Minimal Cost Reconfiguration of Data Placement in a Storage Area Network*†

Hadas Shachnai‡ Gal Tamir§ Tami Tamir¶

Abstract

Video-on-Demand (VoD) services require frequent updates in file configuration on the storage subsystem, so as to keep up with the frequent changes in movie popularity. This defines a natural *reconfiguration problem* in which the goal is to minimize the cost of moving from one file configuration to another. The cost is incurred by file replications performed throughout the transition. The problem shows up also in production planning, preemptive scheduling with set-up costs, and dynamic placement of Web applications. We show that the reconfiguration problem is NP-hard already on very restricted instances. We then develop algorithms which achieve the optimal cost by using servers whose load capacities are increased by $O(1)$, in particular, by factor $1+\delta$ for any small $0 < \delta < 1$ when the number of servers is fixed, and by factor of $2+\varepsilon$ for arbitrary number of servers, for some $\varepsilon \in [0, 1)$. To the best of our knowledge, this particular variant of the *data migration* problem is studied here for the first time.

Keywords: Data placement, reconfiguration, video-on-demand, approximation algorithms.

1 Introduction

Video on Demand (VoD) services have become common in library information retrieval, entertainment and commercial applications. In a VoD system, clients are connected through a network to a set of servers which hold a large library of video programs. Each client can choose a program he wishes to view and the time he wishes to view it. The service should be provided within a small latency and guaranteeing an almost constant transfer rate of the data. The transmission of a movie to a client requires the allocation of unit load capacity (or, a *data stream*) on a server which holds a copy of the movie.

Since video files are typically large, it is impractical to store copies of all movies on each server. Moreover, as observed in large VoD systems (see, e.g., [6, 21]), the distribution of accesses to movie files is highly skewed; indeed, only small fraction of the movies are requested

*A preliminary version of this paper appeared in the Proceedings of the 7th Workshop on Approximation and Online Algorithms, Copenhagen, September 2009.

†This work was partially supported by the Ministry of Trade and Industry MAGNET program through the NEGEV Consortium (www.negev-initiative.org) and by the Israel Science Foundation (grant number 1574/10).

‡Computer Science Department, Technion, Haifa 32000, Israel. E-mail: hadas@cs.technion.ac.il

§Computer Science Department, Technion, Haifa 32000, Israel. E-mail: galtamir@cs.technion.ac.il

¶School of Computer science, The Interdisciplinary Center, Herzliya, Israel. E-mail: tami@idc.ac.il.

frequently, while the vast majority (i.e., more than 80%) of the movies are rarely accessed. Hence, the number of copies held for each movie needs to reflect the frequency of accesses to this movie. The goal is to store the movie files on the servers in a way which enables to satisfy as many client requests as possible, subject to the storage and load capacity constraints of the servers.

Formally, suppose that the system consists of M video program files and N servers. Each movie file i, $1 \leq i \leq M$, is associated with a popularity parameter $p_i^0 \in (0,1]$, where $\sum_{i=1}^{M} p_i^0 = 1$. Each server j, $1 \leq j \leq N$, is characterized by (i) its storage capacity, C_j, that is the number of files that can reside on it,[1] and (ii) its load capacity, L_j, which is the number of data streams that can be read simultaneously from that server. For a given popularity vector $\{p_1^0, \ldots, p_M^0\}$, the *broadcast demand* of file i is $D_i^0 = p_i^0 \mathcal{L}$, where $\mathcal{L} = \sum_{j=1}^{N} L_j$ is the total load capacity of the system.[2] The *data placement problem* is to determine a placement of file copies on the servers and the amount of load capacity assigned to each file copy, so as to maximize the total amount of broadcast demand satisfied by the system. A solution for the placement problem can be represented as two $M \times N$ matrices: (i) The *placement matrix*, A, a $\{0,1\}$-matrix, $A_{i,j} = 1$ iff a copy of movie file i is stored on server j. (ii) The *broadcast matrix* B. $B_{i,j} \in \{0, 1, \ldots, L_j\}$. $B_{i,j}$ is the number of broadcasts of movie i transmitted from server j. A legal placement has to satisfy the following conditions:

- $A_{i,j} = 0 \Rightarrow B_{i,j} = 0$. Clearly, server j can transmit broadcasts of movie i only if it holds a copy of this movie.
- For each server j, $\sum_i B_{i,j} \leq L_j$, that is, the total number of broadcasts transmitted from server j does not exceed its load capacity.
- For each server j, $\sum_i A_{i,j} \leq C_j$, that is, the number of files stored on server j does not exceed its storage capacity.

A placement is *perfect* if it satisfies the broadcast demands of all movie files. Formally, $\forall i$, $\sum_j B_{i,j} = D_i^0$. Under certain conditions, it is known that a perfect placement always exists (see Section 1.2).

The above *static* data placement problem captures well the goal of maximizing throughput in periods of time where broadcast requirements remain unchanged.[3] However, in general, throughout the operation of a VoD system new movies are released and may become most popular, while the popularity of the previously *hot* movies drops. The system should be able to support any change in the distribution on file popularities. Thus, in order to maintain high throughput, the system needs to adjust the placement of file copies and the allocation of load capacity to these copies. This involves replications and deletions of files. File replications incur significant cost as they require bandwidth and other resources on the source, as well as the destination server. Minimizing this cost is crucial for optimizing system performance. This is the focus of our paper.

[1] Unless specified otherwise, we assume that all files have the same size.

[2] The broadcast demands are assumed to be integers. Rounded values can be obtained by standard solutions for the apportionment problem [22].

[3] In VoD system design, this is also known as the *static phase* [20].

Our *dynamic* data placement problem can be formalized as follows. Given a perfect placement of file copies on the servers, with the popularity vector $\langle p_1^0, \ldots, p_M^0 \rangle$, suppose that the popularity vector changes to $\langle p_1, \ldots, p_M \rangle$, with the corresponding broadcast demands $\langle D_1, \ldots, D_M \rangle$. The *reconfiguration problem* is to modify the initial data placement to a perfect placement for $\langle D_1, \ldots, D_M \rangle$ at minimum total cost. In updating system configuration, the cost of storing a new copy of movie file i on server j is given by $s_{i,j}$, while the assignment of load capacity to existing copy of file i on server j is free. We denote by $c_{i,j}$ the cost of having a copy of movie i on server j after the reconfiguration. Given the initial placement matrix A, we denote by A' the placement after reconfiguration. Then, by definition, $c_{i,j} = 0$ if $A_{i,j} = 1$, and $c_{i,j} = s_{i,j}$ if $A_{i,j} = 0$ and $A'_{i,j} = 1$. In other words, the cost of increasing the (i,j)-entry in the assignment matrix, A, is $s_{i,j}$ while changes in the broadcast matrix B are free. The total cost of switching from a placement A to a placement A' is given by $\sum_{i,j|A'_{i,j}=1.} c_{i,j}$. Note that file deletions incur no cost. Clearly, the new assignment must satisfy the three legal-placement conditions.

A VoD system is *homogeneous* if all servers have the same load capacities, i.e., $L_1 = \cdots = L_N = L$, and the same storage capacities, i.e., $C_1 = \cdots = C_N = C$ (see, e.g., [5, 10]). In this paper we assume that the system is *semi-homogeneous*, i.e., all servers have the same load capacities, but may have arbitrary storage capacities.

