|\{i | j \in D_i\}|$, i.e., the number of different sets D_i, that a disk j belongs to. We then define β as $\max_{j=1 \ldots N} \beta_j$. In other words, β is an upper bound on the number of items a disk may need.

Moreover, we may assume that $D_i \neq \emptyset$ and $D_i \cap S_i = \emptyset$. (We simply define the destination set D_i as the set of disks that need item i and do not currently have it. Thus in Fig. 1 item 5 will be dropped.)

Here we only give a high level description of the algorithm. We describe the details in Appendix A.

The basic idea behind the algorithm is to choose a collection of disjoint subsets $G_i \subseteq D_i$ for each item i. The algorithm then focuses on ensuring that all disks in G_i receive item i. The algorithm then uses the $|G_i| + 1$ copies of the items to deliver them to the set D_i using an edge coloring approach. The algorithm itself is slightly more involved because we need to choose initial sources for each item, without making one disk the source for too many items etc.

Basic Data Migration Algorithm.

1. (*Choose sources*) For an item i decide a unique source $s_i \in S_i$ so that $\alpha = \max_{j=1,\ldots,N}(|\{i | j = s_i\}| + \beta_j)$ is minimized. In other words, α is the maximum number of items for which a disk may be a source (s_i) or a destination.

2. (*Large destination sets*) Find a transfer graph for items such that $|D_i| \geq \beta$ as follows.

   (a) (*Choose representatives*) We first compute a disjoint collection of subsets $G_i, i = 1 \ldots \Delta$. Moreover, $G_i \subseteq D_i$ and $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$.

   (b) (*Send to representatives*) We have each item i sent to the set G_i.

   (c) (*From representatives to destination sets*) We now create a transfer graph as follows. Each disk is a node in the graph. We add directed edges from disks in G_i to $(\beta - 1)\lfloor \frac{|D_i|}{\beta} \rfloor$ disks in $D_i \setminus G_i$ such that the out-degree of each node in G_i is at most $\beta - 1$ and the in-degree of each node in $D_i \setminus G_i$ is 1. We redefine D_i as a set of $|D_i \setminus G_i| - (\beta - 1)\lfloor \frac{|D_i|}{\beta} \rfloor$ disks which do not receive item i so that they can be taken care of in Step 3. Note that the redefined set D_i has size $< \beta$.

3. (*Small destination sets*) Find a transfer graph for items such that $|D_i| < \beta$ as follows.

   (a) (*Find new sources in small sets*) For each item i, find a new source s'_i in D_i. A disk j can be a source s'_i for several items as long as $\sum_{i \in I_j} |D_i| \leq 2\beta - 1$ where I_j is a set of items of which j is a new source.

   (b) Send each item i from s_i to s'_i.

   (c) Create a transfer graph. We add a directed edge from the new source of item i to all disks in $D_i \setminus \{s'_i\}$.

4. We now find an edge coloring of the transfer graph obtained by merging two transfer graphs in Steps 2(c) and 3(c). The number of colors used is an upper bound on the number of rounds required to ensure that each disk in D_j gets item j.

There are several components needed to implement this algorithm.

1. Step 1: we use a network flow approach to find an optimum solution for α.

2. Step 2(a): we again use a network flow approach to find the sets G_i.

3. Step 2(b): to get an $O(1)$ approximation this step is quite complex (see Appendix A). We also use a simpler broadcasting scheme which makes the worst case bound $O(\log N)$.

4. Step 3(a): we use an algorithm for the generalized assignment problem [17].

5. Steps 3(b) and 4: we use an algorithm for edge-coloring multigraphs [2].

THEOREM 4.1. *(Khuller, Kim, and Wan [11]) The algorithm described above has a worst case approximation ratio of 9.5.*

5 Data Migration Algorithm Variants

We now describe the different data migration algorithms that we tried in the experiments.

1. 9.5-approximation algorithm for data migration [*Data Mig. (Basic)*]. This algorithm uses several complicated components to achieve constant factor approximation guarantee. We consider simpler variants of these components. The variants may not give good theoretical bounds.

   (a) in Step 2(a) (*Choose representatives*) we find the minimum integer m such that there exist disjoint set G_i of size $\lfloor \frac{|D_i|}{m} \rfloor$. The value of m should be between $\bar{\beta} = \sum_{i=1}^{N} \frac{\beta_i}{N}$ and β.

   (b) in Step 2(b) (*Send to representatives*) we use a simple doubling method to satisfy all requests in G_i. Since all groups are disjoint, it takes $\max_i \log |G_i|$ rounds.

   (c) in Step 3 (*Small destination sets*) we do not find a new source s'_i. Instead S_i send item i to D_i directly for small sets. We try to find a schedule that minimizes the maximum total degree of disks in the final transfer graph in Step 4.

   (d) in Step 3(a) (*Find new sources in small sets*) when we find new source s'_i, S_i can be candidates as well as D_i. If $s'_i \in S_i$, then we can save some rounds to send item i from s_i to s'_i.

   The worst case time complexity of all of the above algorithms, except for variant (c), is $O((n^2 + \Delta)n^2\beta \log \frac{(n^2+\Delta)^2}{n^2\beta})$. The worst case time complexity of variant (c), which does not find new sources s'_i, is $O((n + \Delta)n\Delta \log \frac{(n+\Delta)^2}{n\Delta} \log \Delta)$.

2. Edge-coloring on a transfer graph [*Edge Coloring*]. We can find a transfer graph, given the initial and target layout. Find an edge coloring of the transfer graph to obtain a valid schedule, and the number of colors used is an upper bound on the total number of rounds. The worst case time complexity here is $O((n + \Delta)n\beta \log \frac{(n+\Delta)^2}{n\beta} + n^2\beta^2)$.

3. Heuristics using unweighted matching [*Unweighted Matching*]. Repeatedly remove an unweighted matching from the transfer graph. Its worst case time complexity is $O(n^4\beta)$.

