ABSTRACT

Title of dissertation:	ALGORITHMS FOR DATA MIGRATION
	Yoo Ah Kim, Doctor of Philosophy, 2005
Dissertation directed by:	Professor Samir Khuller Department of Computer Science

This thesis is concerned with the problem related to data storage and management. A large storage server consists of several hundreds of disks. To balance the load across disks, the system computes data layouts that are typically adjusted according to the workload. As workloads change over time, the system recomputes the data layout, and rearranges the data items according to the new layout. We identify the problem of computing an efficient data migration plan that converts an initial layout to a target layout.

We define the data migration problem as follows: for each item, there are a set of disks that have the item (sources) and a set of disks that want to receive the item (destinations). We want to migrate the data items from the sources to destinations. The crucial constraint is that each disk can participate in only one transfer at a time. The most common objective has been to minimize the makespan, which is the time when we finish all the migrations. The problem is NP-hard, and we develop polynomial time algorithms with constant factor approximation guarantees and several other heuristic algorithms. We present the performance evaluation of the different methods through an experimental study.

We also consider the data migration problem to minimize the sum of completion times over all migration jobs or storage devices. Minimizing the sum of completion times of jobs is one of the most common objectives in scheduling literature. On the other hand, since a storage device may run inefficiently while the device is involved in migrations, another interesting objective is to minimize the sum of completion times over all storage devices. We present hardness results and constant factor approximation algorithms for these objectives.

In addition, we consider the case when we have a heterogeneous collection of machines. We assume that heterogeneity is modeled by a non-uniform speed of the sending machine. For the basic problem of multicasting and broadcasting in the model, we show that Fastest Node First scheme gives a approximation ratio of 1.5 for minimizing the makespan. We also prove that there is a polynomial time approximation scheme.

Algorithms for Data Migration

by

Yoo Ah Kim

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

DEDICATION

to

my husband and daughter.

ACKNOWLEDGMENTS

I was very fortunate to have many great people who made this dissertation possible.

First and foremost, I would like to thank my advisor, Samir Khuller. Without his guidance and support, this dissertation would not exist. I admire him as a hardworking researcher and an excellent teacher, and all I have learned from him will benefit me for my whole life.

I thank Professor Gnanalingam Anandalingam, Samrat Bhattacharjee, Neil Spring, and Aravind Srinivasan for serving on my thesis committee and for spending their invaluable time reviewing my dissertation. Professor David Mount and Liviu Iftode also gave me thoughtful comments in the early version of my dissertation.

My graduate life in Maryland was full of happy moments thanks to wonderful friends. In particular, my special gratitude goes to Rajiv Gandhi. He gave me valuable guidance as a senior graduate student in my early years here, and also gave me useful advice during my job search. A great part of this dissertation is done with help of Yung-Chun (Justin) Wan. He is a smart, hardworking researcher and I always enjoyed working with him.

I am forever indebted to my parents. Their everlasting love and support made it possible for me to be what I am now. I also thank my only sister, Eun-Ah for always being my best friend, and my parents-in-law for their encouragement.

Lastly, I would like to thank my husband, Jihwang and my daughter, Jungha. My husband have been a great emotional support, which kept me not to lose my energy and enthusiasm. My daughter's smile always gives me a great joy of life. Words cannot express how much I love them and this dissertation is dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1. Data Migration Problem

A large storage server consists of several hundreds of disks, connected using a dedicated network, called a *Storage Area Network*. Those disks may be used to store multimedia files in video-on-demand servers, or search indexes in search engine clusters, for example. Disks typically have constraints on storage as well as the number of clients that can access data from a single disk simultaneously. For example, suppose that the system consists of disks which have 36 GB of storage capacity and can support a transfer rate of 20 MB/s (load capacity). When we consider MPEG-2 movies of 100 mins each with bandwidth requirement of 4 Mbps, the disk can store 12 movies and support 40 streams.

The ability to match resources with demand through layout affects the performance of a storage system. When we deal with bandwidth intensive applications, e.g., video-on-demand, the demand for popular items can be very high and therefore, the system needs to replicate the data items on several different disks. For example, as shown in Figure 1.1, popular items such as data A and D have several copies stored on different disks. Replication is a common approach to handle high demand and also provides fault tolerance. Therefore, in the layout we compute how many copies of each item we need as well as a mapping that specifies the precise subset of items that each disk needs to store. Given demand for each data item, it is NP-hard to find a specific data layout pattern to maximize the total utilization where utilization refers to the number of clients assigned to a disk that contains the data they want. Several approximation algorithms have been developed for this problem [55, 56, 57, 28, 40]. The goal is to have the system automatically respond to changes in demand patterns and to recompute data layouts. Such systems and their applications

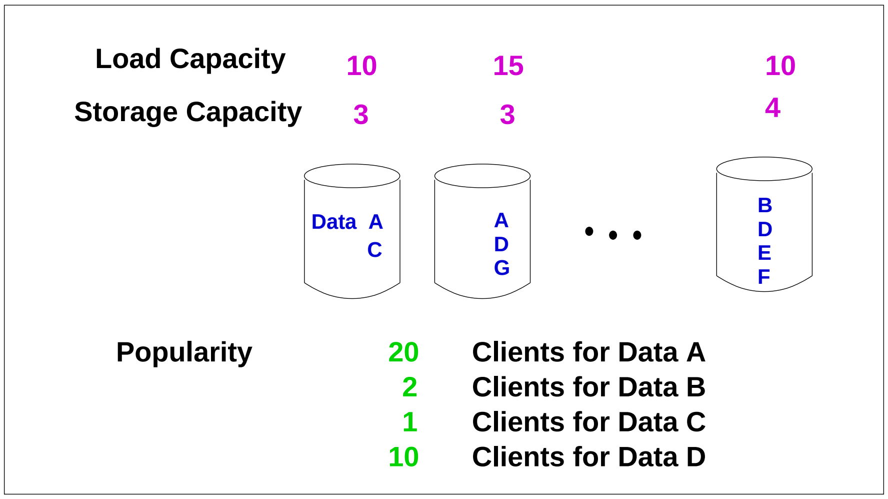


Figure 1.1: Each disk has a storage capacity which is the number of items it can store, and a load capacity which is the number of clients it can serve at any point of time. Each data item has different popularity.

are described and studied in, e.g., [21, 26, 63] and references given therein.

Over time as the demand for data changes, the system needs to create *new* data layouts. We are interested in the problem of computing a *data migration* plan for the set of disks to convert an initial layout to a target layout. An example is shown in Figure 1.2. The system may run inefficiently until migrations are finished since the data layout may not be able to meet the current demand for data. In addition, the system needs to use a significant amount of bandwidth for data migration, which may also degrade the performance of the system. Therefore, it is important to find a migration schedule that can be done as quickly as possible.

1.2. Models and Definitions

In the *data migration* problem, we have disks and data items. Given an initial and target layout, we can compute S_i (source set) and D_i (destination set) for each item i. We define S_i to be a subset of disks that have item i initially. D_i is defined to be a subset of disks that want to receive item i. In other words, all disks in D_i have item i in the target layout, but not in the initial layout. See Figure 1.3 for an example. Our goal is to convert the initial layout to the target layout

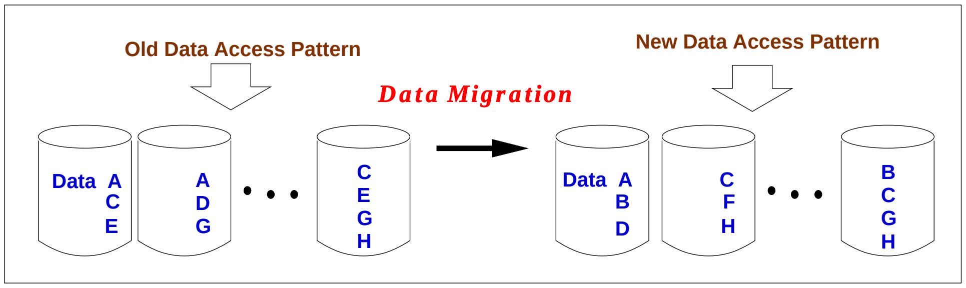


Figure 1.2: An example of the data migration problem

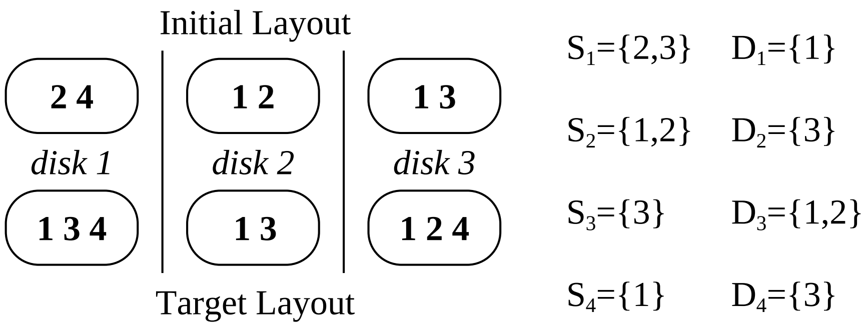


Figure 1.3: An initial and target layout, and their corresponding S_i's and D_i's

as quickly as possible.

We assume that the underlying network is fully connected. In other words, any pair of disks in the system can communicate with each other directly. In general, it is reasonable to assume that multiple transfers can occur in parallel between pairs of disks, e.g., when these disks are attached to different nodes of a distributed system (as would be typical, e.g., in a large-scale storage system).

One crucial constraint is that each disk can participate in the transfer of only one item — either as a sender or as a receiver. The model is one of the most widely used in all the work related to data migration. This assumption is reasonable as trying to transfer multiple distinct pieces of data simultaneously from the same disk is likely to result in transfer inefficiencies and trying to transfer the same piece of data to multiple other disks would place special requirements on the communication medium, e.g., network multicasting capabilities (which is not typical).

Therefore, we assume that any set of transfers can take place at the same time as long as each disk is used as a sender or a receiver (but not both) for only one item. This model best captures the connection of parallel storage devices that are connected on a network and is most appropriate for our application.

There can be other constraints in communication networks. Consider the case where the storage network connecting the disks has the limited switching capacity. We call this a *bounded bandwidth model*. In this model, we can allow only a bounded number of transfers to take place at any given time.

Each transfer may be associated with its length, which is the time required to migrate the data item from its source to destination. The time required for a transfer may vary depending on the size of files, or the speed of machines that are involved in the transfer. One special case of interest is when data items have the same size (these could be data blocks, or files) and machines are homogeneous, in which case all transfers have the same length.

Several different objectives can be considered in data migration. The most common objective in data migration is to minimize the makespan, that is, the time to finish all the migration jobs. More formally, the makespan is defined to be $\max_j C_j$ where C_j is the completion time of migration job j. We also consider the sum of completion times over all transfers ($\sum_j C_j$) or over all storage devices ($\sum_d C_d$) where C_d is the time when disk d finishes all migration jobs that it is involved in.

Finally, we do not allow indirect transfers, in which an item can be sent to a disk other than the target disk temporarily. In other words, a data item can be sent only to disks that desire it and the total number of data transfers performed is thus the minimum possible.

1.3. Contributions

We first consider data migration *with cloning*. The problem generalizes previous work by Coffman *et al.* [22] and Hall *et al.* [29] where they addressed only the data *movement* problem. They assumed that the transfer graph $G = (V, E)$ is given, where each vertex in V represents a disk in the system and an edge (u, v) represents a move operation from u to v or vice versa. So for example, one cannot create extra copies of any data item, but can just change which disks they are stored on.

However, 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 previous work on data migration [22, 29] is based on a degree property of the transfer graph. 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 and 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. Note 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 problems we study can also be considered as generalizations of the gossiping and broadcasting problems. These two problems have been the subject of extensive study [45, 33, 37, 11, 12, 38, 19, 23] as they play an important role in the design of communication protocols in various kinds of networks. The *gossip problem* is defined as follows: given n individuals, each individual has an item of gossip that they wish to communicate to everyone else. In the *broadcast problem*, one individual needs to convey an item of gossip to every other individual. Our results are the first work that relate broadcasting and gossiping with the data migration problem. The basic generalizations we are interested in are (*i*) an item of gossip may be known to several individuals, (*ii*) several items of gossip may be known to an individual, and (*iii*) each item of gossip needs to be communicated to only a subset of individuals.

For the data migration problem with cloning to minimize the makespan, we develop a 9.5 approximation[1] algorithm. This is the first approximation algorithm for this problem. While two of the lower bounds we used are quite simple, we develop a new lower bound using network flows,

[1]A c-approximation algorithm for a minimization problem is a polynomial time algorithm that yields a solution with cost at most c times the optimal.

and use this in the algorithm (without this lower bound, the best bound we can obtain is a $O(\log n)$ factor). For the case where the limited number of transfers can take place at a time (which we call the bounded bandwidth model), we can also obtain a constant factor approximation algorithm, using the constant factor approximation algorithm for the problem without bandwidth constraint.

Moreover, we explore a few simple heuristics for data migration with cloning. Some of these algorithms cannot provide constant approximation guarantees, while for some of the algorithms no approximation guarantee is known. We conduct an extensive study to compare the performance of these data migration algorithms under different changes in user access patterns.

We also identify the correspondence problem, of which solution can have a significant impact on the overall solution for data migration. Data layout algorithms (such as the ones in [55, 56, 57, 40, 28]) take as input a demand distribution for a set of data objects and output a grouping $L_{1'}, L_{2'}, \ldots L_{N'}$ as a desired data layout pattern on disks $1', 2', \ldots, N'$. (The current layout is assumed to be $L_1, L_2 \ldots L_N$.) Note that we do not need the data corresponding to the set of items $L_{1'}$ to be on (the original) disk 1. The algorithm simply requires that a new grouping be obtained where the items in set $L_{1'}$ be grouped together on a disk. Therefore, given the initial and final layout sets, we need to compute a correspondence between them, which specifies which disk needs to store the set of items $L_{i'}$. We show that a good solution to the correspondence problem can improve the performance of the data migration algorithms by a factor of 4.4 in our experiments, relative to a bad solution.

We also consider objectives other than the makespan — the sum of completion times over all storage devices or data migration jobs. For these objectives, we obtain constant factor approximations for the case when the transfer graph for data migration is given (in other words, $|S_i| = |D_i| =$ for all i). We first consider the problem of minimizing the sum of weighted completion times over all storage devices. This problem was first suggested by Coffman *et al.* [22]. In a system where storage devices can be free to do other tasks as soon as their own migrations are complete, minimizing the sum of completion times over all storage devices is an interesting objective since the performance of a device is degraded while it is involved in the migration. We show that the problem is NP-hard by reduction from the Chromatic Index problem [36]. We formulate the problem as an integer program and develop an approximation algorithm based on LP relaxation. We have a simple algorithm that gives a 3-approximation when job lengths are all the same. Recently, Gandhi *et al.* [24] proved that the analysis of our algorithm for unit job lengths is tight. In other words, there are examples in which the total completion time over all devices, using our algorithm, is 3 times the optimal. When job lengths are arbitrary integers, a more involved algorithm gives a 9-approximation.

To minimize the total completion time over all transfers, we have a 10-approximation algorithm when the lengths of edges are arbitrary integers, which improves a 12-approximation algorithm presented by Halldórsson *et al.* [32].

For the problems mentioned above, we assumed that machines in the system are homogeneous. However, in a system where machines are put together over time, they tend to have different capabilities and this leads to a *heterogenous* collection of machines rather than a *homogeneous* collection. We consider a special case of data migration in this model — the *broadcast* problem. In the broadcast problem, a source disk (or processor) has one message to be sent to all other disks. In other words, $|S_i| = 1$ and D_i includes all the disks (other than the source) in the system. *Broadcast* is one of the fundamental operations that are used in such clusters of machines. In addition broadcast is used as a primitive in many parallel algorithms.

In our work, we consider a simple model and algorithm for heterogenous network proposed by Banikazemi *et al.* [3]. In this model, heterogeneity among processors is modeled by a non-uniform speed of the sending processor. A heterogenous cluster is defined as a collection of processors $p_0, p_1, p_2, \ldots, p_n$ in which each processor is capable of communicating with any other processor. Each processor has a transmission time which is the time required to send a message to any other processor in the cluster. Thus the time required for the communication is a function of only the sender. Each processor may send messages to other processors in order, and each processor may receive only one message at a time.

A commonly used method to find a broadcast schedule is referred to as the "Fastest Node

First" (FNF) technique [3]. This works as follows: in each iteration, the algorithm chooses a sender from the set of processors that have received the message (set S) and a receiver from the set of processors that have not yet received the message (set R). The algorithm then picks the sender $s \in S$ so that s can finish the transmission as early as possible, and chooses the receiver $r \in R$ as the processor with the minimum transmission time in R. Then r is moved from R to S and the algorithm continues. The intuition here is that sending the message to fast processors first is a more effective way to quickly propagate the message. This technique is very effective and easy to implement. In practice it works extremely well (using simulations) and in fact frequently finds optimal solutions as well [3]. However, there are situations when this method fails to find an optimal solution.

We show that FNF has a performance ratio of at most 1.5 when compared to the optimal solution for minimizing broadcast time. To prove this, we make use of the fact that the FNF heuristic actually produces an optimal solution for the problem of minimizing the sum of completion times[2]. We also prove that there is a polynomial time approximation scheme (PTAS) for minimizing the broadcast time, and show that if there is a constant number of different transmission speeds, then there is a polynomial time algorithm for minimizing the broadcast time. In addition, we extend the above results to the problem of multicasting, where some other nodes not in the multicast group can be used to improve the broadcast time.

1.4. Outline of Thesis

In Chapter 2, we discuss the previous work on the data migration problem and other related work.

In Chapter 3 – 5, we present the results for data migration with cloning. For this problem, we assume that there can be several copies of an item and the objective is to minimize the makespan. In Chapter 3, we describe a 9.5-approximation algorithm for the problem. We also present some variants of the approximation algorithm and several simple heuristics. In Chapter 4, we formulate

[2]It was shown in the paper by Hall *et al.* [30]. In Chapter 7, we provide a simpler proof for the problem.

the correspondence problem and describe several algorithms for the correspondence problem. The experimental results are presented in Chapter 5.

In Chapter 6, we consider objectives other than the makespan. For this problem, we assume that we are given a transfer graph (that is, we only consider move operations). We first discuss the results to minimize the total completion time over all storage devices. We show that the problem is NP-hard, and present a 3-approximation algorithm for unit transfer lengths and a 9-approximation algorithm for arbitrary transfer lengths. We also consider the total completion time over all transfers and present a 10-approximation algorithm for arbitrary transfer lengths.

In Chapter 7, we discuss the broadcasting problem in heterogenous networks. We prove that FNF gives 1.5-approximation to minimize the broadcast time and also develop a polynomial time approximation scheme. We show that the results can be applied in multicast.

Finally, we conclude our results and suggest future work in Chapter 8.

Chapter 2

Related Work

In this chapter, we describe the previous work in data migration. We also present other work that are related to the data migration problem.

2.1. Previous Work in Data Migration

Coffman *et al.* [22] introduced the data migration problem to minimize the makespan of the migration schedule, i.e., the time by which all migrations are finished. In their work, they assumed that the transfer graph $G = (V, E)$ is given where each vertex $v \in V$ represents a storage device and each edge $e = (u, v)$ corresponds to a transfer that has to be done from u to v, or vice versa. When transfer lengths are arbitrary integers, they showed that greedy algorithms yield a 2-approximation. In the special case where all the transfer have the same length, it is exactly the problem of finding the chromatic index of multi-graphs. The chromatic index is the smallest number χ' of colors required to color the edges of a graph G so that adjacent edges are colored differently. The problem is NP-hard, and several approximation algorithms have been developed [27, 34, 53, 18]. The current best known result is the algorithm using $1.1\chi' + 0.8$ colors [53][1], which in turn yields the same approximation guarantee for the data migration with unit transfer lengths. Coffman *et al.* [22] also presented complexity results and algorithms for various restricted cases of transfer graphs (e.g., bipartite graphs, trees, paths and cycles).