Example 1: Consider a system of two servers which holds 6 movies. The popularity vector is $\langle 0.05, 0.6, 0.05, 0.15, 0.05, 0.1 \rangle$. Both servers have the same load capacity $L_1 = L_2 = 10$, while the storage capacities are $C_1 = 3, C_2 = 4$. Having $\mathcal{L} = 20$, the demand vector is $D^0 = \langle 1, 12, 1, 3, 1, 2 \rangle$. Figure 1(a) presents a possible perfect placement for this instance. The assignment is described by a bipartite graph, in which the left hand side nodes represent movie files and the right hand side nodes represent servers; an edge (i,j) implies that a copy of movie file i is stored on server j. The maximal degree of a server-node is its storage capacity. Assume that the popularity vector is changed to $\langle 0.1, 0.15, 0.05, 0.15, 0.45, 0.1 \rangle$. Figure 1(b) presents a new placement, obtained from the previous one by adding (and deleting) copies of two files. The new placement is perfect for the new demand vector $D = \langle 2, 3, 1, 3, 9, 2 \rangle$. The corresponding assignment and broadcast matrices are given in Table 1. The reconfiguration cost is $c_{1,2} + c_{5,1}$.

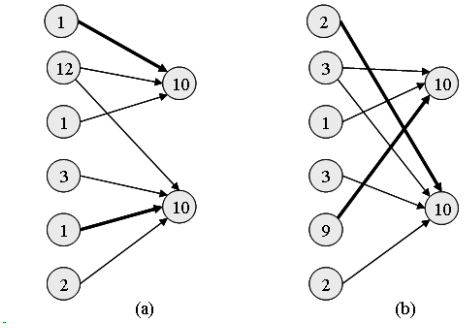

Figure 1: A perfect placement before (a) and after (b) the popularity change. Bold edges represent changes in storage assignment.

Applications: As mentioned above, a main motivation for this work comes from the constant need for dynamic data placement in VoD systems. Our reconfiguration problem shows up also in production planning, as well as in machine scheduling (see a survey in [17]). Suppose that

A	s_1	s_2
1	1	0
2	1	1
3	1	0
4	0	1
5	0	1
6	0	1

B	s_1	s_2
1	1	0
2	8	4
3	1	0
4	0	3
5	0	1
6	0	2

A'	s_1	s_2
1	0	1
2	1	1
3	1	0
4	0	1
5	1	0
6	0	1

B'	s_1	s_2
1	0	2
2	0	3
3	1	0
4	0	3
5	9	0
6	0	2

Table 1: The assignment and broadcast matrices of the placement before (left) and after (right) the popularity change.

M tasks are processed by N machines. Each machine has limited amount of resources and a time interval in which it is active. The resource requirements of the tasks are changing over time. Tasks may need to be reassigned to the machines in order to fit their new requirement. Reassignment of tasks incurs some cost due to migration overhead and the set-up of the machines. The goal is to reassign the tasks to the machine so as to minimize the transition cost. Finally, our problem naturally arises in dynamic placement of clustered Web applications (see, e.g., [8]). Web applications are dynamically placed on server machines so as to adjust system configuration to the availability of resources. The goal is to maximize the amount of client demands that can be satisfied by the applications while minimizing the number of placement changes.

1.1 Our Results

We first show (in Section 2) that the reconfiguration problem is NP-hard, already when the system consists of two servers, with unit reconfiguration costs and very restricted changes in file popularity. In practical scenarios, it is often the case that the new popularity vector has a perfect placement. This occurs, e.g., where the new vector is a *permutation* of the initial vector, that is, the popularity distribution function remains unchanged. For such scenarios, we give in Sections 3 and 4 algorithms which solve the reconfiguration problem optimally, by using servers whose load capacities are increased by a small constant factor. Specifically, for a fixed number of servers, we give in Section 3 an algorithm which accepts as parameter a value $0 < \delta < 1$ and achieves the optimal reconfiguration cost by using servers whose load capacities are $L(1 + \delta)$. The running time of the algorithm depends on the value of δ For more general inputs, in which the number of servers may be arbitrarily large, we give in Section 4 an algorithm that achieves the optimal cost, by using servers whose load capacities are increased by factor of $2+\varepsilon$, for some $\varepsilon \in [0, 1)$.

Our main approximation technique, applied (in Section 4) to general instances of the reconfiguration problem, relies on finding a linear programming relaxation whose optimal (fractional) solution is a lower bound on the optimal solution for our problem, and for which we can apply rounding without increasing the total cost. To find such a relaxation, we iteratively modify the initial linear relaxation for our problem until we obtain a linear program which reduces our problem to job scheduling on unrelated machines. It is worth noting that even though the optimal *integral* solutions for the programs in this sequence cannot be related to the optimum cost for our problem, it holds that the optimal (fractional) solution for each program is a lower bound for the optimum cost for our problem.

1.2 Related Work

The data placement problem has been extensively studied (see, e.g., [20, 5, 10, 18, 8] and a comprehensive survey in [9]). The paper [16] considers the problem of finding a *perfect placement* of movie files on the servers. The paper shows the hardness of the perfect placement problem and that such a placement always exists, e.g., when $\sum_{j=1}^{N} C_j \geq M + N - 1$. The paper [16] also presents an algorithm for the data placement problem, for inputs in which the ratio L_j/C_j is equal for all $1 \leq j \leq N$ (*uniform ratio servers*). The paper shows that the algorithm achieves a ratio of $1 - 1/(1 + C_{min})$ to the optimal, where $C_{min} = \min_j C_j$. Golubchik et al. gave in [5] a tighter analysis of this algorithm and showed that it achieves the ratio $1 - 1/(1 + \sqrt{C_{min}})^2$, and that this ratio is optimal for *any* algorithm for this problem. The paper [5] also presents a PTAS for the data placement problem with uniform ratio servers. Later papers considered a generalized version of the problem, where files may be of different sizes (see, e.g., [10, 18]).

For the more realistic model, where file popularities may change over time, there has been some earlier work which refers to the resulting *data migration* problem: Compute an efficient plan for moving data stored on devices (e.g., a set of servers) in a network from one configuration to another. Since the servers are constrained in handling simultaneous transmissions of files, data migration is done in rounds, where each round handles the delivery of a subset of the files to their destinations. Common objective functions are minimizing the makespan of the migration schedule, or the sum of completion times of the servers (see, e.g., [12, 13]). The paper [11] considers a somewhat 'dual' reconfiguration problem: the goal is to convert the existing layout to a good new layout (that is part of the solution), using a limited number of migration rounds. Surveys of known results for the data migration problem are given in [11, 4]. The data migration problem differs from our reconfiguration problem in several ways. (*i*) The final configuration is given as part of the input for data migration, while it is part of the solution for our problem. (*ii*) In data migration the output is a migration schedule, while no assignment schedule is output when solving the reconfiguration problem, and finally, (*iii*) in data migration we measure the quality of the migration schedule, while in our problem we measure the cost of the final configuration.

There has been some other work on reconfiguration of data placement, in which heuristic solutions were investigated through experimental studies (e.g., [14, 23, 3, 7]). The paper [8] studies a generalization of our reconfiguration problem, in which file deletions incur unit costs, and the files are of arbitrary sizes. The paper presents experimental results for greedy-based heuristics for the problem. We are not aware of earlier theoretical results for the reconfiguration problem.

2 Hardness Result

We show that the reconfiguration problem is NP-hard even if the system consists of only two servers, and even if popularity changes are limited such that the new popularity vector is a permutation of the previous one. In other words, the popularity *distribution function* is preserved. We use a reduction from a variant of the subset-sum problem. For a set of integers X, let S_X denote the total size of elements in X.