4. Heuristics using weighted matching [*Weighted Matching*]. Repeatedly remove a weighted matching from the transfer graph, where the weight between disk v and w is $\max_i(1 + \log \frac{|D_i|}{|S_i|})$, over all items i where $v \in S_i, w \in D_i$, or $w \in S_i, v \in D_i$. The worst case time complexity here is $O(n^4\beta)$.

5. Broadcasting items one by one [*Item by Item*]. We process each item i sequentially and satisfy the demand by doubling the number of copies of an item in each round. The worst case time complexity here is $O(n\beta)$.

6 Experimental Framework and Parameters

The framework of our experiments is as follows:

1. Run the sliding window algorithm [7] to create an initial layout, given the number of user requests for each data object. In Section 6.1 we describe the distributions we used in generating user requests. These distributions are completely specified once we fix the ordering of data objects in order of decreasing demand.

2. Shuffle the ranking of items. Generate the new demand for each item according to the probabilities corresponding to the new ranking of the item. To obtain a target layout, take one of the following approaches.

   (a) Run the sliding window algorithm again with the new request demands.

   (b) Use other (than sliding window) methods to create a target layout. The motivation for exploring these methods is (a) performance issues (as explained later in the paper) as well as (b) that other algorithms (other than sliding window) could be useful for creating layouts. The methods considered here are as follows.

      i. Rotation of items: Suppose we numbered the items in non-increasing order of the number of copies in the initial layout. We make a sorted list of items of size $k = \lfloor \frac{\Delta}{50} \rfloor$, and let the list be $l_1, l_2, \ldots, l_k$. Item l_i in the target layout will occupy the space of item l_{i+1} in the initial layout, while item l_k in the target layout will occupy the positions of item l_1 in the initial layout. In other words, number of copies of items $l_1, \ldots, l_{k-1}$ are decreased slightly, while the number of copies of item l_k is increased significantly.

      ii. Enlarging D_i for items with small S_i: Repeat the following $\lfloor \frac{\Delta}{20} \rfloor$ times. Pick an item s

randomly having only one copy in the current layout. For each item i that has more than one copy in the current layout, there is a probability of 0.5 that item i will randomly give up the space of one of its copies, and the space will be allocated to item s in the new layout for the next iteration. In other words, if there are k items having more than one copy at the beginning of this iteration, then item s is expected to gain $\frac{k}{2}$ copies at the end of the iteration.

3. Run different correspondence algorithms mentioned in Section 3 to match a disk in the initial layout with a disk in the target layout. Now we can find the set of source disks and destination disks for each item.

4. Run different data migration algorithms, and record the number of rounds needed to finish the migration.

6.1 User Request Distributions

We generate the number of requests for different data objects using a Zipf distribution and a Geometric distribution. 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 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 [12]:

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

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

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, e.g., similar to the *measurements* performed in [4]. We assign θ to be 0 and 0.5 in our experiments below.

Geometric Distribution

We also tried a geometric distribution in order to investigate how a more skewed distribution affects the performance of the data migration algorithm. The distribution is defined as follows:

$$\text{Prob(request for movie } i) = (1-p)^{i-1}p \quad \begin{array}{c} \forall i = 1, \ldots, M \\ \text{and} \\ 0 < p < 1 \end{array}$$

where we use p set to 0.25 and 0.5 in our experiments below.

6.2 Shuffling methods

1. Randomly promote 20% of the items. For each chosen item of rank i, we promote it to rank 1 to $i-1$, randomly.

2. Promote the least popular item to the top, and demote all other items by one rank.

6.3 Parameters and Layout Creation

We now describe the parameters used in the experiments, namely the number of disks, space capacity, and load capacity. We ran a number of experiments with 60 disks. For each correspondence method, user request distribution, and shuffling method, we generated 20 inputs (i.e., 20 sets of initial and target layouts) for each set of parameters, and ran different data migration algorithms on those instances. In the Zipf distribution, we used θ values of 0 and 0.5, and in the Geometric distribution, we assigned p values of 0.25 and 0.5.

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. We obtained these numbers from the specifications of the latest SCSI hard drives. For example, the latest 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 the MPEG-4 format. So a disk is capable of storing more movies and supporting more streams.

We show the result of 5 different layout creation schemes by different combinations of methods and parameters to create the initial and target layout. (I): Promoting the last item to the top, Zipf distribution ($\theta = 0$); (II): Promoting 20% of items, Zipf distribution ($\theta = 0$); (III): Promoting the last item to the top, Geometric distribution ($p = 0.5$); (IV): Initial layout obtained from the Zipf distribution ($\theta = 0$). Target layout obtained from the method described in Step 2(b)i in Section 6 (rotation of items); (V): Initial layout obtained from the Zipf distribution ($\theta = 0$). Target layout obtained from the method described in Step 2(b)ii in Section 6 (enlarging D_i for items with small S_i). We also tried other combinations, but only the results of the above 5 combinations are shown.

7 Results

In the tables below we present the average for 20 inputs. Moreover, we present results of two representative inputs individually, to illustrate the performance of the algorithms under the same initial and target layouts. This presentation is motivated as follows. The *absolute* performance of each run is largely a function of the differences between the initial and the target layouts (and this is true for all algorithms). That is, a small difference is likely to result in relatively few rounds needed for data migration, and a large difference is likely to result in relatively many rounds needed for data migration. Since a goal of this study is to understand the *relative* performance differences between the algorithms described above, i.e., given the same initial and target layouts, we believe that presenting the data on a per run basis is more informative (given our goal). That is, considering the average alone somewhat obscures the characteristics of the different algorithms.