Hall *et al.* [29] considered two interesting variants of the data migration problem. Given a transfer graph G with unit transfer lengths, they considered the cases (i) when the *bypass nodes*[2] are used to improve the performance of a data migration, and (ii) when there are space constraints,

[1]The additive term was improved to 0.7 by Caprara *et al.* [18]

[2]A bypass node is a node that is not the target of a move operation, but used as an intermediate holding point for a data item.

e.g., each disk has only one spare unit of storage. They showed that data migration can be done in $2\lceil \Delta/2 \rceil$ rounds with at most $n/3$ bypass nodes, or at most $3\lceil \Delta/2 \rceil$ rounds without bypass nodes, where Δ is the maximum degree of nodes in the transfer graph and n is the number of nodes. With space constraints, they developed an algorithm that takes at most $4\lceil \Delta/4 \rceil$ rounds with at most $n/3$ bypass nodes, or at most $6\lceil \Delta/4 \rceil$ rounds without bypass nodes.

On the other hand, to handle high demand for popular objects, several copies will have to be dynamically created and stored on different disks. The data migration problem with cloning is the *most general problem* of interest where data item i can have several copies in a specified source subset S_i of disks, and needs to be moved to a destination subset D_i.

2.2. Gossiping and Broadcasting

The problems of Gossiping and Broadcasting have been the subject of extensive study [45, 33, 37, 11, 12, 38, 19, 23]. These play an important role in the design of communication protocols in various kinds of networks. The *gossip problem* is defined as follows: there are n individuals and each individual has an item of gossip that they wish to communicate to everyone else. Communication is typically done in rounds, where in each round an individual may communicate with at most one other individual. Some communication models allow for the full exchange of all items of gossip known to each individual in a single round. Other models allow the sending of only one item of gossip from one to the other (half-duplex) or allow each individual to send an item to the individual they are communicating with in this round (full-duplex). In addition, there may be a communication graph whose edges indicate which pairs of individuals are allowed to communicate directly in each round. (In the classic gossip problem, also called the telephone model, communication may take place between any pair of individuals; in other words, the communication graph is the complete graph.) In the *broadcast problem*, one individual needs to convey an item of gossip to every other individual. The two parameters typically used to evaluate the algorithms for this problem are: the number of communication rounds, and the total number of telephone calls placed.

Data migration with cloning is a generalization of the above mentioned gossiping and broadcasting problems. The basic generalizations we are interested in are (*i*) an item of gossip may be known to several individuals, (*ii*) several items of gossip may be known to an individual, and (*iii*) each item of gossip needs to be communicated to only a subset of individuals.

The paper by Liben-Nowell [48] considers a generalization of gossiping and broadcasting, which is exactly the data migration problem with restrictions that each disk contains at most one source item and each item has at most one source. However, the model that he uses is different than the one that we use. In his model, in each telephone call, a pair of users can exchange all the items of gossip that they know. The objective is to simply minimize the total number of phone calls required to convey item i of gossip to set D_i of users. In our case, since each item of gossip is a data item that might take considerable time to transfer between two disks, we cannot assume that an arbitrary number of data items can be exchanged in a single round. Several other papers use the same telephone call model [2, 17, 35, 38, 60].

Other related problems that have been studied are the set-to-set gossiping problem [47, 54] where we are given two possibly intersecting sets A and B of gossipers and the goal is to minimize the number of calls required to inform all gossipers in A of all the gossip known to members in B. Liben-Nowell [48] generalizes this work by defining for each gossiper i the set of relevant gossip that they need to learn.

Several special cases of the data migration problem with cloning have been studied where each data item has only one copy initially, and developed algorithms with better performance guarantees [42]. One special case studied in the paper is the *Multi-source multicast* problem, when Δ data items, each having only one copy, are stored in Δ different disks, and data item i needs to be sent to a specified subset D_i of disks. This problem is shown to be NP-hard by a reduction from a restricted version of 3SAT. Using a simplified version of 9.5-approximation algorithm for the general problem, a polynomial-time algorithm with approximation ratio of 4 was developed and it was further improved to $3 + o(1)$. Allowing bypass nodes, the approximation ratio of 3 could be obtained. Another special case is the *Multi-source broadcast* problem, which is the same as the Multi-source multicast problem except that all disks demand all items, i.e., D_i is all the disks minus its source. A polynomial-time algorithm using at most 3 more rounds than the optimal was developed for this problem. The last special case studied is the *Single-source multicast* problem, when Δ data items, each having only one copy, are all stored on the same disk, and data item i needs to be sent to a specified subset D_i of disks. There is a polynomial-time algorithm using at most Δ more rounds than the optimal.

2.3. Minimizing Total Completion Time

Minimizing the sum of weighted *job* completion times is one of the most common measures in the scheduling literature since it can reflect the priorities of jobs. The problem of minimizing the total edge completion time leads to a special case of the *sum (multi-)coloring* problem [4, 9, 6, 46, 52, 7]. The sum coloring problem is defined as follows. Let $G = (V, E)$ be an undirected graph, possibly with vertex weights. A coloring of a graph is an assignment $\phi : V \to N$ of positive integers to the vertices so that different colors are assigned to adjacent vertices. The objective is to minimize the total sum of colors assigned to the vertices. When each vertex requires multiple colors, it is called the sum multi-coloring problem. Given a transfer graph G for the data migration problem, minimizing the total edge completion time is reduced to the sum coloring of the line graph[3] of G (the sum multi-coloring if edges have different lengths). Bar-Noy *et al.* [4] proved that the sum coloring of line graphs is NP-hard and any greedy coloring gives a 2-approximation. For the sum multi-coloring of line graphs, we give a 10-approximation, using an LP-based algorithm. This improves the previous bound of 12 by Halldórsson *et al.* [32]. The general idea of their algorithm is to have edges wait for some time before it is actually processed and waiting times are chosen based on job lengths. Very recently, Gandhi *et al.* [25] further improved the bound to 7.683.

In a system where storage devices can be free to do other tasks as soon as their own migrations are complete, minimizing the sum of completion times over *all storage devices* is interesting

[3]The line graph of G has a vertex corresponding to each edge of G, and there is an edge between two vertices in the line graph if the corresponding edges are incident on a common vertex in G.

since the performance of a device is degraded while it is involved in migrations. This problem was first suggested in the paper by Coffman *et al.* [22]. We present a simple algorithm that gives a 3-approximation when job lengths are all the same. When job lengths are arbitrary integers, a more involved algorithm gives a 9-approximation. Recently, Gandhi *et al.* [24] proved that the analysis of our algorithm for unit job lengths is tight. They also improved the bound to 5.055 for arbitrary integer lengths. They choose the wait function based on linear programming relaxation given in this thesis and obtain the schedule by having edges wait before it is actually processed (the idea of wait function was first used in the paper by Halldórsson *et al.* [32] for minimizing the job completion time).

2.4. Heterogenous Networks

Various models for heterogenous environments have been proposed in the literature. One general model is the one proposed by Bar-Noy *et al.* [5] where the communication costs between links are not uniform. In addition, the sender may engage in another communication before the current one is complete. An approximation factor with a guarantee of $O(\log k)$ is given for the operation of performing a multicast. Other popular models in the theory literature generally assume an underlying communication graph, with the property that only nodes adjacent in this graph may communicate.

Broadcasting efficiently is an essential operation and many works are devoted to this (see [54, 33, 39, 8, 14] and references therein). In addition, for emergency notification an understanding of how to perform broadcast quickly is essential.

Most of the algorithmic (theoretical) work done on such problems is of little interest to practitioners, because the approximation algorithms tend to be fairly complex and slow, thus defeating the purpose of performing broadcast quickly. On the other hand, a simple model and algorithm was proposed by Banikazemi *et al.* [3]. In this model, heterogeneity among processors is modeled by a non-uniform speed of the sending processor. A heterogenous cluster is defined as a collection of processors $p_1, p_2, \ldots, p_n$ in which each processor is capable of communicating with any other processor. Each processor has a transmission time which is the time required to send a message to any other processor in the cluster. Thus the time required for the communication is a function of only the sender. Each processor may send messages to other processors in order, and each processor may be receiving only one message at a time. It was shown that minimizing broadcast time in this model is NP-hard [30].

A commonly used method to find a broadcast tree is referred to as the "Fastest Node First" (FNF) technique [3]. FNF scheme works as follows: In each iteration, the algorithm chooses a sender from the set of processors that have received the message (set S) and a receiver from the set of processors that have not yet received the message (set R). The algorithm then picks the sender $s \in S$ so that s can finish the transmission as early as possible, and chooses the receiver $r \in R$ as the processor with the minimum transmission time in R. Then r is moved from R to S and the algorithm continues. The intuition is that sending the message to fast processors first is a more effective way to propagate the message quickly. This technique is very effective and easy to implement. In practice it works extremely well (using simulations) and in fact frequently finds optimal solutions as well [3]. However, there are situations when this method fails to find an optimal solution.

It was shown that FNF minimizes the sum of completion times [30] and also gives 2-approximation for minimum broadcast time [49, 50, 51]. Liu [50] also showed that if there are only two classes of processors, then FNF produces an optimal solution. In addition, if the transmission time of every slower processor is a multiple of the transmission time of every faster processor, then again the FNF heuristic produces an optimal solution. So for example, if the transmission time of the fastest processor is 1 and the transmission time of all other processors are powers of 2, then the algorithm produces an optimal solution.

2.5. Other Related Work

Our algorithms make use of known results on edge coloring of multi-graphs. Given a graph G with max degree Δ_G and multiplicity[4] μ, the following results are known (see Bondy-Murty [15] for example). Let χ' be the edge chromatic number of G.

Theorem 2.5.1: *(Vizing [62]) If G has no self-loops then $\chi' \leq \Delta_G + \mu$.*

Theorem 2.5.2: *(Shannon [58]) If G has no self-loops then $\chi' \leq \lfloor \frac{3}{2}\Delta_G \rfloor$.*

[4]The multiplicity of a graph is defined as the maximum number of edges that appear between a pair of vertices.

Chapter 3

Data Migration with Cloning

We develop several algorithms for data migration with cloning. We first present a polynomial time algorithm that gives approximation guraunteee of 9.5. We then describe two simple heuristices. Finally we consider the bandwith limited model where only a limited number of transfers can take place at any point of time. For this model, we also develop constant factor approximation algorithms.

3.1. Motivation

In the previous work in the data migration problem [22, 29], they assumed that we are given a transfer graph $G = (V, E)$ where each node $v \in V$ represents a storage device and each edge $e \in E$ corresponds to a transfer that has to be done from u to v or vice versa. In other words, they addressed only the data *movement* problem. (So for example, one cannot create extra copies of any data item, but can just change which disks they are stored on.)

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 [29] is based on a degree property of the transfer graph. 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).

In our work, we assume that we are given an initial and target layout. Note that since there may be several copies of a data item, there can be several possible transfer graphs and an optimal transfer graph cannot be computed directly.

The problem is NP-hard by reduction from edge coloring and we develop a polynomial-time 9.5-approximation algorithm, which will be presented in the following section. For all our algorithms, we move data only to disks that need the data. Thus we do not use bypass nodes. The total number of data transfers performed is thus the minimum possible.

We use the same communication model as in the work by Hall *et al.* [29, 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). Later we will also discuss a restricted model where the devices may communicate on a matching, but the size of the matching is constrained (we call this the bounded-bandwidth model).

3.2. 9.5-approximation algorithm

Recall that S_i denotes a subset of disks that have item i initially. D_i is defined to be a subset of disks that want to receive item i. In other words, all disks in D_i have item i in the target layout, but not in the initial layout.

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. Note that β is a lower bound on the optimal number of rounds, since the disk i that attains the maximum, needs at least β rounds to receive all the items j such that $i \in D_j$, since it can receive at most one item in each round.

Moreover, we may assume that $D_i \neq \emptyset$ and $D_i \cap S_i = \emptyset$ (we simply define the destination

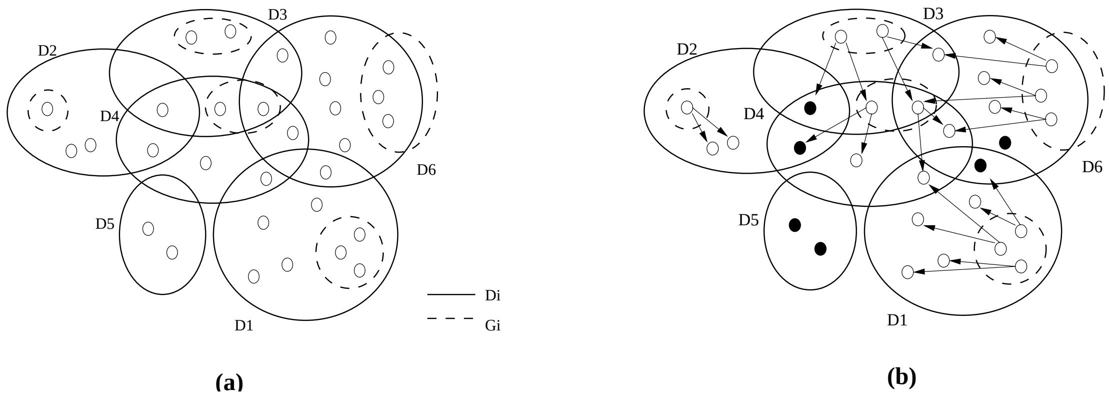


Figure 3.1: **(a)** An example of choosing G_i in Step 2(a) where $\Delta = 6$ and $\beta = 3$.

(b) transfer graph constructed in Step 2(c). Disks marked as black do not receive some data items and will be taken care of in Step 3.

set D_i as the set of disks that need item i and do not currently have it).

Since the algorithm is somewhat complex, we first give a high level description of the algorithm and then discuss the various steps in the following lemmas. Dealing with multiple data items sharing common disks causes some difficulty.

Algorithm Data Migration.

1. 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 that a disk may be a source (s_i) or destination for. *Note that α is also a lower bound on the optimal number of rounds.* In Lemma 3.2.1 we will show how we can find a source for each item.

2. Find a transfer graph for items that have $|D_i| \geq \beta$ as follows.

   (a) We first compute a disjoint collection of subsets $G_i, i = 1 \ldots \Delta$. Moreover, $G_i \subseteq D_i$ and $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$. Figure 3.1(a) shows an example of choosing G_i. (In Lemma 3.2.2, we will show how such G_i's can be obtained.)

   (b) We have each item i sent to the set G_i as shown in Lemma 3.2.5.

   (c) We 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 $D_i \setminus G_i$ from G_i is 1. Figure 3.1(b) shows an example of the transfer graph constructed in this step. We redefine D_i as the 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. Find a transfer graph for items such that $|D_i| < \beta$ as follows.

   (a) 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 for which j is a new source. See Lemma 3.2.7 for the details of this step.

   (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 $D_i \setminus \{s'_i\}$. Lemma 3.2.9 will show that the out-degree of a disk does not exceed $2\beta - 4$.

4. We now find an edge coloring of the transfer graph obtained by merging two transfer graphs in Step 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. In Lemma 3.2.10 we derive an upper bound on the number of required colors.

Lemma 3.2.1: *We can 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.*

Proof: We create a flow network with a source s and a sink t as shown in Figure 3.2. 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. □

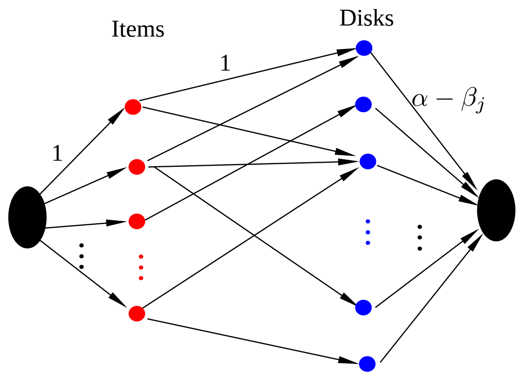


Figure 3.2: Flow network to find α

Lemma 3.2.2: *There is a way to choose disjoint sets G_i for each $i = 1 \ldots \Delta$, such that $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$ and $G_i \subseteq D_i$.*

Proof: First note that the total size of the sets G_i is at most N.

$$\sum_i |G_i| \leq \sum_i \frac{|D_i|}{\beta} = \frac{1}{\beta} \sum_i |D_i|.$$

Note that $\sum_i |D_i|$ is at most βN by definition of β. This proves the upper bound of N on the total size of all the sets G_i.

We now show how to find the sets G_i. As shown in Figure 3.3, 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 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. The min-cut in this network is obtained by simply selecting the outgoing edges from s. We can find a fractional flow of this value as follows: saturate all the outgoing edges from s. From each node i there are $|D_i|$ edges to nodes in W. Suppose $\lambda_i = \lfloor \frac{|D_i|}{\beta} \rfloor$. Send $\frac{1}{\beta}$ units of flow along $\lambda_i \beta$ outgoing edges from i. Note that since $\lambda_i \beta \leq |D_i|$ this can be done. Observe that the total incoming flow to a vertex in W is at most 1 since there are at most β incoming edges, each carrying at most $\frac{1}{\beta}$ units of flow. 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

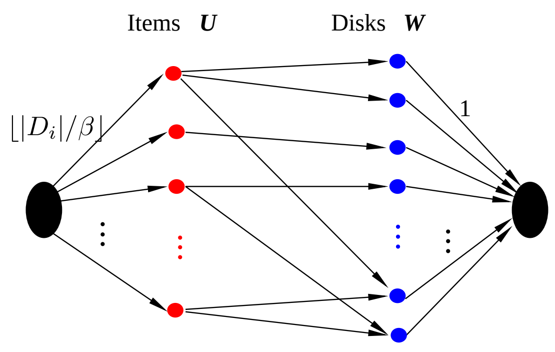


Figure 3.3: Flow network to find G_i

non-zero flow will form the set G_i. □

For 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. The only thing we have to be careful of is that the sources for many data items are shared. However, this broadcast takes at least $\max_i \log |G_i|$ rounds. Unfortunately, we cannot argue that this is a valid lower bound since even though D_i is large, if S_i is large, then there could be a solution using $O(1)$ rounds. 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 3.4. 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}.