Definition 2.1 *The* smallest subsets with a given difference *problem is defined as follows. Given are two sets of non-negative integers* $X = \{x_1, x_2, \ldots x_{n_X}\}$ *and* $Y = \{y_1, y_2, \ldots, y_{n_Y}\}$*, and an*

integer z. W.l.o.g, $n_X \leq n_Y$. It is known that there exists a subset $Y'' \subseteq Y$ of size n_X satisfying $S_X = S_{Y''} + z$. The goal is to find the smallest integer $k \geq 1$ such that there exist $X' \subseteq X$ and $Y' \subseteq Y$, where $|X'| = |Y'| = k$, and $S_{X'} = S_{Y'} + z$. Note that such an integer k must exist, since for $k = n_X$, the sets $X' = X, Y' = Y''$ form a solution.

Example 2: Let $X = \{2, 3, 4, 5\}$, $Y = \{1, 2, 3, 4, 5\}$, and $z = 4$. For $Y'' = \{1, 2, 3, 4\}$ it holds that $|Y''| = n_X = 4$ and $S_X = 14 = S_{Y''} + z$. For this instance, the required k is 1 since there are two subsets of size 1, specifically, $X' = \{5\}, Y' = \{1\}$, such that $S_{X'} = 5 = S_{Y'} + z$.

Lemma 2.1 *The smallest subsets with a given difference problem is NP-hard.*

Proof: We show that the corresponding decision problem is NP-hard. That is, given X, Y, z, k, such that there exists a subset $Y'' \subseteq Y$ of size n_X satisfying $S_X = S_{Y''} + z$, the goal is to decide if there exist subsets X', Y', each having k elements and $S_{X'} = S_{Y'} + z$. It is easy to verify that the minimization problem is solvable in polynomial time if the decision problem is.

The reduction is from the *cardinality subset-sum* problem, which is known to be NP-hard [2]. Given a set $X = x_1, x_2, \ldots x_{n_X}$ of positive integers, a cardinality constraint $k < n_X$, and a target integer w, the goal is to find a subset $X' \subseteq X$ such that $|X'| = k$ and $S_{X'} = w$. W.l.o.g, we assume that (i) $w > k$ (else, a solution can be found by checking if X contains k units), (ii) the largest k elements in X have total size at least w (else, the answer is trivially negative), and (iii) all the elements in X are integers larger than 1 (else, X, w can be scaled).

Given X, k, w, an instance for cardinality subset-sum, construct the following instance for the decision problem of *smallest subsets with a given difference*: $k = k$, $z = w - k$, $X = X$, Y consists of $n_X - 1$ units, and a single element of value $S_X - n_X + 1 - w + k$. Note that the instance is defined properly since, by assumption (i), z is positive, and by assumption (ii), the last element in Y is positive. Also, as required, for $Y'' = Y$ we get that Y has a subset of n_X elements of total size $n_X - 1 + S_X - n_X + 1 - w + k = S_X - z$.

Claim 2.1 *The set X has a subset of k elements having total size w if and only if the answer to the above decision problem of* smallest subsets with a given difference *is positive.*

Proof: There are only two types of subsets of k elements in Y. The first one has k unit elements. For this type of subset, X has a subset X' of k elements summing up to w iff $S_{X'} = w = k + z = S_{Y'} + z$, that is, iff the required subsets exist. The other type of subset $Y' \subseteq Y$ of k elements consists of the last element and $k - 1$ units. The total size of elements in this subset is $S_X - n_X + 1 - w + k + (k - 1) = S_X - n_X - z + k$. We show that no subset of k elements from X can sum to this value plus z. Let X' be a subset of k elements from X. There are $n_X - k$ elements in $X \setminus X'$, whose total size is at least $2(n_X - k)$ (by assumption (iii)). Thus, $S_{X'} \leq S_X - 2(n_X - k)$, which is strictly smaller than $S_X - n_X + k = S_{Y'} + z$. □
This completes the proof of the lemma. □

Based on the hardness of the *smallest subsets with a given difference* problem, we can prove the following.

Theorem 2.2 *The reconfiguration problem is NP-hard already for a system of two servers having identical load capacities, with uniform replication costs, and even if popularity changes are restricted such that the new popularity vector is a permutation of the previous one.*

Proof: We reduce the *smallest subsets with a given difference* problem to a particular instance of the reconfiguration problem. We first show the hardness for two servers with different load capacities, and then extend it to two servers having the same load capacity. Given X, Y, z, such that there exists a subset $Y'' \subseteq Y$ of size n_X satisfying $S_X = S_{Y''} + z$, consider the following instance of reconfiguration: $M = n_X + n_Y$; the demands are $D^0 = \langle Y'', Y \setminus Y'' \cup X \rangle$, where Y'' is a vector consisting of the n_X elements of Y'', and $Y \setminus Y'' \cup X$ is a vector of n_Y elements consisting of elements from $Y \setminus Y''$ followed by the elements of X. The system has two servers with $C_1 = n_X,\ L_1 = S_X - z,\ C_2 = n_Y,\ L_2 = S_Y + z$. A possible perfect placement is to store the first n_X movie files on the first server, and the remaining n_Y movie files on the second server. The load capacities of the servers satisfy exactly the broadcast demands. Assume further that the demands are changed to be the values of the elements in X, Y. Specifically, $D = \langle x_1, x_2, \ldots x_{n_X}, y_1, y_2, \ldots, y_{n_Y} \rangle$. Note that the total demand of the first n_X movies is increased by z, while the total demand of the remaining n_Y movies is decreased by z. Finally, let $s_{i,j} = 1$ be the uniform replication cost.

Figure 2 depicts the reconfiguration problem induced by Example 2 above. The system consists of $M = 9$ movies and two servers having parameters $C_1 = 4, L_1 = 10, C_2 = 5, L_2 = 19$. Figure 2(a) shows a perfect assignment for the demand vector $D^0 = \langle 1, 2, 3, 4, 5, 2, 3, 4, 5 \rangle$. Assume that the popularity changes so that the new demand vector is $D = \langle 2, 3, 4, 5, 1, 2, 3, 4, 5 \rangle$. The guaranteed Y'' implies the solution (b) having cost 8. An optimal solution (c) has cost 2.

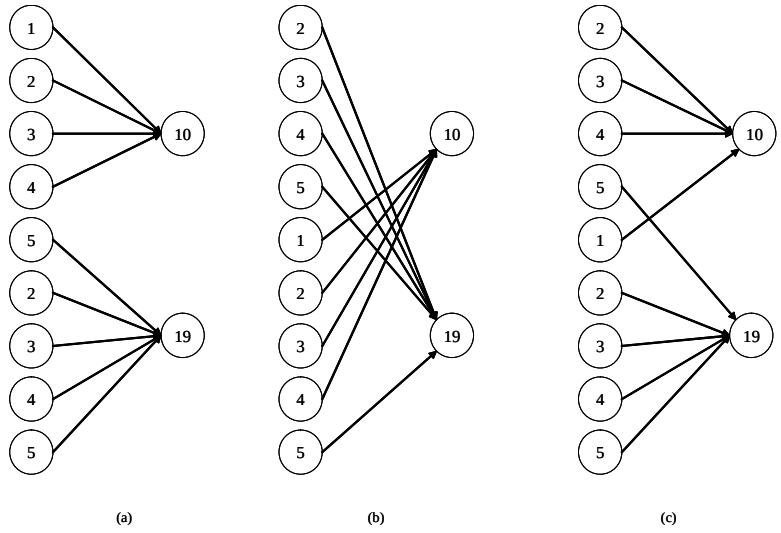

Figure 2: The reconfiguration problem induced by Example 2. (a) initial assignment, (b) reconfiguration having cost 8, and (c) an optimal reconfiguration having cost 2.