7.1 Different correspondence methods

We first investigate how different correspondence methods affect the performance of the data migration algorithms. Figure 4 and Figure 5 show the ratio of the number of rounds taken by different data migration algorithms to the lower bounds, averaged over 20 inputs under parameter setting (A). We observed that the simple min max matching (1) always returns the same matching as the simple min sum matching (2) in all instances we tried. Moreover, using a simpler weight function (2) or a more involved one (3) does not seem to affect the behavior in any significant way (often these matchings are the same). Thus we only present results using simple min max matching, and label it as matching-based correspondence method.

From the figures, we found that using a matching-based method is important and can affect the performance of all algorithms by up to a factor of 4.4 if a bad correspondence, using a random permutation for example, is chosen. Since direct correspondence does not perform as well as other weight-based matchings, this also suggests that a good correspondence method is important. However, the performance of direct correspondence was reasonable when we promoted the popularity of one item. This can be explained by the fact that in this case sliding window obtains a target layout which is fairly similar to the initial layout.

7.2 Different data migration algorithms

From the previous section it is reasonable to evaluate the performance of different data migration algorithms using only the simple min max matching corresponding method. We experimented with different variants of the 9.5-approximation algorithm for data migration, because we found that the schedule spends a significant number of rounds in certain steps. However, there exist simpler methods to perform the same steps that may not guarantee a constant approximation. Consider algorithm Data Mig. (c) which modifies Step 3 (where we want to satisfy small D_i), we found that sending the items from S_i to small D_i directly using edge coloring, without using new sources s'_i, is a much better idea. Even though this makes the algorithm an $O(\Delta)$ approximation algorithm, the performance is very good under both the Zipf and the Geometric distributions, since the sources are not concentrated on one disk and only a few items require rapid doubling.

In addition, we thought that making the sets G_i slightly larger by using $\overline{\underline{\beta}}$ was a better idea (i.e., algorithm Data Mig. (a) which modifies Step 2(a)). This reduces the average degree of nodes in later steps such as in Step 2(c) and Step 3 where we send the item to disks in $D_i \setminus G_i$. However, the experiments shown it usually performs slightly worse than the algorithm using β.

Consider algorithm Data Mig. (d) which modifies Step 3(a) (where we identify new sources s'_i), we found that the performance of the variant that includes S_i, in addition to D_i, as candidates for the new source s'_i is mixed. Sometimes it is better than the original 9.5-approximation algorithm, but more often it is worse.

Consider algorithm Data Mig. (b) which modifies Step 2(b) (where we send the items from the sources S_i to G_i), we found that doing a simple broadcast is generally a better idea, as we can see from the results in Parameter Setting (A), Instance 1 in Table 4 and in Parameter Setting (A), Instance 1 in Table 5. Even though this makes the algorithm an $O(\log n)$ approximation algorithm, very rarely is the size of $\max G_i$ large. Under the input generated by Zipf and Geometric distributions, the simple broadcast generally performs the same as the original data migration algorithm since the size of $\max G_i$ is very small.

Out of all the heuristics, the matching-based heuristics perform very well in practice. The only cases where they perform extremely badly correspond to hand-crafted (by us) bad examples. Suppose, for example, a set of Δ disks are the sources for Δ items (each disk has all Δ items). Suppose further that the destination disks also have size Δ each and are disjoint. The result is listed in Figure 7 and Table 1. Our algorithm sends each of the items to one disk in D_i in the very first round. After that, a broadcast can be done in the destination sets as they are disjoint, which takes $O(\log \Delta)$ rounds in total. Therefore the ratio of the number of rounds used to the lower bound remains a constant. The matching-based algorithm can take up to Δ rounds, as it can focus on sending item i at each round by computing a perfect matching of size Δ between the source disks and the destination disks for item i in each round. Since any perfect matching costs the same weight in this case, the matching focuses on sending only one item in each round. We implemented a variant of the weighted matching heuristic to get around this problem by adding a very small random weight to each edge in the graph. As we can see from Figure 7 and Table 1, although this variant does not perform as well as our migration algorithms, it performs much better than other matching-based data migration algorithms. Moreover, we ran this variant with the inputs generated from Zipf and Geometric distributions, and we found that it frequently takes the same number of rounds as the weighted matching heuristic. In some cases it performs better than the weighted matching heuristic, while in a few cases its performance is worse.

Given the performance of different data migration algorithms illustrated in Figure 6 and in the tables, the two matching-based heuristics are comparable. Matching-based heuristics perform the best in general, then the edge-coloring heuristic, then the data migration algorithms, while processing items one-by-one comes last. The main reason why edge-coloring heuristic performs better than the 9.5-approximation data migration algorithm is because the input contains mostly move operations, i.e., the size of S_i and D_i is at most 2 for at least 80% of the items. The Zipf distribution does not provide enough cloning operations for the data migration algorithm to show its true potential (the sizes of G_i are mostly zero, with one or two items having size of 1 or 2). Processing items one by one is bad because we have a lot of items (more than 300 in Parameter Setting (A)), but most of the operations can be done in parallel (we have 60 disks, meaning that we can support 30 copy operations in one round). Under the Zipf distribution, since the sizes of most sets G_i are zero, the data migration variant that send items from S_i directly to D_i is essentially the same as the coloring heuristic. Thus they have almost the same performance.

Under different input parameters

In additional to the Zipf distribution, we also tried the Geometric distribution because we would like to investigate the performance of the algorithms under more skewed distributions where more cloning of items is necessary. As we can see in Figure 5 and Table 4, we found that the performance of the coloring heuristic is worse than our data migration algorithms, especially when p is large (more skewed) or when the ratio of the load capacity to space capacity is high. However, the matching-based heuristics still perform the best.

We also investigated the effect on the performance of different algorithms under higher ratio of the load capacity to space capacity. We found that the results are qualitatively similar, and thus we omit them here.

Moreover, in the Zipf distribution, we assigned different values of θ, which controls the skewness of the distribution, as 0.0 and 0.5. We found that the results are similar in both cases. While in the Geometric distribution, a higher value of p (0.5 vs 0.25) gives our data migration algorithms an advantage over coloring heuristics as more cloning is necessary.