Lemma 3.2.3: *If m' be the smallest number such that we can construct a solution of $F_{m'}$ that satisfies all demands $|G_i|$, then $M \geq m'$.*

Proof: Suppose that $M < m'$. Given an optimal schedule of Step 2(b), we can construct a solution of the flow network F_M as follows. If a disk j sends item i to a disk in G_i at round $t \leq M$, which makes f copies in G_i subsequently, we send a flow f from x_{jt} to y_i. Note that f cannot be more

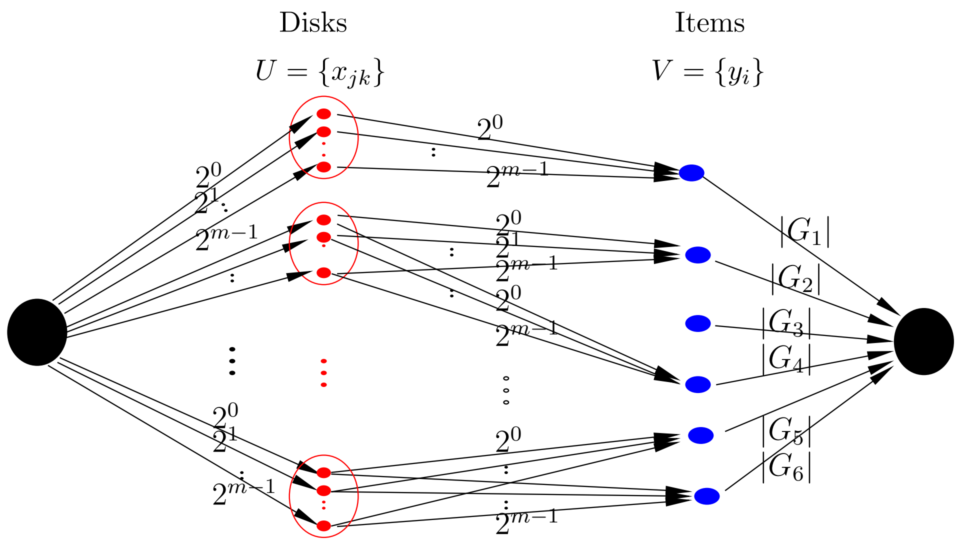


Figure 3.4: An example of constructing F_m where $\Delta = 6$

than 2^{M-t} and therefore, it does not violate the capacity constraint. Since all disks in G_i receive item i after M rounds with this schedule, the corresponding flow satisfies all demands $|G_i|$. This is a contradiction to the assumption that m' is the smallest number to satisfy all demands $|G_i|$. □

In the solution of the flow network $F_{m'}$, a node x_{jk} may send flow to several nodes. But since in our schedule, a disk can copy only one item to a disk at a round, the solution of the flow in $F_{m'}$ may not correspond to a valid schedule.

Lemma 3.2.4: *Given a solution of $F_{m'}$, we can convert it 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.*

Proof: 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 $y_{il}(l = 1 \dots \lfloor \sum_{j,k} z_{ijk} \rfloor)$ 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. Let E_{il} be the set of edges assigned to y_{il} and $c_{il}^{max}(c_{il}^{min})$ be the maximum (minimum) capacity of the edges in E_{il}. In addition, we denote the edges not assigned to $y_{il}(l = 1 \ldots \lfloor \sum_{j,k} z_{ijk} \rfloor)$ as $E_{il'}$ and the maximum capacity of edges in $E_{il'}$ as $c_{il'}^{max}$ where l' is $\lfloor \sum_{j,k} z_{ijk} \rfloor + 1$.

In the resulting bipartite graph with U and $V' = \{y_{il}\}$, z makes a *fractional* matching which matches all vertices in V', but not necessarily all vertices in U. 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. We prove that the sum of capacities of edges matched to y_i is at least $|G_i| - c^{max}$ where c^{max} is the maximum capacity of edges, using an analysis similar to that in Shmoys-Tardos [59]. The sum of capacities of edges matched to y_i is at least

$$\begin{aligned}
\sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor} c_{il}^{min} &\geq \sum_{l=2}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} c_{il}^{max} \\
&\geq \sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} c_{il}^{max} - c_{i1}^{max} \\
&\geq \sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} \sum_{e_{ijk} \in E_{il}} c_{ijk} z_{ijk} - c_{i1}^{max} \\
&\geq \sum_{l=1}^{\lfloor \sum_{j,k} z_{ijk} \rfloor + 1} \sum_{e_{ijk} \in E_{il}} f_{ijk} - c_{i1}^{max} \\
&\geq |G_i| - c^{max}.
\end{aligned}$$

Since $c^{max} \leq 2^{m'-1}$, the second property can be satisfied by setting flow through e_{ijk} as c_{ijk} if e_{ijk} is matched. □

Lemma 3.2.5: *Step 2(b) can be done in $\alpha + 2m' + 1$ rounds.*

Proof: We can do this with the following schedule. 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. Then the maximum degree of a disk is at most $m' + \alpha$ and the multiplicity is 2 since the out-degree of disk j is at most $m' + \alpha - \beta_j$ and the in-degree is at most $\min(\beta_j, 1)$. Therefore, it can be done in $m' + \alpha + 2$ rounds.

Now $|H_i|$ nodes in G_i have item i. 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. $\square$

Lemma 3.2.7 will show how Step 3(a) works. The lemma uses the following result from Shmoys-Tardos [59].

Theorem 3.2.6: *(Shmoys-Tardos [59]) 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.*

Lemma 3.2.7: *For each item i we wish to choose a source disk s'_i from D_i. Let I_j be the set of items for which disk j is chosen as a source. There is a way to choose the sources such that the following properties hold*

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

Proof: We use Theorem 3.2.6 for this step. For example, we can 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 3.2.6 [59], 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.

In our case, it is easy to see that the maximum processing time of any job is $\beta - 1$. We will argue that there is a fractional solution with makespan β. It thus follows that by defining I_j to be the set of items (jobs) assigned to disk (machine) j, the result follows. The fractional solution is obtained by assigning each job fractionally to each machine by setting the assignment variable for job i on machine j to $\frac{1}{|D_i|}$ if $j \in D_i$ then the job is fully assigned fractionally and the fractional load on each machine is at most β. This also gives us a way of finding a fractional solution efficiently. $\square$

Lemma 3.2.8: *Step 3(b) can be done in* $\lfloor \frac{3}{2}\alpha \rfloor$ *rounds.*

Proof: Since disk j can be s'_i in Step 3(a) only if $j \in D_i$, $|I_j| \leq \beta_j$. Therefore, a disk j may need to send $\alpha - \beta_j$ items, and receive β_j items. That means the maximum degree is α and this transfer can make a multi-graph. Given a multi-graph with maximum degree Δ_G, we can find an edge coloring using $\lfloor \frac{3}{2}\Delta_G \rfloor$ colors (see Theorem 2.5.2 [58]) and the lemma follows. $\square$

Lemma 3.2.9: *The maximum out-degree of a disk in the transfer graph in Step 3(c) is at most* $2\beta - 4$.

Proof: If disk j is a new source for k items (in other words, $|I_j| = k$), then the out-degree of disk j is $\sum_{i \in I_j} |D_i \setminus \{s'_i\}| = \sum_{i \in I_j} |D_i| - k$.

It is easy to see that the lemma is true for $k \geq 3$ since $\sum_{i \in I_j} |D_i| \leq 2\beta - 1$. For $k = 1$, the lemma is also true since $\sum_{i \in I_j} |D_i| \leq \beta - 1$ and $\beta \geq 2$ (otherwise, these is no set with size less than β). For $k = 2$, $\sum_{i \in I_j} |D_i| \leq 2\beta - 2$ and therefore, we have the lemma. $\square$

Figure 3.5 shows an example of migrations in Step 3.

Lemma 3.2.10: *The number of colors we need for the final transfer graph in Step 4 is* $(4\beta - 5) + (\beta + 2)$.

Proof: The out-degree of a disk j can be at most $3\beta - 5$ ($\beta - 1$ in Step 2 and $2\beta - 4$ in Step 3). The in-degree is at most β by definition. We claim that the multiplicity of the final transfer graph is

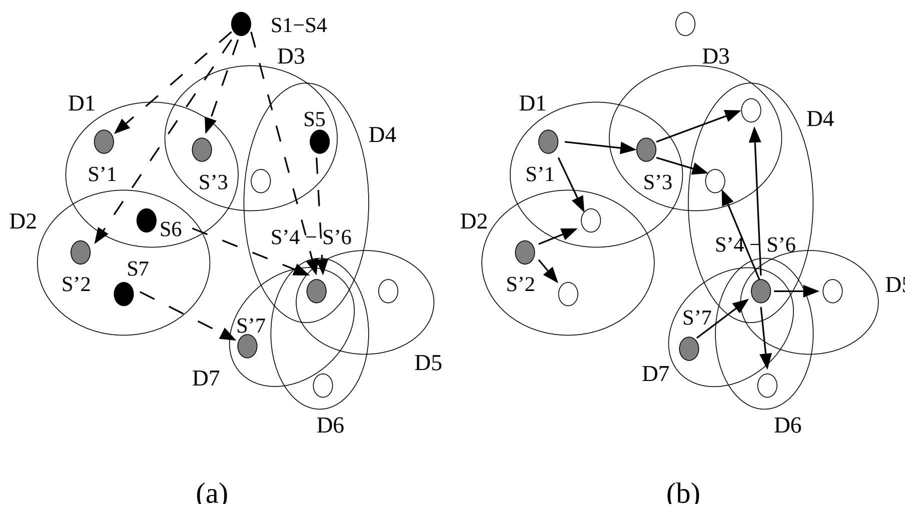


Figure 3.5: An example of Step 3 where $\alpha = 4$ and $\beta = 4$. **(a)** migration from s_i to s'_i **(b)** migration from s'_i to $D_i \setminus \{s'_i\}$.

$\beta+2$. Consider all the edges added in Step 2, we will show the multiplicity induced by these edges is 2. Since all G_i's are disjoint and each disk in G_i sends only item i to disks in $D_i \setminus G_i$, for any pair of disks j_1 and j_2, there can be at most one edge in each direction. Now consider all the edges added in Step 3, if there is an edge between disk j_1 and disk j_2, no matter which disk is the sender, both disks belong to D_i for some i. Thus, there are at most β edges between j_1 and j_2 since a disk can belongs to at most β different D_i's. Therefore, the result follows by Theorem 2.5.1. □

Theorem 3.2.11: *The total number of rounds required for the data migration is at most* $\alpha + 2m' + \lfloor \frac{3}{2}\alpha \rfloor + 5\beta - 2$.

Proof: We need $\alpha+2m'+1$ rounds for Step 2(b) by Lemma 3.2.5 and $\lfloor \frac{3}{2}\alpha \rfloor$ rounds for Step 3(b) by Lemma 3.2.8. Migration according to the coloring of the final transfer graph needs $(4\beta-5)+(\beta+2)$ by Lemma 3.2.10. Therefore, we have the theorem. □

Corollary 3.2.12: *Our algorithm is 9.5-approximation for the data migration problem.*

Proof: Since α, β, m' are lower bounds for the problem, the corollary follows. □

Theorem 3.2.13: *The Data Migration Problem (with cloning) is* NP*-hard.*

Proof: We give a simple reduction from the problem of edge coloring a simple graph with the smallest number of colors (which is known to be NP-hard [36]). Given a graph $G = (V, E)$, we create an instance of a data migration problem with V as disks. For each edge $e_i = (u, v)$ in the graph, we create a new item i, where S_i is $\{u\}$ and D_i is $\{v\}$. It is not difficult to see that the minimum number of colors required in the edge coloring instance is the same as the minimum number of rounds in the corresponding data migration instance. $\square$

3.2.1. A Bad Example

Here we give an example when our data migration algorithm does not perform very well. Consider the problem where there are Δ source disks, each disk having a separate item; in addition, there are $\Delta - 1$ destination disks, each disk requests all Δ items. Thus N is equal to $2\Delta - 1$, Δ is equal to β, and $|D_i|$, the number of disks requesting item i, is equal to $\beta - 1$ for all i. In Step 3 of our data migration algorithm, we need to find Δ new sources s'_i but we have only $\Delta - 1$ destination disks. At least one disk d has to be a new source for two items. Therefore step 3(b) takes at least 2 rounds. In disk d, each item in d has to be sent to the remaining $\Delta - 2$ destination disks. The out-degree is exactly $2\Delta - 4$. The in-degree is $\Delta - 2$. So, we have a node of degree $3\Delta - 6$ in the transfer graph, and the total number of rounds is at least $3\Delta - 4$. The optimal strategy is to have $\Delta - 1$ of Δ source disks sending items to destination disks in a round-robin fashion. This method only takes Δ rounds. Therefore, our algorithm cannot perform better than $(3 - \epsilon)$-approximation.

3.2.2. 9.5-approximation Algorithm Variants

The 9.5-approximation algorithm for data migration (denoted by *KKW (Basic)*) uses several complicated components to achieve the constant factor approximation guarantee. We consider simpler variants of these components. The variants may not give good theoretical bounds, but are simpler to implement. Some of them give better performance.

(a) in Step 2(a) (*Choose representatives*) we find the minimum integer m such that there exist disjoint sets 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 sends item i to D_i directly for small sets. We try to find a schedule which 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 a new source s'_i, S_is can be candidates as well as D_is. 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 algorithms resulting from these variants, 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 $O((n + \Delta)n\Delta \log \frac{(n+\Delta)^2}{n\Delta} \log \Delta)$.

3.3. Heuristics

In this section, we consider several heuristic-based data migration algorithms. Some of these algorithms cannot provide constant approximation guarantees, while for some of the algorithms no approximation guarantee is known. However, we found that the heuristics give better approximate solutions than 9.5-approximation algorithms for most of inputs in the experiments. We will discuss the experimental results of these algorithms in Chapter 5.

3.3.1. Edge Coloring Based Heuristic

In this heuristic, we first create a transfer graph so that the maximum degree of a node is minimized, and then find a schedule using edge coloring on the transfer graph.

1. Since a disk can get the same item from different source disks, we want to select source disks so that the maximum degree of a node is minimized. We can do this by using a flow network. We build a flow network with a source s and a sink t. In addition we have two sets of nodes corresponding to items and source disks. We add edges from s to node i with capacity $|D_i|$

for all item i, and edges from source disks j to t with capacity $c - \beta_j$. Finally, we add edges from item i to source disk j if $j \in S_i$. Suppose c' is the minimum c such that, for all i, the amount of flow from s to i is the same as its capacity. Such c' can be obtained by binary search and solving a network flow problem in each iteration. Let $f^*(i,j)$ be the flow value from item i to source disk j when c' is obtained.

2. We build a transfer graph as follows. Each disk is a node in the graph. For each source disk j having item i initially, we put $f^*(i,j)$ directed edges from disk j to $f^*(i,j)$ different disks in D_i, meaning that source disk j would send item i to $f^*(i,j)$ disks in D_i. From the flow network, we know that $\sum_{j \in S_i} f^*(i,j) = |D_i|$ for all i. Thus, all destination disks are served. Moreover, each source disk j serves at most $c' - \beta_j$ disks, and receives β_j items from other disks. The total degree of each source disk in the transfer graph is at most c'.

3. Find an edge coloring of the transfer graph (which may be a multigraph) to obtain a valid schedule [15], and the number of colors used is an upper bound on the total number of rounds.

The worst-case time complexity here is $O((n+\Delta)n\beta \log \frac{(n+\Delta)^2}{n\beta} + n^2\beta^2)$.

Note that the edgecoloring-based heuristic does not make use of newly created copies of items. Therefore, in case when for some item i, source set S_i is small and D_i is very big, the heuristic may not perform very well since it does not use cloning while migration.

3.3.2. Matching Based Heuristic

We develop another heuristic based on maximum matching.

1. Build a undirected graph as follows. Each disk is a node in the graph. There is an edge between u and v if $u \in S_i$ and $v \in D_i$ or vice versa.

2. Find a maximum matching on this graph. This takes one round to send items in the matched pairs.

3. We update the S_i's and D_i's. If disk v received item i, then v should be removed from D_i and added to S_i. Then v can be used as a source for item i in the next round.

4. If we have not satisfied all demands (there are some non-empty D_i's), go to Step 1.

We may assign weights to edges in Step 1 and then find a maximum-weight matching in Step 2. For example, let $b(i, v, w)$ be the benefit of sending item i from disk $u \in S_i$ to disk $v \in D_i$ and it may be computed as $\log \frac{|D_i|}{|S_i|}$. Then the weight $w(u, v)$ is obtained as $\max(\max_i c(i, v, w), \max_i c(i, w, v))$ since if disk u and v are matched, we would like to send the item with the greatest benefit, and either v or w can be the sender. We discuss more on how to choose weights in Chapter 5.

This heuristic needs to compute a maximum (weighted) matching in each round. Therefore, the worst-case time complexity is $O(n\sqrt{n}\beta T)$ for unweighted case and $O(n^2(\beta + \log n)T)$ for weighted case where T is the number of rounds required for data migration.

3.4. Bounded Bandwidth Model

The following algorithm gives a constant factor approximation when at most C transfers are allowed in each round. Let E_i be the transfers in i-th round in the algorithm for the general model. Then we split each E_i into $\lceil |E_i|/C \rceil$ sets of size at most C and perform each set in a round.

Theorem 3.4.1: *Given ρ-approximation algorithm for the model without bandwidth constraint, we have a $1 + \rho(1 - 1/C)$-approximation algorithm for the bounded bandwidth model where C is the maximum number of transfers allowed in a round.*

Proof: Let us denote the number of rounds required in an optimal solution for the general model and bounded bandwidth model as OPT and OPT', respectively. Also denote the number of rounds in our algorithm as t and t'.

Note that since we move data only to disks that need the data, the total number of data transfers performed in the algorithm is the minimum possible. Thus $OPT' \geq \sum_i |E_i|/C$. Also as $t \leq \rho OPT$ and $OPT \leq OPT'$, we have $t \leq \rho OPT'$.

Therefore,

$$t' = \sum_{i=1}^{t} \lceil \frac{|E_i|}{C} \rceil$$

$$\begin{aligned}
&\leq \sum_{i=1}^{t}(\frac{|E_i|-1}{C}+1) \\
&= \frac{1}{C}\sum_{i=0}^{t}|E_i| + t(1-\frac{1}{C}) \\
&\leq OPT' + \rho OPT'(1-\frac{1}{C}) \\
&= (1+\rho(1-\frac{1}{C}))OPT'
\end{aligned}$$

□

Corollary 3.4.2: *We have a $1 + 9.5(1 - 1/C)$-approximation algorithm for the bounded bandwidth model.*

When we consider only move operations, we can obtain better bounds for the bounded bandwidth model. Without space constraints, the problem can be reduced to edge-coloring multigraphs, which has a 1.1-approximation algorithm with an 0.7 additive term [53, 18].

Corollary 3.4.3: *When we allow only move operations, we have a $1 + 1.1(1 - 1/C)$-approximation algorithm with an 0.7 additive term for the bounded bandwidth model.*

With space constraints, the algorithm by Hall *et al.* [29] gives $\frac{3}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil$-approximation.

Corollary 3.4.4: *When we allow only move operations and there are space constraints, we have a $1 + \frac{3}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil(1 - \frac{1}{C})$-approximation algorithm for the bounded bandwidth model.*

Using at most $n/3$ bypass nodes, Hall *et al.* [29] obtained algorithms which give $\frac{2}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil$-approximation without space constraints and $\frac{4}{\Delta_G}\lceil\frac{\Delta_G}{4}\rceil$-approximation with space constraints. The algorithms add one transfer for every odd cycle. Thus we have $4OPT'/3 \geq \sum|E_i|/C$.

Theorem 3.4.5: *When we allow only move operations and use at most $n/3$ bypass nodes, there is a $\frac{4}{3}+\frac{2}{\Delta_G}\lceil\frac{\Delta_G}{2}\rceil(1-\frac{1}{C})$-approximation algorithm without space constraints and $\frac{4}{3}+\frac{4}{\Delta_G}\lceil\frac{\Delta_G}{4}\rceil(1-\frac{1}{C})$-approximation algorithm with space constraints.*

Proof: We use the same notations as in Theorem 3.4.1. Since $4OPT'/3 \geq \sum|E_i|/C$, we have

$$t' = \sum_{i=1}^{t}\lceil\frac{|E_i|}{C}\rceil$$

$$\begin{aligned}
&\leq \sum_{i=1}^{t}(\frac{|E_i|-1}{C}+1) \\
&= \frac{1}{C}\sum_{i=0}^{t}|E_i| + t(1-\frac{1}{C}) \\
&\leq \frac{4}{3}OPT' + \rho OPT'(1-\frac{1}{C}) \\
&= (\frac{4}{3}+\rho(1-\frac{1}{C}))OPT'
\end{aligned}$$

$\square$

Chapter 4

The Correspondence Problem

In this section, we formulate the correspondence problem, which can have a significant impact on the performance of data migration. We present several algorithms for the problem. We show that a good solution to the correspondence problem can improve the performance of the data migration algorithms by a factor of 4.4 in our experiments, relative to a bad solution.