Since the total storage capacity of the servers is exactly $n_X + n_Y$, in any perfect placement there is exactly one copy of each movie stored on one of the two servers. Thus, the reconfiguration in this case consists of swapping the storage of $2k$ movie files. By definition of the subset problem, it is known that there exists a perfect assignment that can be achieved by swapping $2n_X$ movie files (of Y'' and X). An optimal reconfiguration swaps the minimal number of files. Thus, an optimal solution for the reconfiguration problem specifies subsets $X' \subseteq X$, and $Y' \subseteq Y$ such

that $|X'| = |Y'| = k$, $S_{X'} = S_{Y'} + z$, and k is minimal. The storage capacity is clearly preserved by this swapping. Also, the total demand of the movies assigned to the first server is now $S_{X \setminus X'} + S_{Y'} = S_X - z = L_1$. Similarly, the total demand of the movies assigned to the second server is now $S_{Y \setminus Y'} + S_{X'} = S_Y + z = L_2$. Therefore, the resulting assignment is perfect. In general, any reconfiguration that ends up with a perfect assignment, corresponds to two subsets X', Y' of X, Y respectively, for which $|X'| = |Y'|$ and the difference in the total new demand of the corresponding movies is z. We conclude that a solution for the reconfiguration problem induces a solution for the *smallest subsets with a given difference* problem, implying that the reconfiguration problem is NP-hard.

In order to extend the hardness result for two servers having the same load capacity, we add dummy movies whose demand does not change and that are not reassigned in any optimal solution. Specifically, given X, Y, z, such that there exists a subset $Y'' \subseteq Y$ of size n_X satisfying $S_X = S_{Y''} + z$, consider the following instance of reconfiguration: Let $W > S_X + S_Y$ be a large constant. Then $M = 2n_X + n_Y + 2$; the demands are $D^0 = \langle W(n_X+1) + S_Y - S_X + 2z, Y'', Y \setminus Y'' \cup X, W, W, \ldots, W \rangle$. That is, we add $n_X + 2$ dummy movies. We 'wrap' the vector D^0 described in the reduction for servers with different load capacities with a single first movie that has demand $W(n_X + 1) + S_Y - S_X + 2z$, and $n_X + 1$ last movies each having demand W. In our example, let $W = 100$, then $D^0 = \langle 509, 1, 2, 3, 4, 5, 2, 3, 4, 5, 100, 100, 100, 100, 100 \rangle$. The system has two servers with $C_1 = n_X + 1$, $C_2 = n_Y + n_X + 1$, and $L_1 = L_2 = W(n_X + 1) + S_Y + z$. In our example, $C_1 = 5, C_2 = 10, L_1 = L_2 = 519$. A possible perfect placement is to store the first $1 + n_X$ movie files on the first server, and the remaining $n_Y + n_X + 1$ movie files on the second server. The load capacities of the servers satisfy exactly the broadcast demands. Assume further that the demands are changed to be $D = \langle W(n_X + 1) + S_Y - S_X + 2z, x_1, x_2, \ldots x_{n_X}, y_1, y_2, \ldots, y_{n_Y}, W, W, \ldots, W \rangle$. In our example, let $D = \langle 509, 2, 3, 4, 5, 1, 2, 3, 4, 5, 100, 100, 100, 100, 100 \rangle$. We claim that for the new demand vector D, in any optimal reconfiguration, none of the dummy movies are migrated. Indeed, W is big enough to assure that the only feasible reassignment of the dummy movies is when the first dummy movie is assigned to the second server and all the $n_X + 1$ additional dummy movies are assigned to the first server. Such a reassignment leaves no storage on the first server, and is therefore unfeasible. Therefore, none of the dummy movies is migrated and the problem reduces to the problem considered for servers with variable load capacities. □

3 Minimal Cost Algorithm for Fixed Number of Servers

In this section we present a polynomial time algorithm that finds a minimal-cost reconfiguration for a semi-homogeneous system, assuming that the number of servers, N, is some fixed constant. Given an instance I with N servers of load capacity L and arbitrary storage capacity C_j, $1 \leq j \leq N$, denote by $OPT(I)$ the minimum reconfiguration cost for I. Our Reconfiguration algorithm (given in Figure 3) outputs a placement whose cost is at most $OPT(I)$, on servers of load capacities $(1 + 2\delta)L$, for a small parameter $\delta \in (0, 1]$.

Given the parameter $0 < \delta \leq 1$, we say that a movie file i is *big* if $D_i \geq \delta L$; otherwise, movie i is *small*. Our algorithm handles separately the two types of movies. It produces allocations with the following properties:

(P_1) For any big movie i, and any server j, the broadcast allocation $B_{i,j}$ of server j to movie i is an integral multiple of $\delta^2 L/N$, i.e., $B_{i,j} = k\delta^2 L/N$, where k is an integer in $[1, N/\delta^2]$.

Reconfiguration Algorithm

For each storage and load allocation to the big movies satisfying (P_1) do:

Let L_j^b and C_j^b be the total load and storage allocation to big movies on server j.

For all $1 \leq j \leq N$ do $L_j := L - L_j^b + 2\delta L$ and $C_j := C_j - C_j^b$

Find a minimum cost placement of the small movies on the servers assuming server j has load capacity L_j and storage capacity C_j (see below).

Select a configuration for the big movies which yields minimum total cost.

Figure 3: Algorithm for updating data placement on a set of servers.

(P_2) Each small movie is stored on a single server, on which it is allocated all of its broadcast demand. Formally, for any small movie i, for a single server j, $B_{i,j} = D_i$, and for any $j' \neq j$, $B_{i,j'} = 0$.

Note that the algorithm finds a minimum cost placement for the small movies by initially solving a linear programming relaxation of the problem (given by LP_0). The optimal fractional solution is then rounded to an integral placement. This integral placement has the guarantee that, if the big movies are placed in correspondence with an optimal solution, then the total cost of the computed placement for the whole instance is minimal, using servers with enlarged load capacities.[4]

Towards analyzing our Reconfiguration algorithm, we prove some technical lemmas. We first show that, by allowing a slight increase in the load capacities, we can find in polynomial time a reconfiguration satisfying (P_1) whose total cost is at most $OPT(I)$.

Lemma 3.1 *Restricting the allocation of the big movies to one that satisfies* (P_1) *may require an increase of at most* δL *in the load capacity of each server, with no change in the reconfiguration cost.*

Proof: We show that any assignment of the big movies in which the total broadcast allocation of each server is at most L can be converted into one which satisfies (P_1). The only changes are in the broadcast matrix, B. The assignment matrix, A, remains unchanged, therefore, the reconfiguration cost is the same.

Consider the servers one after the other. For each server j and big movie i, round $B_{i,j}$ up to the next multiple of $\delta^2 L/N$. Note that there are at most N/δ such movies in the system; thus, there are at most N/δ such movies on each server. It follows that the additional load on each server due to rounding is at most $\frac{N}{\delta} \cdot \frac{\delta^2}{N} L = \delta L$. □

We now bound the reconfiguration cost for adding the small movies, once we have a 'good' configuration of the big movies. Recall that, for any server $1 \leq j \leq N$, C_j^b, L_j^b denote the total storage capacity and the total load capacity assigned to big movies on server j, respectively.