Miscellaneous

Tables 5 and 6 show the performance of different algorithms using inputs where the target layout is derived from the initial layout, as described in Step 2(b)i and Step 2(b)ii in Section 6. Note that when the initial layout and the target layout are very similar, the data migration can be done very quickly. The number of rounds taken is much fewer than the number of rounds taken using inputs generated from running the sliding window algorithm twice. This illustrates that it may be worthwhile to consider developing an algorithm which takes in an initial layout and the new access pattern, and then derives a target layout, with the optimization of the data migration process in mind. Developing these types of algorithms and evaluating their performance characteristics is part of our future efforts.

We now consider the running time of the different data migration algorithms. Except for the matching heuristics, all other algorithms' running times are at most 3 CPU seconds and often less than 1 CPU second, on a Sun Fire V880 server. The running time of the matching heuristics depends on the total number of items. It can be as high as 43 CPU seconds when the number of items is around 3500, and it can be lower than 2 CPU seconds when the number of items is around 500.

We also collected the maximum space requirement for each disk needed to complete the migration. Consider disk 3 in Figure 1 and suppose that another disk needs item 3 from disk 3. If disk 3 receives items 2 and 4 before sending out item 3, then its temporary maximum space requirement is 4. However, since all data migration algorithms listed in this paper do not optimize for the temporary maximum space requirement, in many instances, there exists a disk that needs twice the capacity of that disk to finish the migration. We omit the details of the maximum space requirements results due to space limitation.

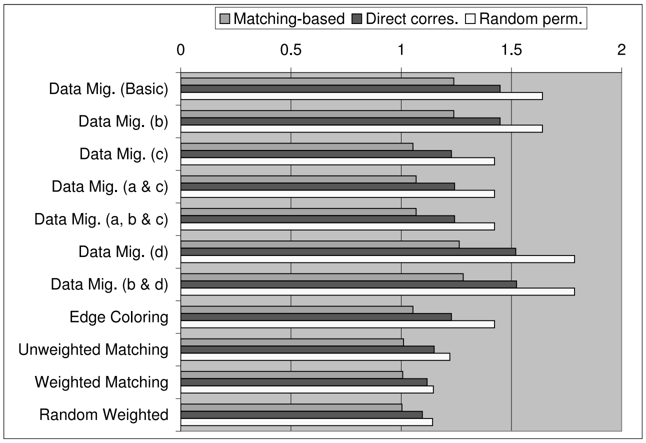


Figure 4: The ratio of the number of rounds taken by the algorithms to the lower bound (28.1 rounds), averaged over 20 inputs, using parameter setting (A) and layout creation scheme (I) (i.e., with 60 disks, space cap of 15, load cap of 40, promoting the last item to the top, and user requests following the Zipf distribution ($\theta = 0$)). The effect under different correspondence methods is shown.

7.3 Final Conclusions

For the correspondence problem question posed in Section 1.1, our experiments indicate that weighted matching

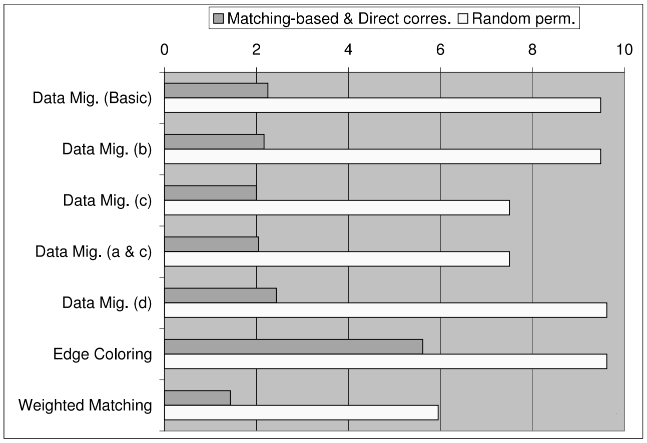


Figure 5: The ratio of the number of rounds taken by the algorithms to the lower bound (6.0 rounds), averaged over 20 inputs, using parameter setting (A) and layout creation scheme (II) (i.e., with 60 disks, space cap of 15, load cap of 40, promoting the last item to the top, and user requests following the Geometric distribution ($p = 0.5$)). The effect under different correspondence methods is shown.

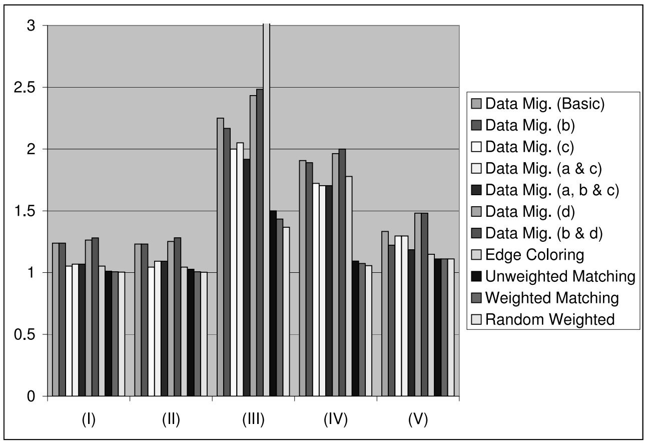


Figure 6: The ratio of the number of rounds taken by the algorithms to the lower bound, averaged over 20 inputs, using min max matching corresponding method and parameter setting (A), under different layout creation schemes (see Section 6.3).