4.1. Motivation

Data layout algorithms (such as the ones in [55, 56, 57, 40, 28]) take as input a demand distribution for a set of data objects and output a grouping $L_{1'}, L_{2'}, \dots L_{N'}$ as a desired data layout pattern on disks $1', 2', \dots, N'$ (the current layout is assumed to be $L_1, L_2 \dots L_N$). Note that we do not need the data corresponding to the set of items $L_{1'}$ to be on (the original) disk 1. The algorithm simply requires that a new grouping be obtained where the items in set $L_{1'}$ be grouped together on a disk. For instance, if $L_3 = L_{1'}$ then by simply "renaming" disk 3 as disk $1'$ we have obtained a disk with the set of items $L_{1'}$, assuming that these two disks are inter-changeable.

Consider the following example. In the first figure of Figure 4.1 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 a source disk to a destination disk. The final transfer graph shown in Figure 4.1 is a possible output of a data migration algorithm. This will take 5 rounds since all the data being copied is coming to a single disk; that is, node 1 will have a high in-degree. Here item V can be obtained from tertiary storage 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 the first figure of Figure 4.2, 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 in Figure 4.2).

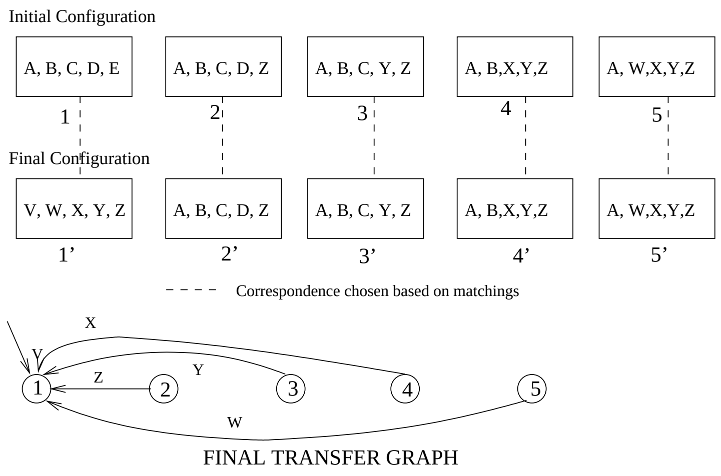


Figure 4.1: Figure to illustrate how a bad correspondence can yield a poor solution for data movement.

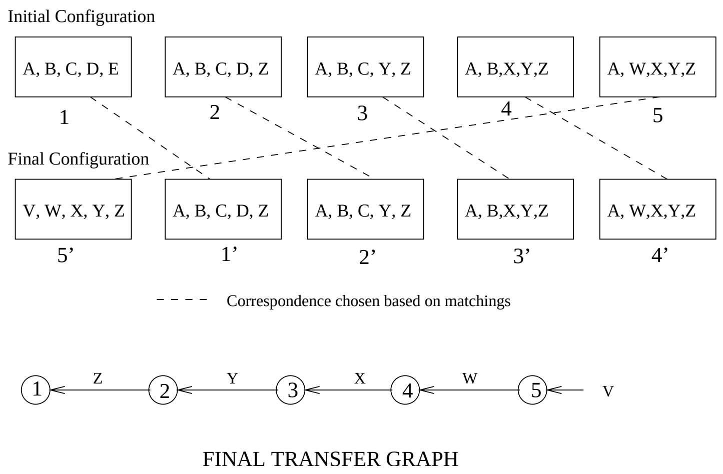


Figure 4.2: Figure to illustrate how a good correspondence can yield significantly better solutions for data movement.

4.2. Problem Definition

Given the initial and final layout sets, we need to compute a perfect matching between them. We create a bipartite graph $G = (U, V, E)$ where U and V are the sets of disks with initial and target layout, respectively. An edge is present between $L_i \in U$ and $L_{j'} \in V$ if disk i has the same capabilities as disk j'. Each edge may have its weight and we find a perfect matching with several different objectives. The detailed algorithms are described in the following section. Once we find the solution in which L_i corresponds to $L_{j'}$ if an edge between L_i and $L_{j'}$ is matched, we create an instance of data migration and invoke an algorithm to compute a migration schedule. Note that we attempt to obtain a good schedule using this two-step approach, because both the data placement problem and the data migration problem are NP-hard.

4.3. Algorithms

To match disks in the initial layout with disks in the target layout, we consider the following algorithms, after creating a bipartite graph with two copies of disks.

4.3.1. Simple min max matching

The weight of edge between disk p in the initial layout and 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). Then our goal is to find a perfect matching that minimizes the maximum weight of the edges in the matching. Effectively, this algorithm pairs disks in the initial layout with disks in the target layout, such that the number of items a disk needs to receive is minimized.

4.3.2. Simple min sum matching

We use the same weight function as in *Simple min max matching* but find a minimum weighted perfect matching. This method minimizes the total number of transfer operations.

4.3.3. Complex min sum matching

We find a minimum weighted perfect matching with different weight function that takes the ease of obtaining an item into account. Note that the larger the ratio of $|D_i|$ to $|S_i|$ is, 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 corresponding to matching these two disks is then $\sum_{i \in S} \max(\log \frac{|D_i|}{|S_i|}, 1)$.

In the experiments (see Chapter 5), we compare the algorithms described above with simple methods such as *Direct Correspondence* (disk i in the initial layout is always matched with disk i in the target layout) and *Random permutation*. We found that using a matching-based correspondence method can improve the performance of all data migration algorithms by a factor of 4.4 relative to choosing a bad correspondence. Moreover, we found that all matching-based correspondence methods considered above are comparable to one another, while the Direct correspondence method performs well only when the initial layout and the target layout are similar. Therefore we believe matching-based methods should be used, even though the Direct correspondence and the Random permutation methods run much faster than the matching-based methods. A more detailed description of the results can be found in Section 5.2.1.

Chapter 5

Experimental Study

In this section, we describe the experiments used to evaluate the performance of different correspondence algorithms and different data migration algorithms, and present the results. There are two central questions in which we are interested.

- **Which correspondence algorithm should we use?**

  We described several correspondence algorithms in Section 4.3. We want to find which correspondence algorithm should be used to improve the overall performance of data migration.

- **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.

5.1. Experimental Framework

The framework of our experiments is as follows:

1. *Create an initial layout*: Run the sliding window algorithm [28], given the number of user requests for each data object. Below we describe the distributions we use in generating user requests. These distributions are completely specified once we fix the ordering of data objects in order of decreasing demand.

2. *Create a target layout*: To obtain a target layout, we take one of the following approaches.

(a) Shuffle the ranking of items. Generate a new demand for each item according to the probabilities corresponding to the new ranking of that item. Run the sliding window algorithm again with the new request demands to obtain a target layout.

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

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

(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) as well as (b) that algorithms other than sliding window could perform well in 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 position 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, the 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. *Find a correspondence*: Run different correspondence algorithms given in Section 4.3 to match a disk in the initial layout with a disk in the target layout. Now we can find the set of source and destination disks for each item.

4. *Compute a migration schedule*: Run different data migration algorithms, and record the number of rounds needed to finish the migration.

5.1.1. User Request Distributions

We generate the number of requests for different data objects using Zipf distributions and Geometric distributions. 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 [44]:

$$\text{For all } i = 1, \ldots, M, \text{Prob(request for movie } i) = \frac{c}{i^{1-\theta}} \text{ and } 0 \leq \theta \leq 1$$

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

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 [20]. We assign θ to be 0 and 0.5 in our experiments below.

Geometric Distribution

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

$$\text{For all } i = 1, \ldots, M, \ \text{Prob(request for movie } i) = (1-p)^{i-1}p \text{ and } 0 < p < 1$$

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

5.1.2. Parameters and Layout Creation

We now describe the parameters used in the experiments. 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 of disks, namely: (A) 15 and 40, (B) 30 and 35, and (C) 60 and 150. 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.

We show the results of 5 different layout creation schemes with different combinations of methods and parameters to create the initial and target layouts. (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 the beginning of Section 5.1 (rotation of items); and (V): Initial layout obtained from the Zipf distribution ($\theta = 0$), target layout obtained from the method described in Step 2(b)ii in the beginning of Section 5.1 (enlarging D_i for items with small S_i).

5.2. Results

In the tables 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. That is, considering the average alone somewhat obscures the characteristics of the different algorithms.

5.2.1. Different correspondence methods

We first investigate how different correspondence methods affect the performance of the data migration algorithms. Figures 5.1 and 5.2 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 (Section 4.3.1) always returns the same matching as the simple min sum matching (Section 4.3.2) in all instances we tried. Moreover, using a simpler weight function (Section 4.3.2) or a more involved one (Section 4.3.3) 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 this 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 in our experiments as compared to a bad correspondence, using a Random permutation for example. 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

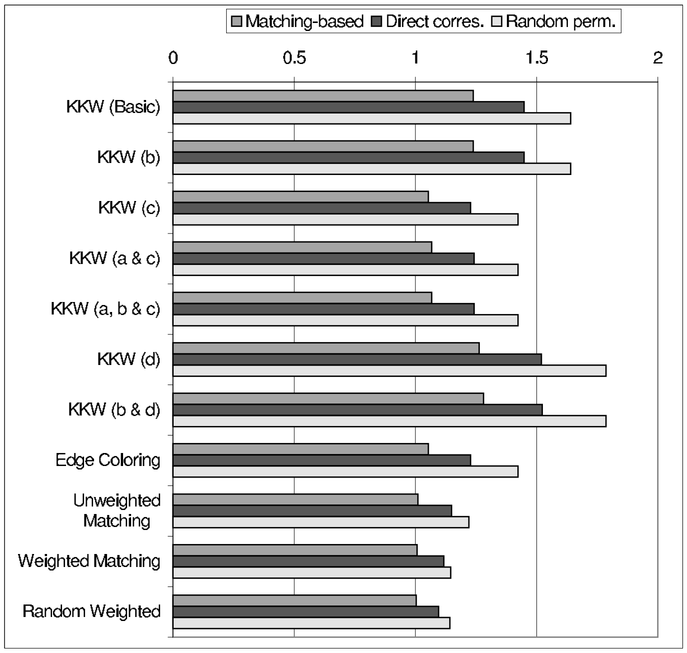


Figure 5.1: The ratio of the number of rounds taken by the algorithms to the lower bound (28.1 rounds), averaged over 20 inputs, using parameter setting (A) and layout creation scheme (I) (i.e., with 60 disks, space cap of 15, load cap of 40, promoting the last item to the top, and user requests following the Zipf distribution ($\theta = 0$)). The effect under different correspondence methods is shown.

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.

5.2.2. 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 correspondence method. We now compare the performance of different variants of our 9.5 approximation algorithm (KKW), which was described in Section 3.2.2. The results are summarized in Figure 5.3. Consider algorithm KKW (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

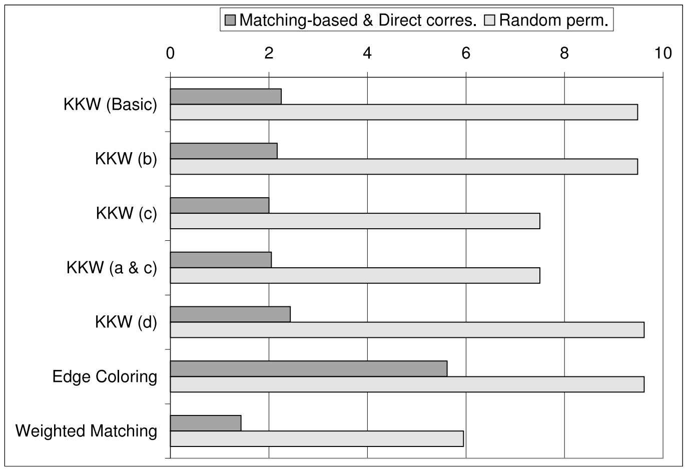


Figure 5.2: The ratio of the number of rounds taken by the algorithms to the lower bound (6.0 rounds), averaged over 20 inputs, using parameter setting (A) and layout creation scheme (II) (i.e., with 60 disks, space cap of 15, load cap of 40, promoting the last item to the top, and user requests following the Geometric distribution ($p = 0.5$)). The effect under different correspondence methods is shown.

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 $\overline{\beta}$ was a better idea (i.e., algorithm KKW (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 show that it usually performs slightly worse than the algorithm using β.

Consider algorithm KKW (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 basic KKW algorithm, but more often it is worse.

Consider algorithm KKW (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 for Parameter Setting (A), Instance 1 in Table 5.4 and for Parameter Setting (A), Instance 1 in Table 5.5. Even though this makes the algorithm an $O(\log n)$ approximation

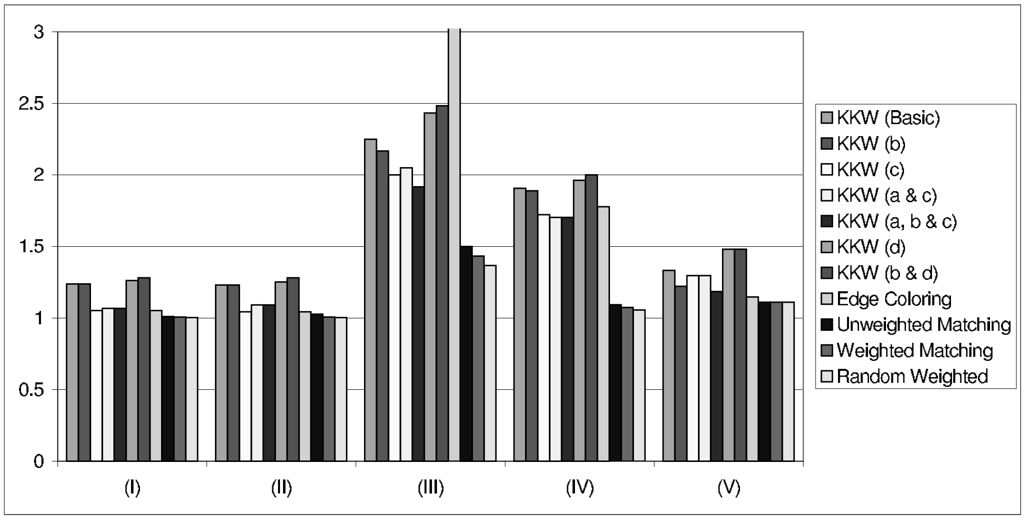


Figure 5.3: The ratio of the number of rounds taken by the algorithms to the lower bound, averaged over 20 inputs, using min max matching correspondence method and parameter setting (A), under different layout creation schemes (see Section 5.1.2). Note that the order of the bars in each cluster is the same as that in the legend.

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 basic KKW 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 results are given in Figure 5.4 and Table 5.1. The KKW 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 5.4 and Table 5.1, although this variant does not perform as well as the KKW algorithm, 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.

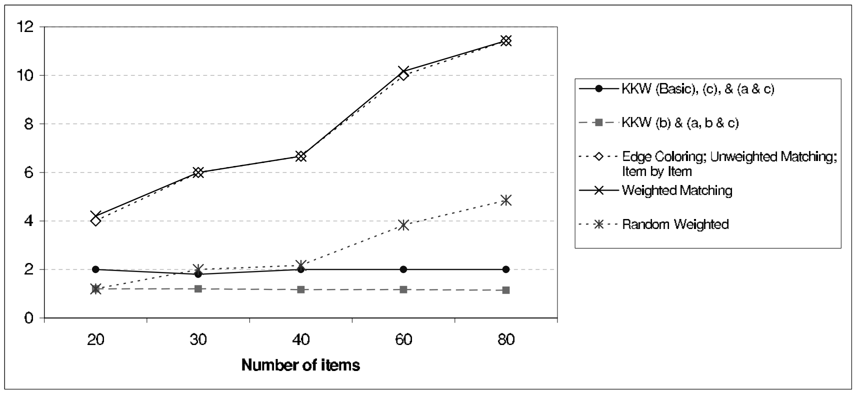


Figure 5.4: The ratio of the number of rounds taken by the algorithms to the lower bound under the worst case.

Given the performance of different data migration algorithms illustrated in Figure 5.3 and in all the tables, the two matching-based heuristics are comparable. Matching-based heuristics perform the best in general, followed by the edge-coloring heuristic, followed by the KKW algorithms. The main reason why the edge-coloring heuristic performs better than the basic KKW algorithm is that the input contains mostly move operations, i.e., the sizes of S_i and D_i are at most 2 for at least 80% of the items. The Zipf distribution does not provide enough cloning operations for the KKW algorithm to show its true potential (the sizes of G_i are mostly zero, with one or two items having sizes of 1 or 2). Under the Zipf distribution, since the sizes of most sets G_i are zero, the data migration variant which sends 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 addition to the Zipf distribution, we 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 5.2 and Table 5.4, we found that the performance of the coloring heuristic is worse than the KKW 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 of a higher ratio of the load to space capacities on the performance of different algorithms. We found the results to be qualitatively similar.

Moreover, in the Zipf distribution, we assigned different values to θ, which controls the skewness of the distribution; specifically, we considered values of 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 the KKW algorithms an advantage over coloring heuristic as more cloning is necessary.

Miscellaneous

Tables 5.5 and 5.6 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 3.2.2. Note that when the initial and target layouts 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 again to create the target layout. 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.

Tables 5.2 and 5.3 show the performance of different algorithms under two shuffling methods described in Step 2(a) in Section 5.1. We found that the results are qualitatively similar.

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 requirements for each disk needed to complete the migration. Consider disk 3 in Figure 1.3 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 maximum temporary space requirement is 4. However, since all data migration algorithms listed in this paper do not optimize for the maximum temporary space requirement, in many instances, there exists a disk that needs twice the original space requirements.

Final Conclusions

At the beginning of this chapter, we posed two major questions. For the correspondence problem question, our experiments indicate that weighted matching is the best approach among the ones we tried.

For the data migration problem question, 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 KKW algorithm, 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. In our experiments, all variants of the KKW algorithm usually take no greater than 10 more rounds than the optimal, when a good correspondence method is used.

Table 5.1: 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
KKW (Basic)	10	9	12	12	14
KKW ((b) doubling)	6	6	7	7	8
KKW ((c) S_i to D_i)	10	9	12	12	14
KKW ((a) $\bar{\beta}$, (c) S_i to D_i)	10	9	12	12	14
Edge Coloring	20	30	40	60	80
Matching	20	30	40	60	80
Weighted Matching	21	30	40	61	80
Random Weighted	6	10	13	23	34

Table 5.2: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching correspondence 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.

Parameter setting:	(A)			(B)			(C)		
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
KKW (Basic)	1.233	1.233	1.238	1.130	1.104	1.122	1.055	1.108	1.096
KKW (b)	1.233	1.233	1.238	1.130	1.104	1.122	1.055	1.108	1.096
KKW (c)	1.000	1.033	1.053	1.000	1.000	1.006	1.000	1.092	1.044
KKW (a & c)	1.033	1.067	1.068	1.022	1.021	1.021	1.000	1.092	1.044
KKW (a, b & c)	1.033	1.067	1.068	1.022	1.021	1.021	1.000	1.092	1.044
KKW (d)	1.000	1.300	1.263	1.000	1.000	1.014	1.191	1.292	1.169
KKW (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
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

Table 5.3: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching correspondence 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.