Lemma 3.2 *Given a configuration of the big movies that corresponds to an optimal solution, let* $C_j = C_j - C_j^b$ *and* $L_j = L - L_j^b + \delta L$. *Then, there exists a polynomial time algorithm that finds for the small movies a placement satisfying property* (P_2), *such that the storage and load*

[4] We give the precise details in the proof of Claim 3.1.

capacities used by small movies on server j are at most C_j and $L_j + \delta L$, respectively, and the total reconfiguration cost is at most $OPT(I)$.

Proof: Let R denote the set of small movies, and $M_r = |R|$. Index the small movies $1, \ldots, M_r$. Denote by $x_{i,j} \in \{0, 1\}$ an indicator variable for the assignment of movie i to server j, $i \in R$ and $1 \leq j \leq N$. The costs $c_{i,j}$ are the given replication costs. Note that once the big movies have been assigned, the servers may have different residual load capacities. To simplify the analysis of our algorithm, we describe a transformation of the residual load capacities, which yields identical load capacity for all the servers.

Let $\hat{L} \geq 1$ be an integer. For a movie i and server j, let $D_{i,j} = D_i \cdot \hat{L}/L_j$ be the broadcast demand of movie i if this movie is assigned only to server j. We may now consider the problem of assigning movies with demands D_{ij}, $1 \leq i \leq M_r$, $1 \leq j \leq N$, on servers whose load capacity is equal to $\hat{L}$.

The assignment will be determined by rounding the solution for the following linear program, LP_0, in which x_{ij} is the fraction of movie i assigned to server j, $1 \leq i \leq M_r$.

$$
\begin{array}{lll}
(LP_0): & \text{minimize} & \displaystyle\sum_{i=1}^{M_r}\sum_{j=1}^{N} x_{i,j} \cdot c_{i,j} \\
& \text{subject to:} & \displaystyle\sum_{i=1}^{M_r} x_{i,j} \cdot D_{i,j} \leq \hat{L} \qquad \text{for } 1 \leq j \leq N, \qquad (1) \\
& & \displaystyle\sum_{i=1}^{M_r} x_{i,j} \leq C_j \qquad \text{for } 1 \leq j \leq N, \qquad (2) \\
& & \displaystyle\sum_{j=1}^{N} x_{i,j} = 1 \qquad \text{for } 1 \leq i \leq M_r, \qquad (3) \\
& & x_{i,j} \geq 0 \qquad \text{for } \ 1 \leq j \leq N, \ \ 1 \leq i \leq M_r.
\end{array}
$$

We note that LP_0 is a relaxation of our problem. Consider an assignment of the movies to the servers that corresponds to an optimal solution. By Lemma 3.1, there exists such an assignment which satisfies (P_1). Denote this assignment by $S_{OPT}(I)$. Let $OPT_b(I)$ be the cost incurred by the big movies in $S_{OPT}(I)$, and denote by x^*_{ij} the fraction of D_i, the total demand of small movie i, assigned to server j. We argue that $\mathbf{x}^* = \{x^*_{ij}, 1 \leq i \leq M_r, 1 \leq j \leq N\}$ is a feasible solution for LP_0, whose cost is at most $OPT(I) - OPT_b(I)$. Clearly, by the definition of D_{ij}, constraints (1) are satisfied for the solution $\mathbf{x}^*$. Constraints (2) are also satisfied for $\mathbf{x}^*$. Indeed, in $S_{OPT}(I)$, the allocation of a fraction $x^*_{ij} > 0$ of the demand of movie i to server j requires one unit of storage, while in LP_0 the amount of storage allocated is $0 < x^*_{ij} \leq 1$. Finally, since in $S_{OPT}(I)$ each movie is fully assigned to the servers, constraints (3) are also satisfied. As for the cost of LP_0 for the solution $\mathbf{x}^*$, we note that, if $x^*_{ij} > 0$, then the cost for assigning movie i to server j in $S_{OPT}(I)$ is $c_{ij} \geq x^*_{ij} \cdot c_{ij}$.

The following property of an optimal solution for LP_0 will be useful in the rounding process.

Claim 3.1 *Given an optimal solution for LP_0 with cost C, there exists a polynomial time algorithm which finds an assignment of the small movies, such that on each server at most one small movie cannot be allocated all of its load requirement. The total cost of this assignment is at most C.*

Proof: Given an optimal (fractional) solution for LP_0, we use a rounding technique of Shmoys and Tardos [19]. Specifically, we construct a bipartite graph in which server j is represented by at most C_j vertices, $1 \leq j \leq N$. A solution for the fractional matching problem on this graph induces an integral matching of the same cost with the same cardinality.

Formally, sort the small movies assigned to server j in non-increasing order by their load requirements on this server. Let $G_B = (V \cup U, E)$ be a bipartite graph, where $U = \{u_i | 1 \leq i \leq M_r\}$ represents the set of small movies, and V is the set of server vertices: $V = \{v_{j,k} | 1 \leq j \leq N, 1 \leq k \leq \sigma_j\}$ where $\sigma_j = \lceil \sum_{i=1}^{M_r} x_{i,j} \rceil$ is the total number of small movies stored on server j. Clearly, $\sigma_j \leq C_j$. The vertices $v_{j,1}, \ldots, v_{j,\sigma_j}$ represent server j, $1 \leq j \leq N$.

The set of edges E of G_B is defined as follows. Given the values of $x_{i,j}$ for $1 \leq i \leq M_r$, $1 \leq j \leq N$, for any server j:

(i) If $\sum_{i=1}^{M_r} x_{i,j} \leq 1$ then there is a single vertex $v_{i,1} \in V$ corresponding to server j. In this case, for any $1 \leq i \leq M_r$ such that $x_{i,j} > 0$, we add in G_B an edge $(u_i, v_{j,1})$, and set its weight to be $w(u_i, v_{j,1}) = x_{i,j}$.

(ii) If $\sum_{i=1}^{M_r} x_{i,j} > 1$, find the minimum index i_1 such that $\sum_{i=1}^{i_1} x_{i,j} \geq 1$, then E contains all the edges $(u_i, v_{j,1})$, $1 \leq i \leq i_1 - 1$ for which $x_{i,j} > 0$. For each of these edges set $w(u_i, v_{j,1}) = x_{i,j}$. Now, add to E an edge $(u_{i_1}, v_{j,1})$, whose weight is $w(u_{i_1}, v_{j,1}) = 1 - \sum_{i=1}^{i_1-1} w(u_i, v_{j,1})$. Thus, the sum of weights of the edges incident to $v_{j,1}$ is exactly 1. If $\sum_{i=1}^{i_1} x_{i,j} > 1$ add an edge $(u_{i_1}, v_{j,2})$, whose weight is $w(u_{i_1}, v_{j,2}) = (\sum_{i=1}^{i_1} x_{i,j}) - 1$. Proceed next to movies with $i > i_1$ i.e., those with smaller broadcast requirements on server j. Similar to the above process for $v_{j,1}$, add edges incident to $v_{j,2}$, until a total of exactly one movie is assigned to $v_{j,2}$, and so on. Let i' be the index of the last movie for which an edge is assigned this way, i.e, $i' = i_{\sigma_j - 1}$. Now, for any $i > i'$ for which $x_{i,j} > 0$ add an edge (u_i, v_{j,σ_j}) and set $w(u_i, v_{j,\sigma_j}) = x_{i,j}$.