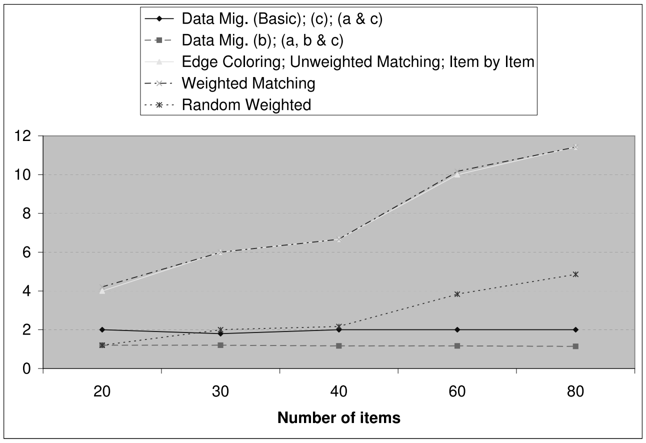


Figure 7: The ratio of the number of rounds taken by the algorithms to the lower bound under the worst case (i.e., when a set of Δ disks are the sources for Δ items (each disk has all Δ items), and the destination disks also have size Δ each and are disjoint).

is the best approach among the ones we tried.

For the data migration problem question posed in Section 1.1, our experiments indicate that the weighted matching heuristic with some randomness does very well. This suggests that perhaps a variation of matching can be used to obtain an $O(1)$ approximation as well. Among all variants of the 9.5-approximation data migration algorithms, letting S_i send item i to D_i directly for the small sets, i.e. variant (c), performs the best. From the above described results we can conclude that under the Zipf and Geometric distributions, where cloning does not occur frequently, the weighted matching heuristic returns a schedule which requires only a few rounds more than the optimal. All variants of the 9.5-approximation data migration algorithms usually take no more than 10 rounds more than the optimal, when a good correspondence method is used.

Table 1: The number of rounds taken by different data migration algorithms, when a set of Δ disks are the sources for Δ items (each disk has all Δ items), and the destination disks also have size Δ each and are disjoint.

Number of items (Δ):	20	30	40	60	80
Lower Bound	5	5	6	6	7
Data Mig. (Basic)	10	9	12	12	14
Data Mig. ((b) doubling)	6	6	7	7	8
Data Mig. ((c) S_i to D_i)	10	9	12	12	14
Data Mig. ((a) $\bar{\beta}$, (c) S_i to D_i)	10	9	12	12	14
Edge Coloring	20	30	40	60	80
Unweighted Matching	20	30	40	60	80
Weighted Matching	21	30	40	61	80
Random Weighted	6	10	13	23	34
Item by Item	20	30	40	60	80

References

Table 2: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching corresponding method with layout creation scheme (I) (i.e., promoting the last item to the top, with user requests following the Zipf distribution ($\theta = 0$)), and under different parameter settings. The first 2 instances and the average over 20 instances are shown.

Parameter setting:	(A)			(B)			(C)		
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
Data Mig. (Basic)	1.233	1.233	1.238	1.130	1.104	1.122	1.055	1.108	1.096
Data Mig. (b)	1.233	1.233	1.238	1.130	1.104	1.122	1.055	1.108	1.096
Data Mig. (c)	1.000	1.033	1.053	1.000	1.000	1.006	1.000	1.092	1.044
Data Mig. (a & c)	1.033	1.067	1.068	1.022	1.021	1.021	1.000	1.092	1.044
Data Mig. (a, b & c)	1.033	1.067	1.068	1.022	1.021	1.021	1.000	1.092	1.044
Data Mig. (d)	1.000	1.300	1.263	1.000	1.000	1.014	1.191	1.292	1.169
Data Mig. (b & d)	1.133	1.167	1.281	1.022	1.021	1.037	1.191	1.292	1.170
Edge Coloring	1.000	1.033	1.053	1.000	1.000	1.006	1.000	1.092	1.044
Unweighted Matching	1.000	1.000	1.011	1.000	1.000	1.000	1.000	1.031	1.009
Weighted Matching	1.000	1.000	1.007	1.000	1.000	1.000	1.000	1.015	1.006
Random Weighted	1.000	1.000	1.004	1.000	1.000	1.000	1.000	1.015	1.008
Item by Item	11.100	10.567	12.313	6.522	7.188	7.822	8.455	7.646	7.654

Table 3: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching corresponding method with layout creation scheme (II) (i.e., promoting 20% of items, with user requests following the Zipf distribution ($\theta = 0$)), and under different parameter settings. The first 2 instances and the average over 20 instances are shown.

Parameter setting:	(A)			(B)			(C)		
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
Data Mig. (Basic)	1.267	1.233	1.231	1.100	1.121	1.092	1.042	1.059	1.048
Data Mig. (b)	1.267	1.233	1.231	1.100	1.121	1.092	1.042	1.059	1.048
Data Mig. (c)	1.033	1.033	1.044	1.000	1.017	1.008	1.008	1.008	1.013
Data Mig. (a & c)	1.067	1.300	1.092	1.000	1.017	1.008	1.008	1.008	1.013
Data Mig. (a, b & c)	1.067	1.300	1.092	1.000	1.017	1.008	1.008	1.008	1.013
Data Mig. (d)	1.367	1.167	1.252	1.167	1.207	1.149	1.203	1.263	1.201
Data Mig. (b & d)	1.367	1.267	1.282	1.167	1.207	1.149	1.203	1.263	1.201
Edge Coloring	1.033	1.033	1.044	1.000	1.017	1.008	1.008	1.008	1.013
Unweighted Matching	1.067	1.000	1.027	1.033	1.034	1.023	1.017	1.025	1.015
Weighted Matching	1.033	1.000	1.007	1.033	1.017	1.003	1.008	1.000	1.003
Random Weighted	1.000	1.000	1.003	1.000	1.017	1.002	1.000	1.000	1.000
Item by Item	16.867	16.067	16.639	14.283	15.345	14.072	15.839	15.686	15.566

[1] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan and J. Wilkes. An Experimental Study of Data Migration Algorithms. *Workshop on Algorithm Engineering*, 2001

[2] C. Berge and J.C. Fournier. A Short Proof for a Generalization of Vizing's Theorem. *Journal of Graph Theory*, Vol 15(3):333–336 (1991).