Parameter setting:	(A)			(B)			(C)		
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
KKW (Basic)	1.267	1.233	1.231	1.100	1.121	1.092	1.042	1.059	1.048
KKW (b)	1.267	1.233	1.231	1.100	1.121	1.092	1.042	1.059	1.048
KKW (c)	1.033	1.033	1.044	1.000	1.017	1.008	1.008	1.008	1.013
KKW (a & c)	1.067	1.300	1.092	1.000	1.017	1.008	1.008	1.008	1.013
KKW (a, b & c)	1.067	1.300	1.092	1.000	1.017	1.008	1.008	1.008	1.013
KKW (d)	1.367	1.167	1.252	1.167	1.207	1.149	1.203	1.263	1.201
KKW (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
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

Table 5.4: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching correspondence 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.

Parameter setting:	(A)			(B)		
Instance:	1	2	Ave	1	2	Ave
KKW (Basic)	2.000	2.167	2.250	1.611	1.611	1.568
KKW (b)	1.875	2.167	2.167	1.611	1.611	1.568
KKW (c)	1.625	2.000	2.000	1.444	1.222	1.286
KKW (a & c)	1.750	2.000	2.050	1.333	1.333	1.296
KKW (a, b & c)	1.625	2.000	1.917	1.278	1.278	1.261
KKW (d)	1.750	2.333	2.433	1.722	1.500	1.533
KKW (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
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

Table 5.5: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching correspondence method with layout creation scheme (IV) (i.e., target layout obtained from the method described in Step 2(b)i in Section 5.1 (rotation of items)), and under different parameter settings.

Parameter setting:	(A)			(B)			(C)		
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
KKW (Basic)	2.400	2.000	1.907	1.417	1.444	1.553	1.333	1.250	1.330
KKW (b)	2.200	2.000	1.889	1.417	1.444	1.553	1.333	1.250	1.330
KKW (c)	2.000	1.600	1.722	1.167	1.111	1.289	1.222	1.000	1.107
KKW (a & c)	1.800	2.000	1.704	1.250	1.333	1.408	1.222	1.083	1.170
KKW (a, b & c)	2.000	2.000	1.704	1.333	1.333	1.408	1.222	1.083	1.170
KKW (d)	2.400	2.400	1.963	1.333	1.444	1.513	1.556	1.083	1.348
KKW (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
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

Table 5.6: The ratio of the number of rounds taken by the data migration algorithms to the lower bounds, using min max matching correspondence method with layout creation scheme (V) (i.e., target layout obtained from the method described in Step 2(b)ii in Section 5.1 (enlarging D_i for items with small S_i)), and under different parameter settings.

Parameter setting:	(A)			(B)			(C)		
Instance:	1	2	Ave	1	2	Ave	1	2	Ave
KKW (Basic)	1.000	1.333	1.333	1.000	1.000	1.114	1.167	1.000	1.140
KKW (b)	1.000	1.333	1.222	1.000	1.000	1.114	1.167	1.000	1.140
KKW (c)	1.000	1.333	1.296	1.000	1.000	1.086	1.000	1.000	1.093
KKW (a & c)	1.000	1.333	1.296	1.000	1.000	1.086	1.167	1.000	1.116
KKW (a, b & c)	1.000	1.333	1.185	1.000	1.000	1.086	1.167	1.000	1.093
KKW (d)	1.000	1.667	1.481	1.000	1.500	1.286	1.500	1.750	1.488
KKW (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
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

Chapter 6

Minimizing Sum of Completion Times

In this chapter, we consider objectives other than the makespan — *total completion time*. For this objective, we assume that we are given a transfer graph. In other words, only move operations are considered. We consider the sum of weighted completion times over all migration *jobs* or storage *devices*. Minimizing the sum of job completion times is one of the most common measures in the scheduling literature since it can reflect the priorities of jobs. On the other hand, in a system where storage devices can be free to do other tasks as soon as their own migrations are complete, minimizing the sum of completion times over all storage devices is an interesting objective since the performance of a device is degraded while it is involved in migration.

6.1. Problem Definition

Given an initial and final layout of data on devices, we create the *transfer graph* $G = (V, E)$, where each vertex $v \in V$ represents a storage device and each edge $e = (u, v) \in E$ corresponds to a migration that must be done from u to v. The crucial constraint is that each vertex can participate in only one data transfer (either send or receive) in a round. Each edge e has its (integral) length p_e which represents the time required to migrate the data from its source to destination. The completion time C_e of an edge e is the time when we finish scheduling the edge. The completion time C_v of a vertex v is the time at which we finish scheduling all edges incident to it. That is, $C_v = \max_{e \in N(v)} C_e$ where $N(v)$ denotes all edges incident to v. Vertices and edges may have their weights, which we denote as w_v and w_e, respectively. Our objective is to find a schedule that minimizes the sum of weighted completion times over all vertices in V ($\sum_v w_v C_v$) or all edges in E ($\sum_e w_e C_e$).

Figure 6.1(a) shows an example of a transfer graph where all weights of vertices are one and

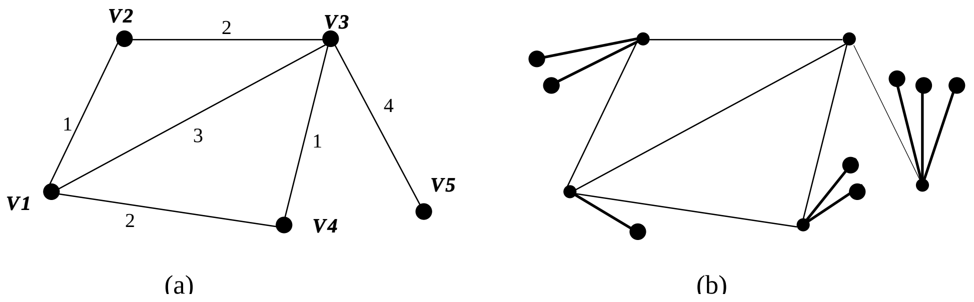


Figure 6.1: **(a)** An example of a migration plan **(b)** Constructing G'

a valid migration plan for it. In the example, the numbers beside the edges are the completion times of edges. Then the completion times of vertices are 3, 2, 4, 2, 4 in order. Therefore, the total completion time of all vertices is 15 and the total completion time of all edges is 13.

6.2. Algorithms for Total Vertex Completion Time

6.2.1. NP-hardness

We can prove that the problem of minimizing the total vertex completion time is NP-hard when vertices have arbitrary weights even though all edges have the same lengths.

Theorem 6.2.1: *The data migration problem to minimize the total weighted completion time over all vertices is NP-hard if vertices have arbitrary weights, even when edge lengths are all the same.*

Proof: We prove this by reduction from the Chromatic Index problem [36]. The Chromatic Index problem is the problem of finding the smallest number $\chi'(G)$ of colors required to color the edges of a graph G so that adjacent edges are colored differently.

Given a simple graph $G = (V, E)$, construct $G' = (V', E')$ by adding dummy vertices and edges as follows. Let $d(v)$ be the degree of a vertex v and $\Delta = \max_v d(v)$. For each vertex v, add $\Delta - d(v)$ dummy vertices and draw edges from v to them. We assign sufficiently large weights w_o (e.g., $w_o = \Delta^2|V|$) to original vertices and one to all dummy vertices. Figure 6.1(b) shows an example of how to construct G'. Note that $\chi'(G')$ is exactly the same as $\chi'(G)$ since all dummy edges incident to v can take any remaining color that does not appear at v in an edge coloring of G.

By Vizing's theorem [62], $\chi'(G)$ is Δ or $\Delta+1$. Suppose that $\chi'(G)$ is Δ (and thus $\chi'(G') = \Delta$). The number of dummy vertices is at most $(\Delta-1)|V|$ since each original vertex can have at most $\Delta - 1$ dummy vertices. Therefore, the optimal solution to our problem for instance G' is at most $w_o\Delta|V| + \Delta(\Delta-1)|V|$ ($w_o\Delta|V|$ for the original vertices and $\Delta(\Delta-1)|V|$ for dummy vertices, assuming that the completion times of all vertices is Δ). Now consider the case when $\chi'(G)$ is $\Delta+1$. The completion times of the original vertices in G' cannot be less than Δ because the degrees are Δ. Also there should be at least one vertex with the completion time no less than $\Delta+1$. Thus the optimal solution to our problem should be at least $w_o\Delta(|V|-1) + w_o(\Delta+1) = w_o\Delta|V| + w_o = w_o\Delta|V| + \Delta^2|V|$. This means that if we can find an optimal solution to minimize the total weighted vertex completion time of G', then we can decide if $\chi'(G) = \Delta$ or $\Delta+1$. □

6.2.2. Integer programming formulation

We first formulate the problem as an integer program, which draws on an IP formulation given in [31] for single-machine scheduling problems.

We should schedule two types of jobs - edge jobs and vertex jobs. Edge jobs correspond to edges e in the transfer graph G and take p_e time units to finish. Edge jobs can be scheduled in parallel as long as they are not adjacent. Vertex jobs correspond to vertices in G and take zero units of time, but cannot be scheduled until all edge jobs incident to the vertex are finished. This means that the completion time of a vertex job is the same as the last completion time of edges incident to it.

We define two variables C_v and C_e, which represent the completion times for vertex v and edge e, respectively. Let us denote the set of edges incident to v as $N(v)$. Define $p(S)$ to be $\sum_{e\in S} p_e$ and $p(S^2)$ to be $\sum_{e\in S} p_e^2$ for any set of edges S. Since we want to minimize the sum of weighted completion times of all vertices, our objective function is

$$\min \qquad \sum_{v\in V} w_v \cdot C_v \tag{6.1}$$

subject to

$$\sum_{e \in S(v)} p_e C_e \geq \frac{p(S(v))^2 + p(S(v)^2)}{2} \quad for\ all\ v,\ S(v) \subseteq N(v) \tag{6.2}$$

$$C_v \geq C_e \quad for\ all\ v, e\ where\ e \in N(v) \tag{6.3}$$

$$C_v \geq p(N(v)) \quad for\ all\ v \tag{6.4}$$

We have Constraints (6.2) because each edge job takes p_e time and any edge jobs incident to a vertex cannot be scheduled at the same time. Hall *et al.*used this type of constraint for single-machine scheduling problems [31]. To derive Constraints (6.2), consider any ordering of edges in $S(v)$. If $e \in S(v)$ is scheduled at ith order, set $C_i = C_e$ and $p_i = p_e$. Then we have

$$\begin{aligned} \sum_{j=1}^{|S(v)|} p_j \cdot C_j &\geq \sum_{j=1}^{|S(v)|} p_j (\sum_{k=1}^{j} p_k) \\ &= \sum_{j=1}^{|S(v)|} \sum_{k=1}^{j} p_j p_k \\ &= \frac{(\sum_{e \in S(v)} p_e)^2 + \sum_{e \in S(v)} p_e^2}{2} \end{aligned}$$

Constraints (6.3) imply that a vertex job can be scheduled only after all edge jobs incident to the vertex are scheduled. In fact, since vertex jobs take zero unit of time, the completion time of a vertex is the same as the last completion time of all edges incident to it. We have constraints (6.4) because the completion time of a vertex should be at least the sum of p_e of all edges $e \in N(v)$.

We relax the integrality constraints on C_v and C_e, and find an optimal solution of the LP. Although the number of constraints of this LP is exponential, it is solvable in polynomial time via the ellipsoid algorithm since there is a polynomial-time separation algorithm for the exponentially large class of Constraints (6.2) [61].

Define C_v^* and C_e^* to be the completion time of v and e in an optimal solution of the LP. For a vertex v and $e \in N(v)$, let us define $S_e(v)$ as $\{e' | e' \in N(v) \bigwedge C_{e'}^* \leq C_e^*\}$. In other words, $S_e(v)$ is the set of edges that are incident to v and scheduled no later than e in the optimal LP solution. Note that $S_e(v)$ includes the edge e itself. Constraints (6.2) give the following lemma. This is a reformulation of Lemma 2.1 in [31] using our notation.

Lemma 6.2.2: *For $e \in N(v)$, $C_e^* \geq p(S_e(v))/2$.*

Proof: We have $\sum_{e' \in S_e(v)} p_{e'} C_{e'}^* \geq p(S_e(v))^2/2$ by Constraints (6.2) and also

$$\sum_{e' \in S_e(v)} p_{e'} C_{e'}^* \leq C_e^* \sum_{e' \in S_e(v)} p_{e'} \leq C_e^* \cdot p(S_e(v)).$$

Combining two inequalities, we have the lemma. □

6.2.3. When edges have unit lengths

In this subsection, we develop a constant approximation algorithm when edge lengths are all the same. We construct our schedule based on an optimal schedule of LP, which we call *Ordered List Scheduling(OLS).*

Algorithm **OLS**

1. Sort edges in non-decreasing order of their completion times in an optimal LP solution.
2. At each time t, we scan edges not yet scheduled in the sorted order and process an edge $e = (u, v)$ at t if no edge in $S_e(u)$ and $S_e(v)$ is assigned at time t.

We can prove that the algorithm *OLS* gives a 3-approximation. Let us define $\tilde{C}_e$ and $\tilde{C}_v$ to be the completion time of e and v by *OLS*, respectively.

Lemma 6.2.3: *For any $e = (u, v)$, $\tilde{C}_e \leq p(S_e(u)) + p(S_e(v)) - p_e$.*

Proof: If e is scheduled at time t, it means that at any time $t' < t$, at least one edge in $S_e(u) \bigcup S_e(v)$ is scheduled (Otherwise, e should be scheduled at t'). This implies the lemma. □

Theorem 6.2.4: *Algorithm OLS gives a 3-approximation for the total vertex completion time when edge lengths are all the same.*

Proof: By Constraints (6.3) and Lemma 6.2.2,

$$C_v^* \geq \max_{e \in N(v)} C_e^* \geq \max_{e \in N(v)} p(S_e(u_e))/2$$

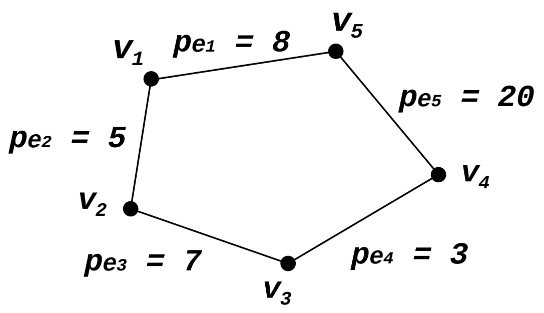


Figure 6.2: A counterexample to *OLS* for the case when edges have arbitrary job lengths.

where u_e is the endpoint of e other than v.

Also by Lemma 6.2.3 and Constraints (6.4), we have

$$\begin{aligned}
\tilde{C}_v &= \max_{e \in N(v)} \tilde{C}_e \\
&\leq \max_{e \in N(v)} (p(S_e(u_e)) + p(S_e(v))) \\
&\leq \max_{e \in N(v)} p(S_e(u_e)) \ + \ |N(v)| \\
&\leq 2C_v^* + |N(v)| \\
&\leq 3C_v^*.
\end{aligned}$$

□

Recently, Gandhi *et al.* [24] proved that the analysis of our algorithm for unit job lengths is tight. They showed that there are examples in which the total vertex completion time using our algorithm is 3 times the optimal.

6.2.4. When edges have arbitrary integer lengths

We now consider the case when the lengths of edge jobs are arbitrary integers. We present a 9-approximation algorithm for minimizing the total completion time over all vertices.

Note that with arbitrary job lengths, algorithm *OLS* does not give an upperbound in Lemma 6.2.3. This is because in *OLS* we schedule any edge job if both endpoints are available. But if the job has large job length, then it may delay other (short) jobs that should be scheduled earlier, considering their completion times in an optimal LP solution. Figure 6.2 shows an example. In this example, the job lengths of edges are $p_{e_1} = 8$, $p_{e_2} = 5$, $p_{e_3} = 7$, $p_{e_4} = 3$, $p_{e_5} = 20$ and weights are all the same. An optimal LP solution of this instance is $C_{e_1} = 13$, $C_{e_2} = 7.8$, $C_{e_3} = 10$, $C_{e_4} = 3$, $C_{e_5} = 26$ and $C_{v_1} = 13$, $C_{v_2} = 12$, $C_{v_3} = 10$, $C_{v_4} = 26$, $C_{v_5} = 28$. If we sort the edges in non-decreasing order of their completion times, we have e_4, e_2, e_3, e_1, e_5. In the algorithm *OLS*, at time $t = 3$, e_5 can be scheduled as both endpoints are available. But it prevents e_1 from being scheduled till $t = 23$ even though e_5 does not belong to $S_{e_1}(v_5)$ (this cannot happen in the unit job length case). Since $p(S_{e_1}(v_1)) + p(S_{e_1}(v_5)) = 21$, Lemma 6.2.3 does not hold.

To avoid this situation, we partition edges into subsets and schedule them in order. The intuition behind this is that we want to prevent any extremely long job from delaying short jobs that should be scheduled before it in an optimal solution. If we schedule an edge set A prior to a set B, then it means that any edge e in A has less C^*_e than edges in B or it has short job length in a sense. We call this algorithm *Partitioned List Scheduling (PLS)*.

Algorithm **PLS**

1. We first partition edges in E into $E'_1, E'_2 \dots E'_k$ using the following recursive algorithm. Let $p(G)$ be $\max_{v \in V} p(N(v))$ in G.
   (a) Sort edges in non-decreasing order of C^*_e in an optimal LP solution.
   (b) Initially $G_0 = G$. We recursively split $G_{i-1} = (V, E_{i-1})$ into $G_i = (V, E_i)$ and $G'_i = (V, E'_i)$ as follows. $E_i = \emptyset$ in the beginning. Scan edges in E_{i-1} in the sorted order and add an edge e to E_i if $p(G_i) \le p(G_{i-1})/2$ after adding e to E_i. Set $E'_i = E_{i-1} - E_i$.
   (c) We repeat this until E_k is empty.
2. We schedule edges by considering them in the order of $E'_k, E'_{k-1}, \cdots, E'_1$. When scheduling edges in E'_i, we use *List Scheduling(LS)*, in which whenever two adjacent vertices become available and there are any remaining edges between them, one of those edges is scheduled.

Lemma 6.2.5: *If an edge $e = (u, v)$ belongs to E'_i, then $C^*_e > p(G_{i-1})/4$.*

Proof: We know that $p(S_e(u)) > p(G_{i-1})/2$ or $p(S_e(v)) > p(G_{i-1})/2$ since otherwise e should be included in E_i. By Lemma 6.2.2, $C^*_e \geq p(S_e(u))/2$ and $C^*_e \geq p(S_e(v))/2$. The lemma follows. □

Lemma 6.2.6: *If an edge $e = (u, v)$ belongs to E'_i, then $\tilde{C}_e \leq 3p(G_{i-1})$ where $\tilde{C}_e$ is the completion time of edge e by Algorithm* PLS.

Proof: We prove this by induction. Let $N_H(v)$ denote a set of edges incident to v in any graph H. If $e \in E'_k$, then $\tilde{C}_e \leq p(N_{G'_k}(u)) + p(N_{G'_k}(v)) \leq 2p(G'_k) = 2p(G_{k-1})$. Now we will prove that if we can finish all edges in $E'_k, E'_{k-1}, \ldots, E'_{i+1}$ within $3p(G_i)$, then all edges in $E'_k, E'_{k-1}, \ldots, E'_i$ can be finished within $3p(G_{i-1})$. To do this, it is enough to show that scheduling edges in E'_i needs at most $3p(G_{i-1})/2$ additional time because $3p(G_i) + 3p(G_{i-1})/2 \leq 3p(G_{i-1})$.