The cost of an edge $(u_i, v_{j,k})$ is $c_{i,j}$, for $1 \leq i \leq M_r$, $1 \leq j \leq N$, and $1 \leq k \leq \sigma_j$. For each server vertex $v_{j,k}$, let $D_{j,k}^{max}$ ($D_{j,k}^{min}$) denote the maximum (minimum) of the broadcast requirements $D_{i,j}$ corresponding to the edges $(u_i, v_{j,k})$ incident to $v_{j,k}$. Then, for all $1 \leq j \leq N$ and $1 \leq k \leq \sigma_j - 1$,

$$D_{j,k}^{min} \geq D_{j,k+1}^{max}. \tag{4}$$

Recall that a vector $\mathbf{y}$ on the edges of a graph is a *fractional matching* if, for each vertex w, the sum of entries of $\mathbf{y}$ corresponding to the edges incident to w is at most 1. The fractional matching *exactly matches* a vertex w if the corresponding sum is equal to 1. A fractional matching is a *matching* if each entry of $\mathbf{y}$ is in $\{0, 1\}$. We note that the weight function on the edges of G_B defines a fractional matching, in which all movie vertices u_i, $1 \leq i \leq M_r$, and all server vertices $v_{j,k}$, $1 \leq j \leq N$, $1 \leq k \leq \sigma_j - 1$, are exactly matched. Thus, there exists an integral matching of the same cost that matches all the movie vertices (see, e.g., [15]).

We now summarize the steps of the rounding procedure which assigns the small movies to the servers.

1. Given an optimal solution for LP_0, form the bipartite graph G_B.

2. Find a min-cost (integer) matching H that exactly matches all movie vertices in G_B.

3. For each edge $(u_i, v_{j,k}) \in H$ place movie i on server j.

We show that the assignment obtained in Step 3. of the algorithm has cost C, and that the overall load capacity used on any server is at most $\hat{L}$ plus the maximum load generated by any small movie. By the above discussion, the integral matching found in Step 2. has cost C. Since the cost of the assignment is equal to the cost of the matching, the solution output by the algorithm has the optimal cost C.

Next, we bound the total load capacity used on any server. Consider the movies assigned to server j. For any $1 \leq j \leq N$, there are $\sigma_j \leq C_j$ vertices representing server j in G_B. Each of these vertices $v_{j,k}$ adds at most one movie file to server j (the movie which corresponds to the edge selected for the matching H, among those incident to $v_{j,k}$). Therefore, at most C_j small movies are assigned to server j. It follows that the total broadcast requirement of the movies on server j is at most

$$\begin{aligned}
\sum_{k=1}^{\sigma_j} D_{j,k}^{max} &= D_{j,1}^{max} + \sum_{k=2}^{\sigma_j} D_{j,k}^{max} \leq D_{j,1}^{max} + \sum_{k=1}^{\sigma_j - 1} D_{j,k}^{min} \\
&\leq D_{j,1}^{max} + \sum_{k=1}^{\sigma_j} \sum_{\{i|(u_i, v_{j,k}) \in E\}} D_{i,j} \cdot w(u_i, v_{j,k}) \\
&= D_{j,1}^{max} + \sum_{i=1}^{M_r} D_{i,j} x_{i,j} \leq D_{j,1}^{max} + \hat{L}
\end{aligned}$$

The second inequality follows from (4). □

Transforming back the load on server j from the scaled value to the original one, we have that $\hat{L}$ is converted to L_j; also, since $D_{j,1}^{max} \leq \max_{i,j} D_{i,j}$, we have that $D_{j,1}^{max}$ is transformed to $D_{i_{max}}$, for some $1 \leq i_{max} \leq M_r$. Thus, the total load on any server j, $1 \leq j \leq N$, is at most $L_j + \delta L$. □

The next lemma will be used to bound the running time of the Reconfiguration algorithm.

Lemma 3.3 *The set of possible configurations of copies of the big movies, along with the corresponding allocations of load capacities, has a polynomial size.*

Proof: Consider the entries corresponding to big movies in the broadcast matrix B. We show that there is a polynomial number of ways to fill these entries. For each server (column in B) there are at most N/δ positive entries in the corresponding column, each has a value $k\delta^2 L/N$, for an integer $k \in [1, N/\delta^2]$. Thus, there are at most $(N/\delta^2)^{N/\delta}$ possible ways to fill each column in B, and $(N/\delta^2)^{N^2/\delta}$ possible ways to determine the allocation to big movies. This value is polynomial since N is fixed. □

The main result of this section is the following.

Theorem 3.4 *Let $N \geq 1$ be a fixed constant. Given an instance I, with each server having load capacity L and arbitrary storage capacities $C_j \geq 1$, $1 \leq j \leq N$, the Reconfiguration algorithm finds in polynomial time a placement of the files whose cost is at most $OPT(I)$, by using servers with load capacities $L(1+2\delta)$.*

Proof: By Lemma 3.1, there exists an allocation of the movies whose total cost is at most $OPT(I)$, which satisfies (P_1), such that the load allocated on each server to big movies is at most $L(1+\delta)$. Let $OPT_b(I)$ be the reconfiguration cost for the big movies in this allocation. Since the Reconfiguration algorithm performs exhaustive search over all possible allocations of the big movies satisfying (P_1), the algorithm will find the desired allocation, for which LP_0 yields a cost at most $OPT(I) - OPT_b(I)$.

By Lemma 3.2, rounding an optimal solution, $C \leq OPT(I) - OPT_b(I)$, for LP_0 yields a feasible allocation for the small movies of cost at most C, which requires on each server a total load at most $L(1+2\delta)$.

Finally, by Lemma 3.3, the overall running time of the algorithm is at most $(N/\delta^2)^{N^2/\delta}$ multiplied by the time required to solve the linear program LP_0 (note that this dominates the running time of the minimum cost maximum matching subroutine used in the rounding step for LP_0). □

4 Minimal Cost Algorithm for Arbitrary Number of Servers

In this section we consider a system with *arbitrary* number of servers. We first show that when the number of movies is *fixed*, our problem can be optimally solved for any number of servers.

Theorem 4.1 *The reconfiguration problem is solvable in polynomial time when M, the number of movies, is fixed.*

Proof: Number the collection of subsets of the M movies by $1, \ldots, 2^M$, then the assignment of movie files to the N servers can be represented as an *assignment vector* of length 2^M, in which the kth entry gives the number of servers that contain the kth subset of movies. The number of possible assignment vectors is $\binom{2^M+N-1}{2^M-1}$. It is possible to compute the cost of each assignment vector as follows. Construct the bipartite graph $G_B = (U, V, E)$, in which $|U| = |V| = N$. Each vertex $u \in U$ corresponds to a server, and each vertex $v \in V$ corresponds to a subset of movies for which there is a non-zero entry in the assignment vector. In other words, if the kth entry in the vector is equal to h, $1 \leq h \leq N$, then in G_B there are h vertices in V which correspond to the kth subset of movies. There is an edge $(u, v) \in E$ if the server corresponding to u has storage capacity larger than the number of movies in the subset that corresponds to v; the cost of (u, v) is the cost of assigning this subset of movies on v. Next, solve the minimum weight perfect matching problem on G_B. The cost of each perfect matching (if one exists) is the cost of the corresponding assignment vector. Given a perfect matching of minimum cost, use a flow network to test the feasibility of the given assignment vector, namely, to verify that each movie i can be allocated its load requirement D_i. Finally, among the feasible assignments, select the one having the smallest cost. □

For the case where M may be arbitrarily large, we present below a polynomial time algorithm which finds a minimal-cost reconfiguration in a semi-homogeneous system. Given an instance

I, our algorithm outputs a minimal-cost placement, by using servers of load capacities $(2+\varepsilon)L$ where

$$\varepsilon = max_{\{i|D_i>L\}}\{D_i/L - \lfloor D_i/L \rfloor\} \text{ if } \max_i D_i > L, \text{ else } \varepsilon = 0. \quad (5)$$

4.1 The Algorithm

The following is an overview of the algorithm. (i) Partition the movies to *big* and *small*; movie i is big if $D_i > L$, else movie i is small. (ii) Solve a linear programming relaxation to obtain a lower bound on the optimal solution for the reconfiguration problem. (iii) Round the (fractional) solution of the linear program to obtain an integral solution of optimal cost. (iv) Use the integral solution to assign movie copies to N servers, where server j has storage capacity C_j and load capacity $(2+\varepsilon)L$, for some $\varepsilon \in [0,1)$ (as defined in (5)).