[3] J. A. Bondy and U. S. R. Murty. Graph Theory with applications. *American Elsevier*, New York, 1977.

[4] A. L. Chervenak. Tertiary Storage: An Evaluation of New Applications. *Ph.D. Thesis, UC Berkeley*, 1994.

[5] C.-F. Chou, L. Golubchik, J. C. S. Lui and I.-H. Chung. Design of Scalable Continuous Media Servers. *Special issue on QoS of Multimedia Tools and Applications*, 17(2-3):181–212, 2002.

[6] S. Ghandeharizadeh and R. R. Muntz. Design and Implementation of Scalable Continuous Media Servers. *Parallel Computing Journal*, 24(1):91–122, 1998.

[7] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella and A. Zhu. Approximation Algorithms for Data Placement on Parallel Disks. *Proc. of ACM-SIAM SODA*, 2000.

[8] J. Hall, J. Hartline, A. Karlin, J. Saia and J. Wilkes. On Algorithms for Efficient Data Migration. *Proc. of ACM-SIAM SODA*, 620–629, 2001.

[9] I. Holyer. The NP-Completeness of Edge-Coloring. *SIAM J. on Computing*, 10(4):718–720, 1981.

[10] S. Kashyap and S. Khuller. Algorithms for Non-Uniform Size Data Placement on Parallel Disks. *Conference on Foundations of Software technology and Theoretical Computer Science* (FST&TCS), 2003.

[11] S. Khuller, Y. Kim and Y-C. Wan. Algorithms for Data Migration with Cloning. *22nd ACM Symposium on Principles of Database Systems* (PODS), 27–36, 2003.

[12] D. E. Knuth. *The Art of Computer Programming, Volume 3*. Addison-Wesley, 1973.

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

[14] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Proc. of Workshop on Approximation Algorithms*, 2000.

[15] H. Shachnai and T. Tamir. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. *Proc. of Workshop on Approximation Algorithms*, 2003.

[16] C.E. Shannon. A theorem on colouring lines of a network. *J. Math. Phys.*, 28:148–151, 1949.

[17] D.B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem *Mathematical Programming*, A 62, 461–474, 1993.

Table 4: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching corresponding method with layout creation scheme (III) (i.e., promoting the last item to the top, with user requests following the Geometric distribution ($p = 0.5$)), and under different parameter settings. The first 2 instances and the average over 20 instances are shown.

Parameter setting:	(A)			(B)		
Instance:	1	2	Ave	1	2	Ave
Data Mig. (Basic)	2.000	2.167	2.250	1.611	1.611	1.568
Data Mig. (b)	1.875	2.167	2.167	1.611	1.611	1.568
Data Mig. (c)	1.625	2.000	2.000	1.444	1.222	1.286
Data Mig. (a & c)	1.750	2.000	2.050	1.333	1.333	1.296
Data Mig. (a, b & c)	1.625	2.000	1.917	1.278	1.278	1.261
Data Mig. (d)	1.750	2.333	2.433	1.722	1.500	1.533
Data Mig. (b & d)	1.875	2.333	2.483	1.556	1.556	1.538
Edge Coloring	3.875	5.667	5.617	2.000	2.111	1.980
Unweighted Matching	1.000	1.500	1.500	1.111	1.111	1.116
Weighted Matching	1.000	1.500	1.433	1.056	1.111	1.111
Random Weighted	1.000	1.333	1.367	1.056	1.056	1.010
Item by Item	6.250	12.833	12.683	3.667	3.667	5.719

Table 5: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching corresponding method with layout creation scheme (IV) (i.e., target layout obtained from the method described in Step 2(b)i in Section 6 (rotation of items)), and under different parameter settings. The first 2 instances and the average over 20 instances are shown.

Parameter setting:	(A)			(B)			(C)		
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
Data Mig. (Basic)	2.400	2.000	1.907	1.417	1.444	1.553	1.333	1.250	1.330
Data Mig. (b)	2.200	2.000	1.889	1.417	1.444	1.553	1.333	1.250	1.330
Data Mig. (c)	2.000	1.600	1.722	1.167	1.111	1.289	1.222	1.000	1.107
Data Mig. (a & c)	1.800	2.000	1.704	1.250	1.333	1.408	1.222	1.083	1.170
Data Mig. (a, b & c)	2.000	2.000	1.704	1.333	1.333	1.408	1.222	1.083	1.170
Data Mig. (d)	2.400	2.400	1.963	1.333	1.444	1.513	1.556	1.083	1.348
Data Mig. (b & d)	2.200	2.200	2.000	1.250	1.667	1.658	1.556	1.333	1.393
Edge Coloring	1.800	2.000	1.778	1.000	1.000	1.250	1.222	1.000	1.125
Unweighted Matching	1.000	1.000	1.093	1.000	1.000	1.026	1.000	1.000	1.000
Weighted Matching	1.000	1.200	1.074	1.000	1.000	1.013	1.000	1.000	1.009
Random Weighted	1.000	1.200	1.056	1.000	1.000	1.013	1.000	1.000	1.009
Item by Item	10.000	9.800	8.889	6.333	8.111	9.592	15.778	12.000	12.536

[18] V. G. Vizing. On an estimate of the chromatic class of a p-graph (Russian). *Diskret. Analiz.* 3:25–30, 1964.

[19] J. Wolf, H. Shachnai and P. Yu. DASD Dancing: A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems. *ACM SIGMETRICS/Performance Conf.*, 157–166, 1995.

Appendices

A Details of the algorithm

In this appendix, we describe the details of our migration algorithm briefly outlined in Section 4.