Consider an edge $e = (u, v)$ belonging to E'_i. $LS(G'_i)$ will complete e within $p(N_{G'_i}(u) \bigcup N_{G'_i}(v)) \leq p(N_{G'_i}(u)) + p(N_{G'_i}(v)) - p_e$ time. Because e is not included in E_i, $p(N_{G_i}(u)) + p_e > p(G_{i-1})/2$ or $p(N_{G_i}(v)) + p_e > p(G_{i-1})/2$. Without loss of generality, let us assume that $p(N_{G_i}(v)) + p_e > p(G_{i-1})/2$. Then

$$\begin{aligned}
p(N_{G'_i}(u)) + p(N_{G'_i}(v)) - p_e &< p(N_{G'_i}(u)) + p(N_{G'_i}(v)) + p(N_{G_i}(v)) - p(G_{i-1})/2 \\
&= p(N_{G'_i}(u)) + p(N_{G_{i-1}}(v)) - p(G_{i-1})/2 \\
&\leq p(N_{G_{i-1}}(u)) + p(N_{G_{i-1}}(v)) - p(G_{i-1})/2 \\
&\leq 3p(G_{i-1})/2.
\end{aligned}$$

This proves the lemma. □

Theorem 6.2.7: *Algorithm* PLS *gives us a 9-approximation for the total completion time of vertices.*

Proof: The completion time of a vertex v in our schedule is determined by the last scheduled edge $e^v = (u, v)$ in $N(v)$. Let e^v be scheduled as a part of E'_i. To finish $E'_k, E'_{k-1}, \ldots, E'_{i+1}$, we need at most $3p(G_i) \leq 3p(G_{i-1})/2$. In addition, to schedule e^v in E'_i, we need at most

$p(N_{G'_i}(u)) + p(N_{G'_i}(v)) - p_e$ and it is less than $p(N_{G_{i-1}}(u)) + p(N_{G_{i-1}}(v)) - p(G_{i-1})/2$ (see the third line from the bottom of the proof for Lemma 6.2.6). Therefore,

$$\begin{aligned}\tilde{C}_v &= \tilde{C}_{e^v} \\ &\le 3p(G_{i-1})/2 + p(N_{G_{i-1}}(u)) + p(N_{G_{i-1}}(v)) - p(G_{i-1})/2 \\ &\le 2p(G_{i-1}) + p(N_{G_{i-1}}(v)) \\ &< 8C^*_{e^v} + p(N(v)) \qquad \text{(by Lemma 6.2.5)} \\ &\le 9C^*_v.\end{aligned}$$

□

6.3. Algorithm for Total Edge Completion Time

In this section, we consider the total completion time over all transfer jobs.

Recall that C_e denotes the completion time of edge e, and for any set of edges S, $p(S) = \sum_{e \in S} p_e$ and $p(S^2) = \sum_{e \in S} p_e^2$. Then IP formulation for the problem is

$$\min \quad \sum_{e \in E} w_e \cdot C_e \tag{6.5}$$

$$\text{subject to} \tag{6.6}$$

$$\sum_{e \in S(v)} p_e C_e \ge \frac{p(S(v))^2 + p(S(v)^2)}{2} \quad for\ all\ v,\ S(v) \subseteq N(v) \tag{6.7}$$

Even when the lengths of transfers are all the same, the problem is shown to be NP-hard by Bar-Noy *et al.* [4]. They also show that any greedy algorithm gives a 2-approximation when the lengths of transfers are all the same. For arbitrary transfer lengths, note that algorithm *PLS* gives a 12-approximation since for each edge e in E'_i, the completion time is upperbounded by $3p(G_{i-1})$ and $p(G_{i-1}) \le 4C^*_e$. Here we present a 10-approximation algorithm by slightly modifying *PLS*.

We modify Step 2 in *PLS* as follows.

Algorithm **Modified PLS**

1. Partition edges into $E'_k, E'_{k-1}, \cdots, E'_1$ as in *PLS*
2. We schedule edges by considering them in the order of $E'_k, E'_{k-1}, \cdots, E'_1$. When scheduling edges in each E'_i, we compute two different schedules and choose the better of two solutions.
   (a) find a schedule using *LS*.
   (b) find another schedule in which edges are performed in the reverse order of *LS*.

Lemma 6.3.1: *Modified PLS finds a schedule in which* $\sum_{e\in E'_i} w_e \tilde{C}_e \le \sum_{e\in E'_i} w_e(\frac{9}{4}p(G_{i-1}) + p_e)$ *for all* i *where* $\tilde{C}_e$ *is the completion time of* e *in our algorithm.*

Proof: Edges in E'_i can be scheduled only after all edges in $E'_j (j > i)$ are finished. We break $\tilde{C}_e$ into two parts: the delay caused by finishing $E'_j (j > i)$ and the delay after we start scheduling edges in E'_i. The first part cannot be greater than $\frac{3}{2}p(G_{i-1})$. Denote the second part as $\bar{C}_e$. Note that $\bar{C}_e$ is also upperbounded by $\frac{3}{2}p(G_{i-1})$.

If $\sum_{e\in E'_i} w_e \bar{C}_e \le \sum_{e\in E'_i} w_e \cdot \frac{3}{4}p(G_{i-1})$ in the schedule generated by *LS*, we are done since $\sum_{e\in E'_i} w_e \tilde{C}_e \le \sum_{e\in E'_i} w_e(\frac{3}{2}p(G_{i-1}) + \bar{C}_e) \le \sum_{e\in E'_i} w_e \cdot \frac{9}{4}p(G_{i-1})$. If not, consider the schedule generated in the reverse order. In the schedule, if an edge e was scheduled from $\bar{C}_e - p_e$ to $\bar{C}_e$ in the given *LS*, we schedule e starting from $\frac{3}{2}p(G_{i-1}) - \bar{C}_e$ and finish at time $\frac{3}{2}p(G_{i-1}) - \bar{C}_e + p_e$. Then the modified completion time of e is at most $3p(G_{i-1}) - \bar{C}_e + p_e$. Thus

$$\begin{aligned}
\sum_{e\in E'_i} w_e \tilde{C}_e &\le \sum_{e\in E'_i} w_e(3p(G_{i-1}) - \bar{C}_e + p_e) \\
&= \sum_{e\in E'_i} w_e(3p(G_{i-1}) + p_e) - \sum_{e\in E'_i} w_e \bar{C}_e \\
&< \sum_{e\in E'_i} w_e(3p(G_{i-1}) + p_e) - \sum_{e\in E'_i} w_e \cdot \frac{3}{4}p(G_{i-1}) \\
&= \sum_{e\in E'_i} w_e(\frac{9}{4}p(G_{i-1}) + p_e).
\end{aligned}$$

$\square$

Theorem 6.3.2: *This algorithm gives us a 10-approximation for the total completion time over all edges.*

Proof:

$$\begin{aligned}
\sum_{e\in E} w_e\tilde{C}_e &= \sum_i \sum_{e\in E'_i} w_e\tilde{C}_e \\
&\leq \sum_i \sum_{e\in E'_i} w_e(\frac{9}{4}p(G_{i-1}) + p_e) \qquad \text{(by Lemma 6.3.1)} \\
&< \sum_{e\in E} w_e \cdot 10C^*_e \qquad \text{(by Lemma 6.2.5)} \\
&\leq 10 \cdot OPT.
\end{aligned}$$

$\square$

Chapter 7

Broadcasting in Heterogenous Networks

For the problems in the previous chapters, we assumed that machines in the system are homogeneous. However, in a system where machines are put together over time, they tend to have different capabilities and this leads to a *heterogenous* collection of machines rather than a *homogeneous* collection. We consider a special case of data migration in this model — the *broadcast* problem. In the broadcast problem, a source disk (or processor) has one message to be sent to all other disks. In other words, $|S_i| = 1$ and D_i includes all the disks (other than the source) in the system. *Broadcast* is one of the fundamental operations that are used in such clusters of machines. In addition, broadcast is used as a primitive in many parallel algorithms.

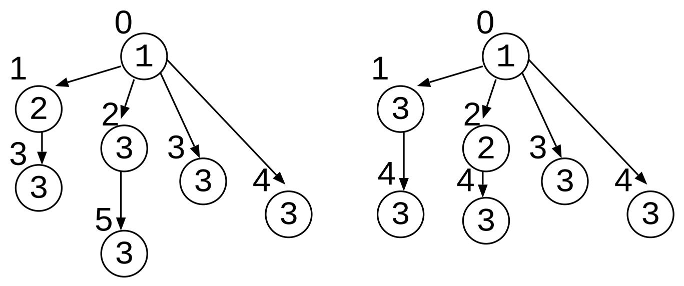
7.1. Problem Definition

We are given a set of processor $p_0, p_1, \ldots, p_n$. There is one message to be broadcast from p_0 to all other processors $p_1, p_2, \ldots, p_n$. Each processor p_i can send a message to another processor with transmission time t_i once it has received the message. Each processor can be either sending a message or receiving a message at any point of time. Without loss of generality, we assume that $t_1 \leq t_2 \leq \ldots \leq t_n$ and $t_0 = 1$.

We define the completion time of processor p_i to be the time when p_i has received the message. Our objective is to find a schedule that minimizes $C_{\max} = \max_i c_i$ where c_i is the completion time of processor p_i. In other words, we want to find a schedule that minimizes the time required to send the message to all the processors.

Our proof makes use of the following results from [3].

Theorem 7.1.1: *[3] There exists an optimal broadcast tree in which all processors send messages without delay.*



(a) FNF (b) Optimal Solution

Figure 7.1: An example that FNF does not produce an optimal solution. Transmission times of processors are inside the circles. Times at which nodes receive a message are also shown.

Theorem 7.1.2: *[3] There exists an optimal broadcast tree in which every processor has transmission time no less than its parent.*

7.2. Fastest Node First

A broadcast operation is implemented as a broadcast tree. Each node in the tree represents a processor of the cluster. The root of the tree is the source of the original message. The children of a node p_i are the processors that receive the message from p_i. The completion time of a node is the time at which it completes receiving the message from its parent. The completion time of the children of p_i is $c_i + j \cdot t_i$, where c_i is the completion time of p_i, t_i is the transmission time of p_i and j is the child number. In other words, the first child of p_i has a completion time of $c_i + t_i$ $(j = 1)$, the second child has a completion time of $c_i + 2t_i$ $(j = 2)$ etc. See Figure 7.1 for an example.

A commonly used method to find a broadcast tree is referred to as the "Fastest Node First" (FNF) technique [3]. This works as follows: In each iteration, the algorithm chooses a sender from the set of processors that have received the message (set S) and a receiver from the set of processors that have not yet received the message (set R). The algorithm then picks the sender from $s \in S$ so that s can finish the transmission as early as possible, and chooses the receiver $r \in R$ as the processor with the minimum transmission time in R. Then r is moved from R to S and the algorithm continues. The intuition is that sending the message to fast processors first is

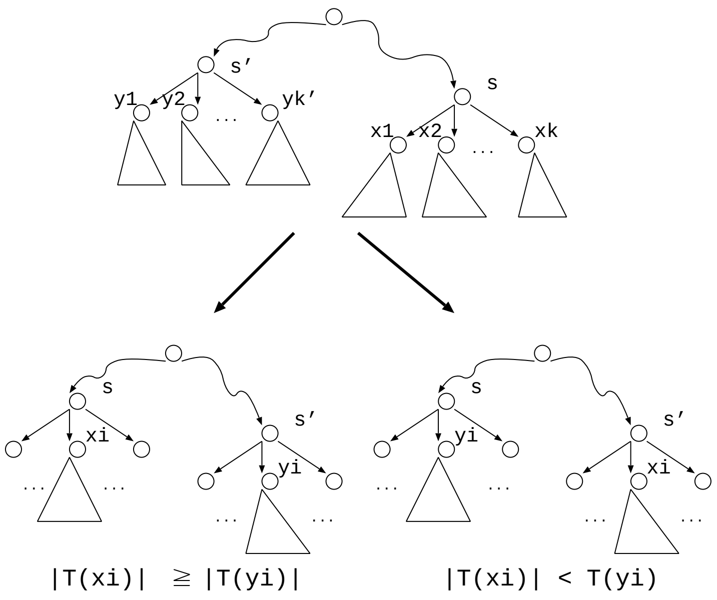


Figure 7.2: Figure shows how to modify the schedule of $T(s)$ and $T(s')$.

a more effective way to propagate the message quickly. This technique is very effective and easy to implement. In practice it works extremely well (using simulations) and in fact frequently finds optimal solutions as well [3]. However, there are situations when this method also fails to find an optimal solution. A simple example is shown in Figure 7.1.

7.2.1. Minimizing the Sum of Completion Times

In this section, we show that the FNF scheme finds an optimal schedule to minimize the sum of completion times of all the nodes, i.e., it minimizes $C_{\text{sum}} = \sum_i c_i$. This was originally shown in the paper by Hall *et al.* [30] but here we provide a simpler proof for the problem.

Suppose that we are given an (optimal) schedule that minimizes the sum of completion times and the sum of completion times is C. If there is a processor $p_i (i \geq 1)$ whose completion time is later than processor p_j $(i < j)$, i.e., $c_i > c_j$ in the schedule, then we can find a modified schedule such that processor $p_i (i \geq 1)$ has completion time no later than processor p_j and the sum of completion times is no more than C.

Let us define a permutation $\pi : \{1, 2 \ldots n\} \rightarrow \{1, 2, \ldots n\}$. Then any schedule can be represented as an ordered list of processors $(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)})$ by sorting in non-decreasing order of completion times (ties broken in accordance with their indices).

We first prove a lemma similar to Theorem 7.1.2 for sum of completion times. The proof is the same as Theorem 7.1.2 [3]. We include the proof for the completeness.

Lemma 7.2.1: *There exists a broadcast tree such that it minimizes the sum completion times and for any processor p_i other than p_0, its children have the transmission times no less than p_i.*

Proof: We prove by contradiction. Suppose that in an optimal solution for sum of completion times, there are processors p_i and p_j $(i, j \neq 0)$ with $t_i < t_j$ and p_i is a child of p_j. p_i finishes receiving the message at time c_i and let $P(c_i)$ denote the processors that have finished receiving the message by time c_i.

We consider the following schedule *only for processors in* $P(c_i)$. The schedule is similar to the original schedule but p_i and p_j are exchanged. Let p_j finish receiving the message at time c'_j in the modified schedule. Clearly $c_i > c'_j$. It is easy to see that for the processors in $P(c_i)$ the completion times are no later in the modified schedule, and therefore for the remaining processors there also exists a schedule in which their completion times are no later than the original schedule. Thus we have the lemma. □

Lemma 7.2.2: *Let $(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)})$ be an optimal schedule for the problem with sum of completion times C. Let s be the smallest index such that $\pi(s) \neq s$. Then we can find a schedule with sum of completion times no more than C and $i = \pi(i)$ for all $1 \leq i \leq s$.*

Proof: In the given optimal broadcast tree, let us call the node corresponding to processor p_i as node i. We denote the subtree rooted at node i as $T(i)$. Consider subtrees $T(s)$ and $T(s')$ where $s' = \pi(s)$. We know that s cannot be an ancestor of s' as $c_{s'} \leq c_s$. Also s' cannot be an ancestor of s as $t_s < t_{s'}$ by Lemma 7.2.1. Therefore, $T(s)$ and $T(s')$ are disjoint.

Let node s have children $x_1, x_2, \ldots, x_k$ and node s' have $y_1, y_2, \ldots, y_{k'}$ as shown in Figure 7.2. We change the schedule as follows. First we exchange s and s'. In other words, the modified completion time of p_s becomes $c_{s'}$ and the completion time of $p_{s'}$ becomes c_s. Clearly, this does not increase the sum of completion times. For all i $(1 \leq i \leq \max(k, k'))$, we compare the size of subtree $T(x_i)$ and $T(y_i)$ and attach the bigger one to s and the smaller one to s' as i-th child. (if there does not exist a child, simply consider the size of the subtree as zero)

We can prove that this modification does not increase the sum of completion times. The difference of the sum of completion times for subtree $T(x_i)$ and $T(y_i)$ depends on which parent they are attached to. In case that $|T(x_i)| \geq |T(y_i)|$, the completion times of processors in $T(x_i)$ are decreased by $c_s - c_{s'}$ since the completion time of p_{x_i} is changed from $c_s + i \cdot t_s$ to $c_{s'} + i \cdot t_s$. The completion times of processors in $T(y_i)$ are increased by $c_s - c_{s'}$ since the completion time of p_{y_i} is changed from $c_{s'} + i \cdot t_{s'}$ to $c_s + i \cdot t_{s'}$. Therefore the difference is

$$(c_s - c_{s'})(|T(y_i)| - |T(x_i)|) \leq 0$$

In case that $|T(x_i)| < |T(y_i)|$, the completion times of processors in $T(x_i)$ are increased by $i \cdot (t_{s'} - t_s)$ because the completion of x_i is modified from $c_s + i \cdot t_s$ to $c_s + i \cdot t_{s'}$. and the completion times of processors in $T(y_i)$ are decreased by $i \cdot (t_{s'} - t_s)$ because the completion of x_i is modified from $c_{s'} + i \cdot t_{s'}$ to $c_{s'} + i \cdot t_s$. Therefore the difference is

$$i \cdot (t_{s'} - t_s)(|T(x_i)| - |T(y_i)|) \leq 0$$

Therefore we have the lemma. □

By repeating the procedure in Lemma 7.2.2, we can find a schedule such that processors receive the message in non-decreasing order of their transmission times and the sum of completion times is no more than the optimal. We thus conclude:

Theorem 7.2.3: *Algorithm FNF minimizes the sum of completion times.*

In fact, we can prove even stronger result by applying the same procedure only to a subset of processors p_i $(1 \leq i \leq k)$ for any $k \leq n$.

Corollary 7.2.4: *Algorithm FNF minimizes the sum of completion times over all processors p_i $(1 \leq i \leq k)$ for any $k \leq n$.*

Proof: We do the same procedure as in Lemma 7.2.2 except that we only count processors p_i

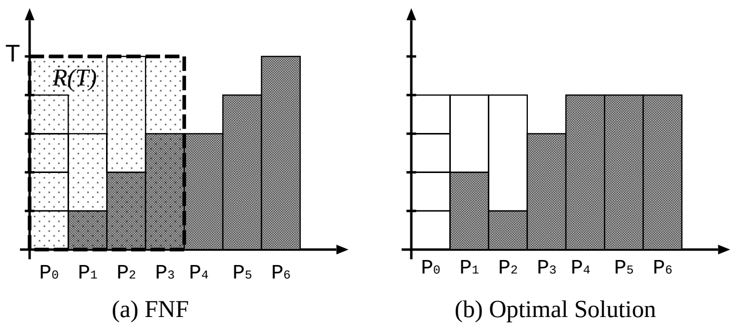


Figure 7.3: An example of Bar charts corresponding to the schedules created by the instance specified in Figure 7.1.

$(1 \leq i \leq k)$ when we compute the sizes of subtrees. $\square$

We will use this corollary in the next section to prove that the FNF scheme gives a 1.5-approximation for minimizing broadcast time.

7.2.2. 1.5-approximation for Minimizing Broadcast Time

In this section, we prove that FNF scheme gives 1.5-approximation.

Let us consider the bar chart as shown in Figure 7.2.2 (a), where processors are listed in non-decreasing order of transmission time in the horizontal line and each processor has a block whose height corresponds to its completion time in a schedule (These two charts correspond to the schedule created on the instance specified in Figure 7.1). We call these bars *grey blocks.* Each processor p_i can start sending messages as soon as it receives the message, and send to one processor in each t_i time unit. A *white block* corresponds to each message sent by p_i. Therefore, the height of a white block is the same as the transmission times of the corresponding processors.