Solving an LP Relaxation In the following we show how a natural LP relaxation for our problem can be modified to obtain another linear program, from which we derive an optimal integral solution. Let $x_{ij} \in [0,1]$ denote the fraction of the load capacity L of server j allocated to big movie i. We denote by y_{ij} the fraction of D_i allocated to small movie i on server j. Also, $c_{i,j}$ is the given replication cost (which depends on the initial configuration). Consider the following linear programming relaxation, LP_1, for the reconfiguration problem.

$$
\begin{aligned}
(LP_1): \quad \text{minimize} \quad & \sum_{i\in Big}\sum_{j=1}^{N} x_{i,j}\cdot c_{i,j} + \sum_{i\in Small}\sum_{j=1}^{N} y_{i,j}\cdot c_{i,j} \\
\text{subject to:} \quad & \sum_{i\in Big} x_{i,j}\cdot L + \sum_{i\in Small} y_{i,j}\cdot D_i \le L \qquad \text{for } 1\le j\le N & (6)\\
& \sum_{i\in Big} x_{i,j} + \sum_{i\in Small} y_{i,j} \le C_j \qquad \text{for } 1\le j\le N & (7)\\
& \sum_{j=1}^{N} x_{i,j} = \frac{D_i}{L} \qquad \text{for } i\in Big & (8)\\
& \sum_{j=1}^{N} y_{i,j} = 1 \qquad \text{for } i\in Small & (9)\\
& 0\le x_{i,j}\le 1 \qquad \text{for } 1\le j\le N,\quad i\in Big \\
& 0\le y_{i,j}\le 1 \qquad \text{for } 1\le j\le N,\quad i\in Small
\end{aligned}
$$

Constraints (6) ensure that the total load capacity used by copies of the big movies and by the small movies on each server is at most L. Constraints (7) ensure that the total storage required on server j is at most C_j. The constraints (8) and (9), together with constraints (6), guarantee that each (big or small) movie is allocated D_i broadcasts.

Next, we modify LP_1 as follows. For any big movie i, let $k_i = \lfloor D_i/L \rfloor$. Consider the linear program LP_2. Note that constraints (10) allow to assign to big movie i some fraction of D_i/k_i on server j; also, constraints (12) guarantee that big movie i is allocated D_i broadcasts.

$$
\begin{aligned}
(LP_2): \quad \text{minimize} \quad & \sum_{i \in Big} \sum_{j=1}^{N} x_{i,j} \cdot c_{i,j} + \sum_{i \in Small} \sum_{j=1}^{N} y_{i,j} \cdot c_{i,j} \\
\text{subject to:} \quad & \sum_{i \in Big} x_{i,j} \cdot \frac{D_i}{k_i} + \sum_{i \in Small} y_{i,j} \cdot D_i \leq L \qquad \text{for } 1 \leq j \leq N \qquad (10) \\
& \sum_{i \in Big} x_{i,j} + \sum_{i \in Small} y_{i,j} \leq C_j \qquad \text{for } 1 \leq j \leq N \qquad (11) \\
& \sum_{j=1}^{N} x_{i,j} = k_i \qquad \text{for } i \in Big \qquad (12) \\
& \sum_{j=1}^{N} y_{i,j} = 1 \qquad \text{for } i \in Small \qquad (13) \\
& 0 \leq x_{i,j} \leq 1 \qquad \text{for } \ 1 \leq j \leq N, \quad i \in Big \\
& 0 \leq y_{i,j} \leq 1 \qquad \text{for } \ 1 \leq j \leq N, \quad i \in Small
\end{aligned}
$$

Now, we partition each big movie i to k_i sub-movies. Thus, we replace the variables $x_{i,j}$, $1 \leq j \leq N$, $i \in Big$ by the set of variables $x_{i,j,r}$, $1 \leq r \leq k_i$. Intuitively, we partition the load requirement of movie i to k_i, so we can now consider k_i sub-movies, where each needs to be allocated $\hat{D}_i = \frac{D_i}{k_i}$ broadcasts. We rewrite the linear programming relaxation as LP_3.

Rounding the Fractional Solution Note that LP_3 can be viewed as the linear programming relaxation of an input for job scheduling on unrelated machines, with costs and cardinality constraints, in which we need to schedule a set of jobs on N unrelated machines. The set of jobs $\mathcal{J}$ corresponds to all the small movies and the collection of sub-movies for the big movies, i.e., $|\mathcal{J}| = |Small| + \sum_{i \in Big} k_i$. The processing time of a job corresponding to a small movie i on machine j is $p_{ij} = D_i$, and the processing time of any of the jobs corresponding to the k_i sub-movies of big movie i on machine j is $p_{ij} = \lceil D_i/k_i \rceil$. Note that if $k_i = 1$ then $p_{ij} = D_i < 2L$. If $k_i > 1$ then $D_i/k_i < 1.5L$ implying (for all $L > 1$) $\lceil D_i/k_i \rceil < 2L$. In the reduction to the scheduling problem, the cost of processing job i on machine j is c_{ij}, $\forall \ \ i$ and $1 \leq j \leq N$. The makespan of any machine j is at most L, and the maximal number of jobs that can be assigned to machine j is C_j.[5] The goal is to schedule all jobs on the machines, subject to the makespan and cardinality constraints, so as to minimize the total cost. The rounding technique in [19] guarantees that, for given values C and T, a schedule of cost at most C and makespan at most $2T$ is generated, if there exists a (fractional) solution to the scheduling problem of cost C and makespan T.

[5] This is the *cardinality* constraint.