1. Step 1: We find a source $s_i \in S_i$ for each item i so that $\max_{j=1,\dots,N}(|\{i|j = s_i\}| + \beta_j)$ is minimized, using a flow network as follows. We create a flow network with a source s and a sink t as shown in Figure 8. We have two set of nodes corresponding to disks and items. Add directed edges from s to nodes for items and also directed edges from item i to disk j if $j \in S_i$. The capacities of all those edges are one. Finally we add an edge from the node corresponding to disk j to t with capacity $\alpha - \beta_j$. We want to find the minimum α so that the maximum flow of the network is Δ. We can do this by checking if there is a flow of Δ with α starting from $\max \beta_j$ and increasing by one until it is satisfied. If there is outgoing flow from item i to disk j, then we set j as s_i.

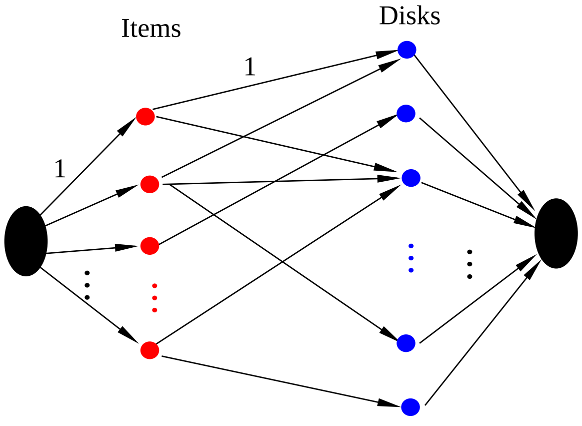


Figure 8: Flow network to find α

Table 6: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching corresponding method with layout creation scheme (V) (i.e., target layout obtained from the method described in Step 2(b)ii in Section 6 (enlarging D_i for items with small S_i)), and under different parameter settings. The first 2 instances and the average over 20 instances are shown.

Parameter setting:		(A)			(B)			(C)	
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
Data Mig. (Basic)	1.000	1.333	1.333	1.000	1.000	1.114	1.167	1.000	1.140
Data Mig. (b)	1.000	1.333	1.222	1.000	1.000	1.114	1.167	1.000	1.140
Data Mig. (c)	1.000	1.333	1.296	1.000	1.000	1.086	1.000	1.000	1.093
Data Mig. (a & c)	1.000	1.333	1.296	1.000	1.000	1.086	1.167	1.000	1.116
Data Mig. (a, b & c)	1.000	1.333	1.185	1.000	1.000	1.086	1.167	1.000	1.093
Data Mig. (d)	1.000	1.667	1.481	1.000	1.500	1.286	1.500	1.750	1.488
Data Mig. (b & d)	1.000	1.667	1.481	1.000	1.500	1.286	1.667	1.750	1.512
Edge Coloring	1.000	1.000	1.148	1.000	1.000	1.057	1.000	1.000	1.047
Unweighted Matching	1.000	1.000	1.111	1.000	1.000	1.029	1.000	1.000	1.047
Weighted Matching	1.000	1.000	1.111	1.000	1.000	1.029	1.000	1.000	1.047
Random Weighted	1.000	1.000	1.111	1.000	1.000	1.029	1.000	1.000	1.047
Item by Item	7.000	5.000	5.815	15.000	7.750	8.200	8.833	11.750	11.349

2. Step 2(a): We choose disjoint sets G_i for each $i = 1 \dots \Delta$, again using a network flow approach. As shown in Figure 9, we create a flow network with a source s and sink t. In addition we have two sets of vertices U and W. The first set U has Δ nodes, each corresponding to a disk that is the source of an item. The set W has N nodes, each corresponding to a disk in the system. We add directed edges from s to each node in U, such that the edge (s, i) has capacity $\lfloor \frac{|D_i|}{\beta} \rfloor$. We also add directed edges with infinite capacity from node $i \in U$ to $j \in W$ if $j \in D_i$. We add unit capacity edges from nodes in W to t. We find a max-flow from s to t in this network. An integral max flow in this network will correspond to $|G_i|$ units of flow going from s to i, and from i to a subset of vertices in D_i before reaching t. The vertices to which i has non-zero flow will form the set G_i.

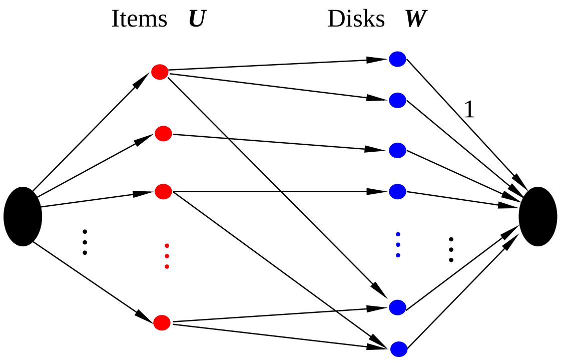


Figure 9: Flow network to find G_i

3. Step 2(b): The simple solution would be to broadcast the data to each group G_i from the chosen source, since the groups are disjoint. However, this broadcast takes at least $\max_i \log |G_i|$ rounds. Unfortunately, this would give us an $O(\log N)$ approximation guarantee. The method described below, develops stronger lower bounds for this situation.

   Let M be the number of steps required to send all items i to all disks in G_i in an optimal schedule of Step 2(b). To find a lower bound for M, we construct the following flow network F_m (parameterized by an integer m) as shown in Figure 10. We have a source s and two sets of nodes U and V. U has $N \cdot m$ nodes $x_{jk} (j = 1 \dots N, k = 1 \dots m)$. V has Δ nodes $y_i (i = 1 \dots \Delta)$ and y_i has demand $|G_i|$. There is an edge e_{ijk} from x_{jk} to y_i and its capacity c_{ijk} is 2^{m-k} if a disk j has item i initially. There are edges from s to nodes x_{jk} in U with capacity 2^{m-k}. If m' be the smallest number such that we can construct a solution of $F_{m'}$ that satisfies all demands $|G_i|$, then $M \geq m'$.