Definition 1: *We define the number of* fractional blocks *as the number of (fractional) messages a processor can send by the given time. In other words, given a time T, the number of fractional blocks of processor p_i is $(T - c_i)/t_i$.*

For example, the number of fractional blocks of p_2 by time T is 2 in the FNF schedule (see Figure 7.2.2(a)).

Definition 2: *We define $R(T)$ as the rectangular region bounded by time 0 and T and including processors from p_0 to $p_{\lfloor n/2 \rfloor}$ in the bar chart.*

An example is shown in Figure 7.2.2(a).

We only include processors $p_0, p_1, \ldots p_{\lfloor n/2 \rfloor}$ in $R(T)$ because of the following lemma.

Lemma 7.2.5: *There is an optimal schedule in which only the $\lfloor n/2 \rfloor$ fastest processors and the source processor send messages, that is, processor $p_i(\lfloor n/2 \rfloor + 1 \le i \le n)$ need not send any messages.*

Proof: We prove this by showing that there is an optimal broadcast tree in which every internal node (except the source) has at least one child that is a leaf. Suppose an internal node $s(\neq p_0)$ does not have a leaf child. Then we move the processor which receives the message last in subtree $T(s)$ to a child of s. It is easy to see that this does not increase the makespan of the schedule by Theorem 7.1.2. By repeatedly applying this modification to the given broadcast tree, we can find an optimal broadcast tree satisfying the property. □

Lemma 7.2.6: *Algorithm FNF maximizes the number of fractional blocks in $R(T)$ for any T.*

To prove this lemma, we first prove the following proposition.

Proposition 7.2.7: $\sum_{i=1}^{m} a_i/b_i \ \ge \ \sum_{i=1}^{m} a_i'/b_i$ *if* $\sum_{i=1}^{l} a_i \ \ge \ \sum_{i=1}^{l} a'_i$ *for all* $1 \le l \le m$ *and* $0 < b_i \le b_{i+1}$ *for all* $1 \le i \le m-1$.

Proof: We will show that for all $1 \le l \le m$

$$\sum_{i=1}^{m} \frac{a_i}{b_i} \ge \sum_{i=1}^{l} \frac{a_i'}{b_i} + \sum_{i=l+1}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{l} \frac{a_i - a_i'}{b_l}.$$

Then if we set l as m, we have the proposition as $\sum_{i=1}^{m} a_i \ge \sum_{i=1}^{m} a'_i$.

We prove this by induction. For $l = 1$, it is clearly true since

$$\sum_{i=1}^{m} \frac{a_i}{b_i} = \frac{a_1'}{b_1} + \sum_{i=2}^{m} \frac{a_i}{b_i} + \frac{a_1 - a_1'}{b_1}.$$

Suppose that it is true when $l = k$. Then

$$\sum_{i=1}^{m} \frac{a_i}{b_i} \quad \ge \quad \sum_{i=1}^{k} \frac{a_i'}{b_i} + \sum_{i=k+1}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{k} \frac{a_i - a_i'}{b_k}$$

$$\begin{aligned}
&\geq \sum_{i=1}^{k} \frac{a'_i}{b_i} + \sum_{i=k+1}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{k} \frac{a_i - a'_i}{b_{k+1}} \\
&= \sum_{i=1}^{k+1} \frac{a'_i}{b_i} + \sum_{i=k+2}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{k+1} \frac{a_i - a'_i}{b_{k+1}}.
\end{aligned}$$

□

Proof of Lemma 7.2.6. Let us denote the completion time of p_i in FNF and any other given schedule as c_i and c'_i, respectively. We need to show that $\sum_{i=0}^{\lfloor n/2 \rfloor}(T - c_i)/t_i \geq \sum_{i=0}^{\lfloor n/2 \rfloor}(T - c'_i)/t_i$. In fact, since $c_0 = c'_0 = 0$, it is enough to show that $\sum_{i=1}^{\lfloor n/2 \rfloor}(T - c_i)/t_i \geq \sum_{i=1}^{\lfloor n/2 \rfloor}(T - c'_i)/t_i$. Since we have Proposition 7.2.7 and $t_i \leq t_{i+1}$, it is enough to show that $\sum_{i=1}^{l}(T - c_i) \geq \sum_{i=1}^{l}(T - c'_i)$ for all $1 \leq l \leq \lfloor n/2 \rfloor$. This is true because $\sum_{i=1}^{l} c_i \leq \sum_{i=1}^{l} c'_i$ for all $1 \leq l \leq \lfloor n/2 \rfloor$ by Corollary 7.2.4. □

Let the makespan of an optimal schedule and FNF be T_{OPT} and T_{FNF}, respectively. Then we have the following lemma.

Lemma 7.2.8: *The number of fractional blocks by FNF in $R(T_{FNF})$ is at most $3n/2$.*

Proof: If a processor receives the message from processor p_i, then it can be mapped to a white block of p_i. To finish the schedule, we should send the message to n processors and therefore, there are n white blocks in $R(T_{FNF})$. In addition, each processor $p_i (0 \leq i \leq \lfloor n/2 \rfloor)$ may have a fraction of block which is not finished by time T_{FNF}. But at least one processor should have no incomplete block since the makespan of FNF is T_{FNF}. Thus the number of fractional blocks is at most $n + \lfloor n/2 \rfloor \leq 3n/2$. □

Theorem 7.2.9: *There are at most n fractional blocks in $R(\frac{2}{3}T_{FNF})$ in any schedule.*

Proof: It is enough to show that FNF can have at most n fractional blocks in $R(\frac{2}{3}T_{FNF})$ since FNF maximizes the number of blocks by Lemma 7.2.6. By time $\frac{2}{3}T_{FNF}$, each processor can have only $\frac{2}{3}$ of the number of blocks it has in $R(T_{FNF})$. Let f_i be the number of fractional blocks of processor p_i in $R(T_{FNF})$ and f'_i be the number of fractional blocks in $R(\frac{2}{3}T_{FNF})$. Since $f_i = \frac{T_{FNF} - c_i}{t_i}$ and $f'_i = \frac{\frac{2}{3}T_{FNF} - c_i}{t_i}$, we have $f'_i \leq \frac{2}{3}(\frac{T_{FNF} - \frac{3}{2}c_i}{t_i}) \leq \frac{2}{3}f_i$. Therefore, we have at most $\frac{2}{3} \cdot 3n/2 = n$ fractional blocks in $R(\frac{2}{3}T_{FNF})$. □

Corollary 7.2.10: *Algorithm FNF gives a 1.5-approximation.*

Proof: Since we need to send the message to n processors in the optimal schedule, we should have n blocks in $R(T_{OPT})$. It implies that $T_{OPT} \geq \frac{2}{3}T_{FNF}$. □

When transmission times are in a small range, the FNF heuristic has a better bound.

Theorem 7.2.11: *Suppose $C' = n/(2\sum_{i=1}^{\lfloor n/2 \rfloor} 1/t_i)$ then the FNF heuristic finds a solution of cost at most $T_{OPT} + C'$.*

Proof: In the bar chart of an optimal solution, the number of fractional blocks we can have between height T_{FNF} and T_{OPT} is $t/t_0 + t/t_1 + \ldots + t/t_{\lfloor n/2 \rfloor}$ where $t = T_{FNF} - T_{OPT}$. Since we have at least n fractional blocks in $R(T_{OPT})$ and at most $3n/2$ fractional blocks in $R(T_{FNF})$, $\sum_{i=1}^{\lfloor n/2 \rfloor} t/t_i \leq n/2$. Therefore $T_{FNF} - T_{OPT} \leq n/(2\sum_{i=1}^{\lfloor n/2 \rfloor} 1/t_i)$. □

Theorem 7.2.12: *If the transmission times of the fastest $\lfloor n/2 \rfloor$ processors are in the range $[1 \ldots C]$, then the FNF heuristic finds a solution of cost at most $T_{OPT} + C$.*

Proof: In the bar chart of an optimal schedule, let f_i be the number of fractional blocks of processor p_i by time T_{OPT} and g_i be the number of *complete* blocks of processor p_i by T_{OPT}. In the bar chart of FNF schedule, let f'_i be the number of fractional blocks of processor p_i by time T_{OPT}.

Define g'_i to be the number of complete blocks by time $T_{OPT} + C$ and we need to prove that $\sum_{i=0}^{i=\lfloor n/2 \rfloor} g'_i \geq n$. For $1 \leq i \leq \lfloor n/2 \rfloor$, since $t_i \leq C$, we have $g'_i \geq \lceil f'_i \rceil$ Therefore, the number of *complete* blocks in $R(T_{OPT} + C)$ is

$$\begin{aligned}\sum_{i=0}^{i=\lfloor n/2 \rfloor} g'_i &\geq g'_0 + \sum_{i=1}^{i=\lfloor n/2 \rfloor} \lceil f'_i \rceil \geq g_0 + \sum_{i=1}^{i=\lfloor n/2 \rfloor} f'_i \\ &\geq g_0 + \sum_{i=1}^{i=\lfloor n/2 \rfloor} f_i \geq g_0 + \sum_{i=1}^{i=\lfloor n/2 \rfloor} g_i.\end{aligned}$$

The third inequality comes from the fact that FNF maximizes the number of fractional blocks in $R(T)$ for any T and $f_0 = f'_0$.

We also have $\sum_{i=0}^{i=\lfloor n/2 \rfloor} g_i \geq n$ by definition of T_{OPT}. Therefore, $T_{FNF} \leq T_{OPT} + C$.

□

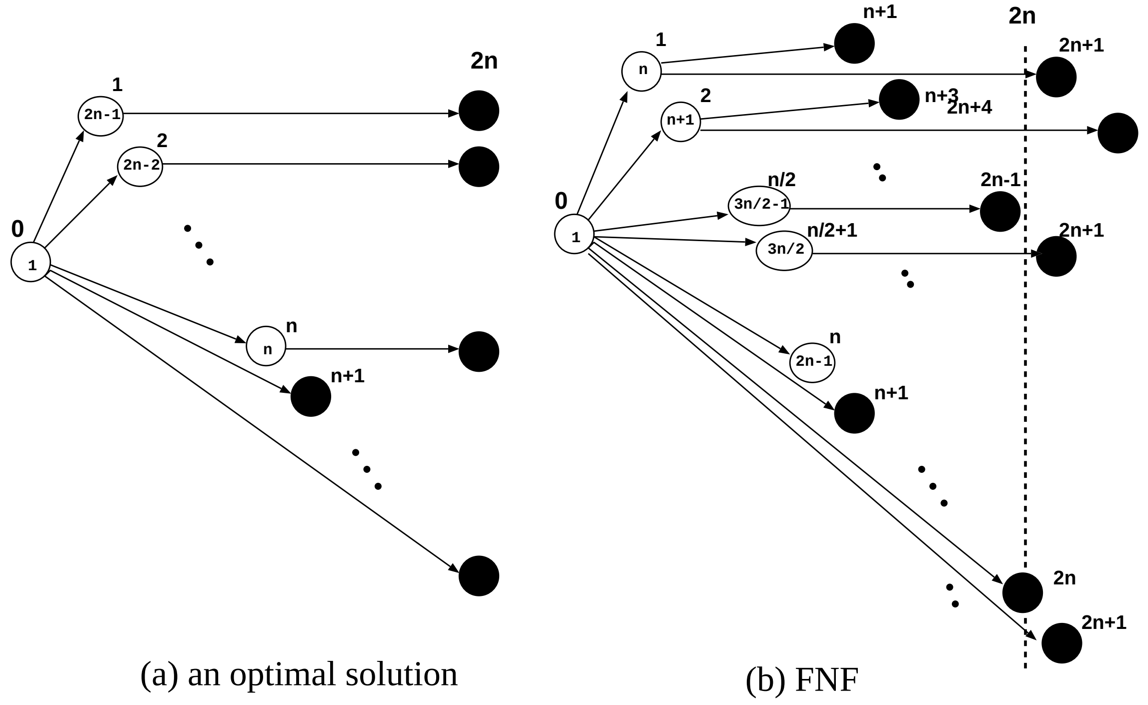


Figure 7.4: A bad example of FNF. The number inside a node is the transmission time of the processor and the number next to a node is the time it received the message. Black nodes are very slow processors. At time $2n$ (the dotted line), the optimal solution finishes broadcasting but in the FNF schedule, $\frac{n}{2}$ processors have not received the messages as yet.

7.2.3. Bad Example

Bhat *et al.* [14] gave an example and proved that the broadcast time by FNF can be $\frac{17}{16}$ times the optimal in the example. In this section, we show that in fact, FNF gives the broadcast time of $\frac{25}{22}$ times the optimal on the same example.

Consider the example shown in Figure 7.4. We have the source with transmission time 1 and $2n$ processors with a very large transmission time. Also there are n processors with transmission times $n, n+1, \ldots, 2n-1$. In the optimal schedule, the source should send messages to processors with transmission time $2n-1, 2n-2, \ldots, n$, respectively. In other words, at time i the node with transmission time $2n-i$ receives the message from the source. Immediately after receiving the message, each of these processors send a message to one of the slow processors. The schedule completes at time $2n$.

In the FNF schedule, the source sends messages to processors with transmission time $n, n+1, \ldots, 2n-1$, respectively and again immediately after receiving the message, each of these processors send a message to one of the slow processors. At time $2n$, $\frac{n}{2}$ of the slow processors have not yet received the message. After time $2n$, processor with transmission time $n+i-1$ will send another message to a slow processor at time $2n+3i-2$. At the same time, processor with transmission time $\frac{3n}{2}+i-1$ send a message at time $2n+2i-1$. The source sends a message every time unit. Therefore, if t is the time needed to send messages to the remaining $\frac{n}{2}$ processors, then we have

$$\lfloor \frac{t+2}{3} \rfloor + \lfloor \frac{t+1}{2} \rfloor + t \geq \frac{n}{2}.$$

Therefore, we need at least $\frac{3n}{11} - \frac{7}{11}$ additional time. That means that the broadcast time of FNF can be $\frac{25}{22}$ times the optimal for large values of n.

7.3. Polynomial Time Algorithm for Constant Number of Different Transmission Speeds

If there is a constant number of different transmission speeds, then we can find an optimal schedule in polynomial time. Suppose we have k different transmission speeds $(t_1, \ldots, t_k)$ and there are n_i processors with transmission time t_i. At the beginning, a processor with transmission time t_s has the message. Then the problem instance can be represented as $< t_s, (t_1, n_1), (t_2, n_2), \ldots (t_k, n_k) >$.

Using dynamic programming, we compute an optimal schedule as follows. If we select a processor with transmission time t_i as a recipient, then after time t_s two processors (with transmission time t_s and t_i) have the message. Therefore, the two new subproblems are $< t_s, (t_1, n'_1), \ldots (t_k, n'_k) >$ and $< t_i, (t_1, n''_1), \ldots (t_k, n''_k) >$ where $n'_j + n''_j = n_j$ for $j \neq i$ and $n'_i + n''_i = n_i - 1$. So we choose a recipient among k different transmission speeds and there are at most $(n_1+1)(n_2+1)\ldots(n_k+1)(\leq n^k)$ ways to split processors into two sets. We take the solution that minimizes the maximum of two subproblems among all the possible choices. Since there are at most $k \cdot n^k$ possible subproblems, this can be done in polynomial time if k is constant.

7.4. Polynomial Time Approximation Scheme

We now describe a polynomial time approximation scheme for the problem of performing broadcast in the minimum possible time. Unfortunately, the algorithm has a very high running time when compared to the *fastest node first* heuristic.

We will assume that we know the broadcast time T of the optimal solution. Since $t_0 = 1$, we know that the minimum broadcast time T is between 1 and n, and we can try all possible values of the form $(1+\epsilon)^j$ for some fixed $\epsilon > 0$ and $j = 1 \ldots \lceil \frac{\log n}{\log(1+\epsilon)} \rceil$. In this guessing process we lose a factor of $(1+\epsilon)$.

Let $\epsilon' > 0$ be a fixed constant. We define a set of fast processors F as all processors whose transmission time is at most $\epsilon' T$. Formally, $F = \{p_j | t_j \leq \epsilon' T, 1 \leq j \leq n\}$. Let S be the set of remaining (slow) processors. We partition S into collections of processors of similar transmissions speeds. For $i = 1 \ldots k$, define $S_i = \{p_j | \epsilon' T(1+\epsilon')^{i-1} < t_j \leq \epsilon' T(1+\epsilon')^i, 1 \leq j \leq n\}$ where k is $\lceil \frac{\log(1/\epsilon')}{\log(1+\epsilon')} \rceil$. Since $t_1 \leq t_2 \leq \ldots \leq t_n$, $F = \{p_1, \ldots, p_{|F|}\}$ and $S = \{p_{|F|+1}, \ldots p_n\}$.

We first send messages to processors in F using FNF. We prove that there is a schedule with broadcast time at most $(1+O(\epsilon))T$ such that all processors in F receive the message first. We then find a schedule for slow processors based on a dynamic programming approach.

Schedule for F We use the FNF heuristic to send the message to processors in set F. Assume that the schedule for F has a broadcast time of T_{FNF}. In this schedule every processor $p_j \in F$ becomes idle at some time between $T_{FNF} - t_j$ and T_{FNF}.

We will prove that there is a schedule with broadcast time at most $(1+O(\epsilon))T$ such that all processors in F receive the message first, and then send it to the slow processors. In an optimal schedule, let P_1 be the first $|F|$ processors (except the source) which finish receiving the message and T_1 be the time when all processors in P_1 finish receiving the message. The following lemma relates T_{FNF} with T_1.

Lemma 7.4.1: $T_1 \geq T_{FNF} - \epsilon' T$.

Proof: We prove this by contradiction. Consider a FNF schedule with set P_1. If an optimal schedule is able to have all processors in P_1 finish receiving the message before $T_{FNF} - \epsilon' T$, then FNF can finish sending the messages to P_1 before time T_{FNF} (by Corollary 7.2.12). Since set F includes the fastest $|F|$ processors, this means that FNF can finish broadcasting for F before time T_{FNF}; it is a contradiction since T_{FNF} is the earliest time that processors in F receive the message in FNF. □

At time T_1 there can be some set of processors P_2 which have received the message *partially.* Note that $|P_2| \leq |P_1| + 1$ since every processor in P_2 should receive the message from a processor in P_1 or from the source.

Lemma 7.4.2: *There is a schedule in which all processors in F receive the message no later than any processor in S and the makespan of the schedule is at most $(1 + 4\epsilon')T$.*

Proof: The main idea behind the proof is to show that an optimal schedule can be modified to have a certain form. Notice that by time T_1 the optimal schedule can finish broadcasting for P_1 and partially send the message to P_2, and in additional time $T - T_1$ the optimal schedule can finish broadcasting for the remaining processors.

In FNF schedule, all processors in F have the message at time T_{FNF}. Since $|P_1 \bigcup P_2| \leq 2|F| + 1$ and any processor in F has speed at most $\epsilon' T$, in additional $3\epsilon' T$ time we can finish broadcasting the message to $P_1 \bigcup P_2$. Once we broadcast the message to $P_1 \bigcup P_2$, we send the message to the remaining processors as in the optimal solution.

The broadcast time of this schedule is at most $T_{FNF} + 3\epsilon' T + T - T_1 \leq T_{FNF} + 3\epsilon' T + T - (T_{FNF} - \epsilon' T) = (1 + 4\epsilon')T$. □

Create all possible trees of S: For the processors in S, we will produce a set $\mathcal{S}$ of labeled trees $\mathcal{T}$. A tree $\mathcal{T}$ is any possible tree with broadcast time at most T consisting of a subset of processors in S. Then we label a node in the tree as i if the corresponding processor belongs to S_i ($i = 1 \ldots k$). We prove that the number of different trees is constant.