Given an optimal solution for LP_3, we can apply the rounding technique of Shmoys and Tardos [19], as described in the proof of Claim 3.1. The resulting integral solution can be used to determine the storage allocation. The assignment matrix is given by the variables $x_{i,j,r}, y_{i,j}$ and the broadcast matrix is given by the allocated processing time. Formally, for a small movie i, assign a single copy of i on server j if $y_{i,j} = 1$, i.e., $A_{i,j} = 1$ and $B_{i,j} = D_i$. For any big movie i, note that each sub-movie is allocated $\left\lceil \frac{D_i}{k_i} \right\rceil > L$ broadcasts.[6] Therefore, given that the makespan of the rounded solution is at most $2L$, for all i, j, $\sum_{r=1}^{k_i} x_{i,j,r} \in \{0, 1\}$. If $\sum_{r=1}^{k_i} x_{i,j,r} = 1$ then assign a copy of big movie i on server j, i.e., $A_{i,j} = 1$ and $B_{i,j} = \left\lceil \frac{D_i}{k_i} \right\rceil$.

$$
\begin{aligned}
(LP_3): \quad \text{minimize} \quad & \sum_{i \in Big} \sum_{r=1}^{k_i} \sum_{j=1}^{N} x_{i,j,r} \cdot c_{i,j} + \sum_{i \in Small} \sum_{j=1}^{N} y_{i,j} \cdot c_{i,j} \\
\text{subject to:} \quad & \sum_{i \in Big} \sum_{r=1}^{k_i} x_{i,j,r} \cdot \frac{D_i}{k_i} + \sum_{i \in Small} y_{i,j} D_i \leq L \qquad 1 \leq j \leq N \\
& \sum_{i \in Big} \sum_{r=1}^{k_i} x_{i,j,r} + \sum_{i \in Small} y_{i,j} \leq C_j \qquad 1 \leq j \leq N \\
& \sum_{j=1}^{N} x_{i,j,r} = 1 \qquad \text{for} \quad i \in Big, \quad 1 \leq r \leq k_i \\
& \sum_{j=1}^{N} y_{i,j} = 1 \qquad \text{for} \quad i \in Small \\
& 0 \leq x_{i,j,r} \leq 1 \qquad \text{for} \quad 1 \leq j \leq N, \quad i \in Big, \quad 1 \leq r \leq k_i \\
& 0 \leq y_{i,j} \leq 1 \qquad \text{for} \quad 1 \leq j \leq N, \quad i \in Small
\end{aligned}
$$

4.2 Analysis

Let $OPT(I)$ denote the cost of an optimal solution for an instance I of the reconfiguration problem.

Lemma 4.2 *The optimal solution for LP_3 is a lower bound for $OPT(I)$.*

Proof: Given an instance I, clearly, the optimal solution for LP_1 is a lower bound for $OPT(I)$. Also, any feasible solution for LP_1 is a feasible solution for LP_2; therefore, an optimal solution for LP_2 is a lower bound for the optimal solution of LP_1. Now, consider a feasible solution for LP_2. By setting in LP_3 $x_{i,j,r} = \frac{x_{i,j}}{k_i}$, for all $i \in Big$, $1 \leq r \leq k_i$, and $1 \leq j \leq N$, we get a feasible solution for LP_3. It follows that the optimal solution for LP_3 is a lower bound for the optimal solution of LP_2. This completes the proof. □

[6] In case k_i divides D_i, we can view big movie i as k_i small movies whose demands are equal to L. Thus, it is sufficient to consider here the case where $\lceil D_i/k_i \rceil > L$.

Theorem 4.3 *The above algorithm outputs in polynomial time a solution of cost at most $OPT(I)$. The movies can be stored on N servers with storage capacities $C_1, \ldots, C_N$ and load capacities $(2+\varepsilon)L$, where ε is defined in (5).*

Proof: By Lemma 4.2, the cost of an optimal solution for LP_3 is at most $OPT(I)$. As shown in the proof of Claim 3.1, the rounding procedure preserves the cost of the linear program. We now bound the extra amount of load capacity required for the placement of the files on the servers in the solution output by the algorithm. The maximal broadcast demand of any sub-movie of big movie i is at most $L(1+\varepsilon)$, where ε is defined in (5), and this is also the maximum processing time of any job in the input for the scheduling problem. Hence, by using the rounding technique of Shmoys and Tardos [19], the resulting assignment of files to server may require an increase of at most $L(1+\varepsilon)$ in the load capacity of any server.

Finally, the running time of the algorithm is dominated by the time required to solve the linear program LP_3, which is polynomial in the input size. □

Acknowledgment. We thank two anonymous referees for many valuable comments and suggestions.

References

[1] C.F. Chou, L. Golubchik, and J.C.S. Lui. A performance study of dynamic replication techniques in continuous media servers. *Proc. of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS)*, pp. 256–264, 2000.

[2] A. Caprara, H. Kellerer, U. Pferschy, D. Pisinger, Approximation algorithms for knapsack problems with cardinality constraints, *European Journal of Operational Research* 123, 333–345, 2000.

[3] J. Dukes, J. Jones. Using Dynamic Replication to Manage Service Availability in a Multimedia Server Cluster. In *Proc. of MIPS* 2004, 194–205.

[4] R. Gandhi, M. M. Halldórsson, G. Kortsarz and H. Shachnai, Improved results for data migration and open shop scheduling". *ACM Transactions on Algorithms*, 2(3): 116–129, 2006.

[5] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proc. of SODA*, 223–232, 2000.

[6] C. Griwodz, M. Bar, L. C. Wolf, Long-term movie popularity models in Video-on-demand Systems: Or the life of an On-demand movie. *Proc. of ACM Multimedia*, 1997, 349–357.

[7] X. Guo, J. Li, J. Yang and J. Wang, The Research on Dynamic Replication and Placement of File Using Dual-Threshold Dynamic File Migration Algorithm. In *Proc. of CSSE* (3), 2008, 236–240.

[8] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, Dynamic application placement for clustered web applications. In *Proc. of WWW*, 2006.

[9] S. Kashyap, Algorithms for data placement, reconfiguration and monitoring in storage networks. Ph.D. Dissertation. Computer Science Department, Univ. of Maryland, 2007.

[10] S. Kashyap and S. Khuller, Algorithms for non-uniform size data placement on parallel disks. *FST & TCS*, 2003.

[11] S. Kashyap, S. Khuller, Y-C. Wan and L. Golubchik. Fast reconfiguration of data placement in parallel disks. *ALENEX*, 2006.

[12] S. Khuller, Y. Kim, and Y. C. Wan. Algorithms for Data Migration with Cloning. *In Proc. of the 22nd ACM Symposium on Principles of Database Systems*, pages 27–36, 2003.

[13] Y. Kim. Data Migration to Minimize the Average Completion Time. *Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms,* pages 97–98, 2003.

[14] P.W.K. Lie, J.C.S. Lui, and L. Golubchik. Threshold-based dynamic replication in large-scale video-on-demand systems. In *Proc. of the Eighth International Workshop on Research Issues in Database Engineering (RIDE)*, 52–59, 1998.

[15] L. Lovász and M.D. Plummer. *Matching Theory*, American Mathematical Society, 2009.

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

[17] H. Shachnai and T. Tamir, Polynomial time approximation schemes for class-constrained packing problems. *J. of Scheduling*, vol. 4(6), pp. 313–338, 2001.

[18] H. Shachnai and T. Tamir, Approximation Schemes for Generalized 2-dimensional Vector Packing with Application to Data Placement. In *Proc. of APPROX*, 2003.

[19] D. S. Shmoys and E. Tardos, Scheduling unrelated machines with Costs. *Proc. of SODA*, 1993.

[20] J.L. Wolf, P.S. Yu, and H. Shachnai. Disk load balancing for video-on-demand systems. *ACM Multimedia Systems J.*, 5:358–370, 1997.

[21] H. Yu, D. Zheng, B.Y. Zhao, W. Zheng. Understanding user behavior in large scale video-on-demand systems. *The 1st ACM SIGOPS/EuroSys European Conf. on Comp. Systems*, 2006.

[22] H. Peyton Young. Equity in theory and practice. Princeton University Press, 1995.

[23] X. Zhou and C.Z. Xu, Optimal video replication and placement on a cluster of video-on-demand servers. Proc. of the *International Conference on Parallel Processing* (ICPP), 2002.