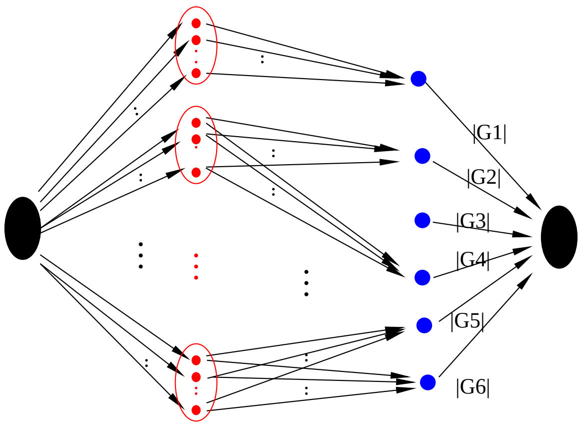


Figure 10: An example of constructing F_m where $\Delta = 6$

   The solution of the flow network $F_{m'}$ may not correspond to a valid schedule since a node x_{jk} may send flow to several nodes. we convert the solution to a solution satisfying the following properties.

- node x_{jk} sends flow to at most one node in V.
- the solution satisfies at least $|G_i| - 2^{m'-1}$ demands for each item i.

First, we define a variable z_{ijk} for an edge from x_{jk} to y_i and set $z_{ijk} = f_{ijk}/c_{ijk}$ where f_{ijk} is the flow through e_{ijk} in solution $F_{m'}$. We substitute nodes $y_{il}(l = 1 \ldots \lfloor \sum_{j,k} z_{ijk} \rfloor)$ for each node y_i in V. We distribute edges having nonzero flow to y_i as follows. Sort edges in non-increasing order of their capacities. Assign edges to y_{i1} until the sum of z values of assigned edges is greater than or equal to one. If the sum is greater than one, we split the last edge (denote as $e_{ij'k'}$) into $e_{ij'k'_1}$ and $e_{ij'k'_2}$. Assign $e_{ij'k'_1}$ to y_{i1} and define $z_{ij'k'_1}$ so that the sum of z values of edges assigned to y_{i1} is exactly one. Set $z_{ij'k'_2} = z_{ij'k'} - z_{ij'k'_1}$. We repeat this so that for all nodes y_{il}, the sums of z values of the assigned edges are one. In the resulting bipartite graph with U and $V' = \{y_{il}\}$, z makes a *fractional* matching which matches all vertices in V'. Therefore, we can find an integral matching that matches all vertices in V' and the matching satisfies the first property in the lemma. Now we merge nodes y_{il} into y_i. Then each y_i matches exactly $\lfloor \sum_{j,k} z_{ijk} \rfloor$ edges.

Now Step 2(b) can be done in $\alpha + 2m' + 1$ rounds based on the above solution. First we choose $\min(\lfloor \sum_{j,k} z_{ijk} \rfloor + 1, |G_i|)$ disks in G_i and denote those disks as H_i. Disk j sends item i to a disk in H_i if edge e_{ijk} is matched for some k. If $|H_i| > \lfloor \sum_{j,k} z_{ijk} \rfloor$, there is one disk in H_i which cannot receive item i. The disk receives item i from s_i. It can be done in $m' + \alpha + 2$ rounds. Since $|G_i|/|H_i| \leq 2^{m'-1}$, we can make all disks in G_i have item i in additional $m' - 1$ rounds.

4. Step 3(a): the following result from Shmoys-Tardos [17] will be used for this step.

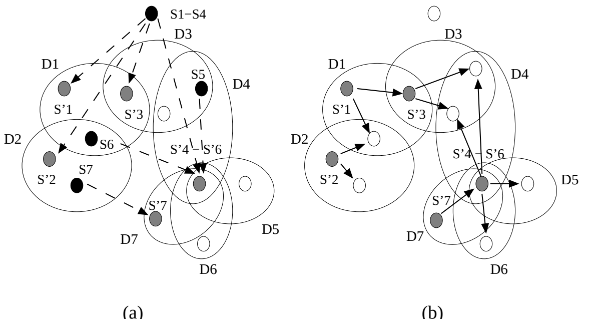


Figure 11: An example of Step 3 where $\alpha = 4$ and $\beta = 4$. **(a)** migration from s_i to s'_i **(b)** migration from s'_i to $D_i \setminus \{s'_i\}$.

THEOREM A.1. *(Shmoys-Tardos [17]) We are given a collection of jobs $\mathcal{J}$, each of which is to be assigned to exactly one machine among the set $\mathcal{M}$; if job $j \in \mathcal{J}$ is assigned to machine $i \in \mathcal{M}$, then it requires p_{ij} units of processing time, and incurs a cost c_{ij}. Suppose that there exists a fractional solution (that is, a job can be assigned fractionally to machines) with makespan P and total cost C. Then in polynomial time we can find a schedule with makespan $P + \max p_{ij}$ and total cost C.*

For each item i we wish to choose a source disk s'_i such that the following properties hold (I_j denotes the set of items for which disk j is a source).

- If $i \in I_j$ then $j \in D_i$.
- $\sum_{i \in I_j} |D_i| \leq 2\beta - 1$.

We create an instance of the problem of scheduling machines with costs. Items correspond to jobs and disks correspond to machines. For each item i we define a cost function as follows. $C(i,j) = 1$ if and only if $j \in D_i$, otherwise it is a large constant. Processing time of job i (corresponding to item i) is $|D_i|$ (uniform processing time on all machines). Using Theorem A.1 [17], the scheduling algorithm finds a schedule that assigns each job (item) to a machine (disk) to minimize the makespan. They show that the makespan is at most the makespan in a fractional solution plus the processing time of the largest job. Moreover, the cost of their solution is at most the cost of the optimal solution, namely the number of items. We cannot assign an item (job) to a disk (machine) if the disk is not in the destination set for the item.