Lemma 7.4.3: *The size of $\mathcal{S}$ is constant for fixed $\epsilon' > 0$.*

Proof: First consider the size of a tree $\mathcal{T}$ (that is, the number of processors in the tree). Let us denote it as $|\mathcal{T}|$. Since the transmission time of processors in S is greater than $\epsilon' T$, we need at least $\epsilon' T$ time units to double the number of processors that received the message. It means that given a processor as a root of the tree, within time T we can have at most $2^{1/\epsilon'}$ processors receive the message. Therfore, $|\mathcal{T}| \leq 2^{1/\epsilon'}$. Now each node in the tree can have different label $i = 1 \ldots k$. To obtain an upperbound of the number of different trees, given a tree $\mathcal{T}$ we transform it to a complete binomial tree of size $2^{1/\epsilon'}$ by adding nodes labeled as 0. Then the number of different trees is at most $(k+1)^{2^{1/\epsilon'}}$. $\square$

Attach $\mathcal{T}$ to F: Let the completion time of every processor $p_j \in F$ be c_j. Each processor p_j in F sends a message to a processor in S every t_j time unit. Therefore, a fast processor p_j can send messages to at most $X_j = \lfloor \frac{T-c_j}{t_j} \rfloor$ other processors. Let $X = \sum_{p_j \in F} X_j$. Let us consider the time x_i of each sending point in X. We sort those x_i in nondecreasing order and attach a tree from $\mathcal{S}$ to each point (See Figure 7.5). Note that we can attach at most $|X|$ trees of slow processors. Clearly $|X| \leq n$.

We check if an attachment is feasible, using dynamic programming. Recall that we partition slow processors into a collection of processors $S_1, S_2, \ldots S_k (k = \frac{\log(1/\epsilon')}{\log(1+\epsilon')})$. Let s_i denote the number of processors in set S_i. We define a state $s[j, n_1, n_2, \ldots n_k]$ $(0 \leq j \leq |X|$, $0 \leq n_i \leq s_i)$ to be true if there is a set of j trees in $\mathcal{S}$ that we can attach to first j sending points and the corresponding schedule satifies the following two conditions: i) the schedule completes by time T and ii) exactly n_i processors in S_i appear in j trees in total. Our goal is to find out whether $s[j, s_1, s_2, \ldots s_k]$ is true for some j, which means that there is a feasible schedule with makespan at most T. The number of states is at most $O(n^{k+1})$ since we need at most n trees ($|X| \leq n$) and $s_i \leq n$.

Now we prove that each state can be computed in constant time. Given $s[j-1, \ldots]$, we compute $s[j, n_1, n_2, \ldots n_k]$ as follows. We try to attach all possible trees in $\mathcal{S}$ to x_j. Then $s[j, n_1, n_2, \ldots n_k]$ is true if there exists a tree $\mathcal{T}'$ such that the makespan of $\mathcal{T}'$ is at most $T - x_j$ and $s[j-1, n_1 - m_1, n_2 - m_1, \ldots n_k - m_k]$ is true where $\mathcal{T}'$ has m_i slow processors belonging to

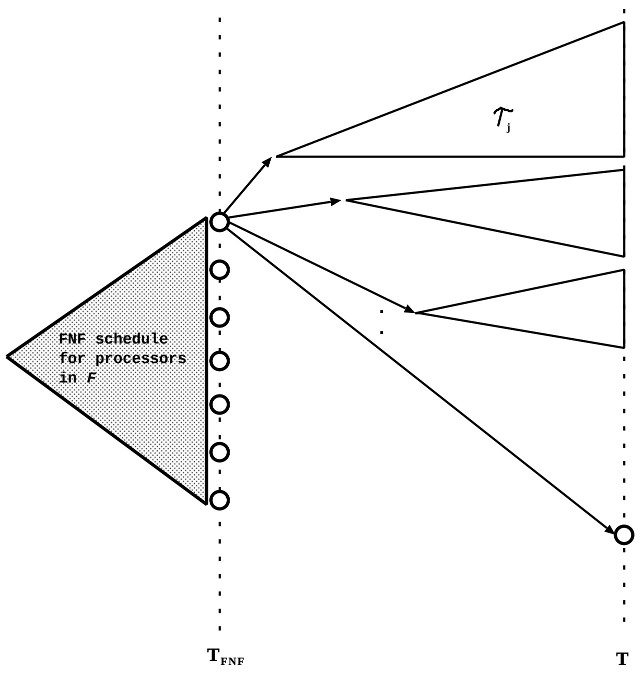


Figure 7.5: Attach trees for slow processors to fast processors

set S_i. It can be checked in constant time since the size of $\mathcal{S}$ is constant (Lemma 7.4.3).

Theorem 7.4.4: *Given a value T, if a broadcast tree with broadcast time T exists, then the above algorithm will find a broadcast tree with broadcast time at most $(1+\epsilon')(1+4\epsilon')T$.*

Proof: Consider the best schedule among all schedules in which processors in F receive the message first. By Lemma 7.4.2, the broadcast time of this schedule is at most $(1+4\epsilon')T$. We round up the transmission time of p_j in S_i to $\epsilon' T(1+\epsilon')^i$ where i is the smallest integer such that $t_j \leq \epsilon'(1+\epsilon')^i T$. By this rounding, we increase the broadcast time by factor of at most $1+\epsilon'$. Therefore, the broadcast time of our schedule is at most $(1+\epsilon')(1+4\epsilon')T$. □

Theorem 7.4.5: *The algorithm takes as input the transmission times of the n processors, and constants $\epsilon, \epsilon' > 0$. The algorithm finds a broadcast tree with broadcast time at most $(1+\epsilon)(1+\epsilon')(1+4\epsilon')T$ in polynomial time.*

Proof: We try the above algorithm for all possible value of the form $T = (1+\epsilon)^j$ for $j = 1 \ldots \lceil \frac{\log n}{\log(1+\epsilon)} \rceil$. This will increase the broadcast time by factor of at most $1+\epsilon$. Therefore the broadcast time of our schedule is at most $(1+\epsilon)(1+\epsilon')(1+4\epsilon')T$.

For each given value $(1+\epsilon)^j$, we find FNF schedule for processors in F (it takes at most $O(n \log n)$) and attach trees of slow processors to processors in F, using dynamic programming. As we discussed earlier, the number of states is $O(n^{k+1})$ and each state can be checked if it is feasible in $O((k+1)^{2^{1/\epsilon'}})$ time, which is constant. Thus the running time of our algorithm is $O(\lceil \frac{\log n}{\log(1+\epsilon)} \rceil (n \log n + (k+1)^{2^{1/\epsilon'}+1} \cdot n^{k+1})$ where k is $\lceil \frac{\log(1/\epsilon')}{\log(1+\epsilon')} \rceil$. □

7.5. Multicast

A multicast operation involves only a subset of processors. By utilizing fast processors which are not in the multicast group, we can reduce the multicasting time significantly. For example, suppose that we have m processors with transmission time t_1 and m more processors with transmission time t_2 where $t_1 < t_2$. Let we want to multicast a message to all processors with transmission time t_2. If we only use processors in the multicast group, it will take $t_2 \cdot \log m$ time. But if we utilize processors with transmission time t_1, we can finish the multicast in $t_1 \cdot (\log m + 1)$. Therefore, when $t_1 \ll t_2$, the speed-up is significant.

Theorem 7.5.1: *Suppose that we have a ρ-approximation algorithm for broadcasting. Then we can find a ρ-approximation algorithm for multicasting.*

Proof: Note that if an optimal solution utilizes k processors not in the multicast group, then those processors are the k fastest ones. Therefore, if we know how many processors participate in multicasting, we can use our ρ-approximation algorithm for broadcasting. By trying all possible k and taking the best one, we have ρ-approximation for multicasting. □

Theorem 7.5.2: *We have a polynomial time approximation scheme for multicasting.*

Proof: The proof is similar to Theorem 7.5.1. We find approximation schemes with k fastest processors not in the multicast group for all possible k, and take the best one.

□

Chapter 8

Conclusion

We have considered the data migration problem, which is the problem of finding an efficient schedule to migrate data in a network. For data migration with cloning, we develop a 9.5-approximation algorithm to minimize the makespan. We also present some variants of the approximation algorithm and several simple heuristics. We formulate the correspondence problem which can affect the performance of data migration, and describe several algorithms for the problem. We conduct an extensive study to compare the performance of these correspondence and data migration algorithms under different changes in user access patterns.

We consider objectives other than the makespan. For these objectives, we assume that we are given a transfer graph (that is, we only consider move operations). For minimizing the total completion times over all storage devices, we show that the problem is NP-hard, and present a 3-approximation algorithm for unit transfer lengths and a 9-approximation algorithm for arbitrary transfer lengths. We also consider the total completion times over all transfers and present a 10-approximation algorithm for arbitrary transfer lengths.

We also studied the broadcasting problem in heterogenous networks. We prove that the *Fastest Node First*(FNF) gives a 1.5-approximation to minimize the broadcast time, and also develop a polynomial time approximation scheme. We show that the results can be applied to multicast, where some other nodes not in the multicast group can be used to improve the broadcast time.

Several questions still remain. Experimental results suggest that it may be worthwhile to consider developing an algorithm that takes an initial layout and the new demand 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 are part of our future efforts.

For minimizing the sum of completion times over all devices, Gandhi *et al.* [24] recently proved that our analysis of *OLS* is tight when jobs require the same amount of time. However, it is possible that other algorithms can be used to improve the approximation ratio of 3. When job lengths are arbitrary integers, the current best results are a 5.055-approximation [24] for vertex completion times and a 7.683-approximation [25] for job completion times. We do not know any tight example of these bounds nor any results on hardness of approximation. We believe that the current approximation ratio for arbitrary integer cases can be improved.

One interesting generalization of heterogeneity would be for the situation when clusters of disks are connected in a wide area network. The time required to transfer one unit of data between a pair of disks in different clusters may be an order of magnitude higher than the time required to transfer data between a pair of disks in the same cluster. We obtained some results [43] for the model in which the number of rounds required to transfer one unit of data between a pair of disks in different clusters is a certain number of rounds, and one round is required to transfer one unit of data between a pair of disks in the same cluster. It would be interesting to study the problem when the time required to transfer a data item between a pair of disks in the same cluster depends on the sending processor (as in the model used in Chapter 7), and/or the communication graph between clusters is a general graph.

In practice, the metric of a *good* schedule may not simply be the completion time. For example, the solution we have obtained performs the migration schedule ignoring the interference with user traffic. The assumption may not be appropriate in modern computer environments, where it is expected to guarantee a certain level of quality of service. It will be interesting to investigate further constraints and metrics on data migration and develop efficient strategies that can provide good performance.

BIBLIOGRAPHY

[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*, pp. 145 - 158, 2001.

[2] B. Baker and R. Shostak. Gossips and Telephones. *Discrete Mathematics*, 2:191–193, 1972.

[3] M. Banikazemi, V. Moorthy and D. K. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations. *International Conference on Parallel Processing* , pp. 460–467, 1998.

[4] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai and T. Tamir. On Chromatic Sums and Distributed Resource Allocation. *Information and Computation*, 140:183-202, 1998.

[5] A. Bar-Noy, S. Guha, J. Naor and B. Schieber. Multicasting in Heterogeneous Networks. *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 448-453, 1998.

[6] A. Bar-Noy, M. M. Halldórsson and G. Korsarz. Tight Bound for the Sum of a Greedy Coloring. *Information Processing Letters*, 71:135-140, 1999.

[7] A. Bar-Noy, M. M. Halldórsson, G. Korsarz, H. Shachnai and R. Salman. Sum Multi-coloring of Graphs. *Journal of Algorithms*, 37(2):422-450, 2000.

[8] A. Bar-Noy and S. Kipnis. Designing broadcast algorithms in the Postal Model for Message-passing Systems. *Mathematical Systems Theory*, 27(5), pp. 431–452,1994.

[9] A. Bar-Noy and G. Kortsarz. The Minimum Color-sum of Bipartite Graphs. *Journal of Algorithms*, 28:339-365, 1998.

[10] 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.

[11] J. Bermond, L. Gargano and S. Perennes. Optimal Sequential Gossiping by Short Messages. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, Vol 86, 1998.

[12] J. Bermond, L. Gargano, A. A. Rescigno and U. Vaccaro. Fast gossiping by short messages. *SIAM Journal on Computing*, Vol 27(4):917–941, 1998.

[13] S. Berson, S. Ghandeharizadeh, R. R. Muntz and X. Ju. Staggered Striping in Multimedia Information Systems. *SIGMOD*, 1994.

[14] P. Bhat, C. Raghavendra and V. Prasanna. Efficient Collective Communication in Distributed Heterogeneous Systems. *Proceedings of the International Conference on Distributed Computing Systems*, pp. 15–24, 1999.

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

[16] J. Bruck, D. Dolev, C. Ho, M. Rosu and R. Strong, Efficient Message Passing Interface(MPI) for Parallel Computing on Clusters of Workstations. *J. Parallel Distributed Computing*, 40:19–34, 1997.

[17] R. T. Bumby. A Problem with Telephones. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):13–18, March 1981.

[18] A. Caprara and R. Rizzi. Improving a Family of Approximation Algorithms to Edge Color Multi-graphs. *Information Processing Letters*, 68:11-15, 1998.

[19] E. J. Cockayne and A. G. Thomason. Optimal Multi-message Broadcasting in Complete Graphs. *Utilitas Mathematica*, 18:181–199, 1980.

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

[21] 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.

[22] E. G. Coffman, M. R. Garey, D. S. Johnson and A. S. Lapaugh. Scheduling File Transfers. *SIAM Journal on Computing*, 14(3):744-780, 1985.

[23] A. M. Farley. Broadcast Time in Communication Networks. *SIAM Journal on Applied Mathematics*, 39(2):385–390, 1980.

[24] R. Gandhi, M. Halldorsson, G. Kortsarz and H. Shachnai, Improved results for data migration and open-shop scheduling. *The 31st International Colloquium on Automata, Languages and Programming* (ICALP), LNCS 3142, pp. 658–669, 2004.

[25] R. Gandhi, M. Halldorsson, G. Kortsarz and H. Shachnai. Improved Bounds for Sum Multicoloring and Weighted Completion Time of Dependent Jobs. *Proc. of the Second Workshop on Approximation and Online Algorithms (WAOA)*, pp. 68–82, 2004.

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

[27] M. K. Goldberg. Edge-coloring of Multi-graphs: Recoloring Technique. *J. Graph Theory*, 8:121-137, 1984.

[28] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella and A. Zhu. Approximation Algorithms for Data Placement on Parallel Disks. *Proc. of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA), pp. 223–232, 2000.

[29] J. Hall, J. Hartline, A. Karlin, J. Saia and J. Wilkes. On Algorithms for Efficient Data Migration. *Proc. of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA), 620–629, 2001.

[30] Nicholas G. Hall, Wei-Ping Liu and Jeffrey B. Sidney. Scheduling in Broadcast Networks. *Networks*, 32(4), pp. 233 - 253, 1998.

[31] L.A. Hall, A. S. Schulz, D.B. Shmoys and J. Wein. Scheduling to Minimize Average Completion Time: Off-line and On-line Algorithms. *Mathematics of Operations Research*, 22:513-544, 1997.

[32] M. M. Halldórsson, G. Kortsarz and H. Shachani. Minimizing Average Completion of Dedicated Tasks and Partially Ordered Sets. *4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pp. 114–126, 2001.

[33] S. M. Hedetniemi, S. T. Hedetniemi and A. Liestman. A Survey of Gossiping and Broadcasting in Communication Networks. *Networks*, 18:129–134, 1988.

[34] D.S. Hochbaum, T. Nishizeki and D.B. Shmoys. A Better than "Best Possible" Algorithm to Edge Color Multi-graphs. *J. of Algorithms*, 7:79-104, 1986.

[35] A. Hajnal, E. C. Milner and E. Szemeredi. A Cure for the Telephone Disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972.

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

[37] J. Hromkovic, R. Klasing, B. Monien and R. Peine. Dissemination of Information in Interconnection Networks (Broadcasting and Gossiping). *Combinatorial Network Theory*, pp. 125–212, D.-Z. Du and D.F. Hsu (Eds.), Kluwer Academic Publishers, Netherlands, 1996.

[38] C. A. J. Hurkens. Spreading Gossip Efficiently. *Nieuw Archief voor Wiskunde*, 5(1):208–210, 2000.

[39] R. Karp, A. Sahay, E. Santos and K.E.Schauser. Optimal Broadcast and Summation in the Logp Model. *Proceedings of 5th Annual Symposium on Parallel Algorithms and Architectures*, pp. 142–153, 1993.

[40] 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), LNCS 2914, pp. 265–276, 2003.

[41] S. Khuller, Y. Kim and Y-C. Wan. Algorithms for Data Migration with Cloning. *SIAM J. on Computing*, 33(2):448-461, 2004.

[42] S. Khuller, Y. Kim and Y-C. Wan, On Generalized Broadcasting and Gossiping. *11th Annual European Symposium on Algorithms* (ESA), 2003.

[43] S. Khuller, Y. Kim and Y-C. Wan, Broadcasting on Networks of Workstations. To appear in *17th ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA), 2005.

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

[45] W. Knodel. New gossips and telephones. *Discrete Mathematics*, 13:95, 1975.

[46] E. Kubicka, G. Kubicki and D. Kountanis. Approximation Algorithms for the Chromatic Sum. *Proceedings of the First Great Lakes Computer Science Conference*, LNCS 1203:25-21, Springer-Verlag, 1989.

[47] H. M. Lee and G. J. Chang. Set to Set Broadcasting in Communication Networks. *Discrete Applied Mathematics*, 40:411–421, 1992.

[48] D. Liben-Nowell. Gossip is Synteny: Incomplete Gossip and an Exact Algorithm for Syntenic Distance. *Proc. of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA), 177–185, 2001.

[49] R. Libeskind-Hadas, J. R. K. Hartline, P. Boothe, G. Rae and J. Swisher. On Multicast Algorithms for Heterogeneous Networks of Workstations. *J. of Parallel and Distributed Computing*, 61, pp.1665-1679, 2001.

[50] P. Liu. Broadcasting Scheduling Optimization for Heterogeneous Cluster Systems. *Journal of Algorithms* 42, pp.135-152, 2002.

[51] P. Liu and T-H. Sheng. Broadcasting Scheduling Optimization for Heterogeneous Cluster Systems. *ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pp 129–136, 2000.

[52] S. Nicoloso, M. Sarrafzadeh and X. Song. On the Sum Coloring Problem on Interval Graphs. *Algorithmica*, 23:109-126, 1999.

[53] T. Nishizeki and K. Kashiwagi. On the 1.1 edge-coloring of multigraphs. *SIAM J. on Discrete Math.*, 3:391–410, 1990.

[54] D. Richards and A. L. Liestman. Generalizations of Broadcasting and Gossiping. *Networks*, 18:125–138, 1988.

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

[56] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, Vol. 4(6):313–338, 2001.

[57] H. Shachnai and T. Tamir. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. *Proc. of Workshop on Approximation Algorithms*(APPROX), pp. 165–177, 2003.

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

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

[60] R. Tijdeman. On a Telephone Problem. *Nieuw Archief voor Wiskunde*, 19(3):188–192, 1971.

[61] M. Queyranne. Structure of a Simple Scheduling Polyhedron. *Math. Programming*, 58:263-285, 1993.

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

[63] 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.