Minimal Cost Reconfiguration of Data Placement in 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 some small $0 < \delta < 1$ when the number of servers is fixed, and by factor of $2+2\varepsilon$ for arbitrary number of servers, for some $0 < \varepsilon < 1$. To the best of our knowledge, this fundamental optimization problem is studied here for the first time.

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, 20]), the distribution of accesses to movie files is highly skewed; indeed, only small fraction of the movies are requested 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, $1 \le i \le M$, is associated with a popularity parameter $p_i^0 \in (0,1]$, where $\sum_{i=1}^{M} p_i^0 = 1$. Each server j, $1 \le j \le N$, 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

*Department of Computer Science, The Technion, Haifa 32000, Israel. E-mail: hadas@cs.technion.ac.il.
†Department of Computer Science, The 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

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

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 to 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.

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 $\sum_{i,j} c_{i,j}$. Note that deletion of a movie file is free. 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

[2] The broadcast demands are assumed to be integers. Rounded values can be obtained by standard solutions for the apportionment problem [21].

[3] In VoD system design, this is also known as the *static phase* [19].

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.

we assume that the system is *semi-homogeneous*, i.e., all servers have the same load capacities and 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 $c_{1,2} + c_{5,1}$.

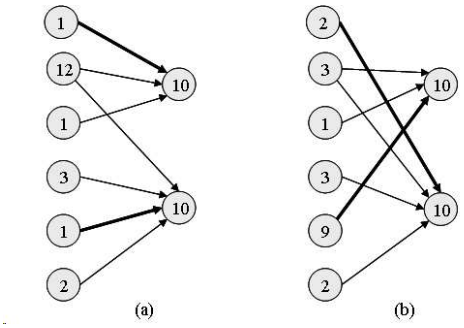

Figure 1: A perfect placement before (a) and after (b) the popularity change. Bold edges represent changes in storage assignment.

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 [16]. Suppose that 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. Assuming that the new popularity vector has a perfect placement,[4] 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 some 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 (see Section 3.1). 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 $2 + 2\varepsilon$, for some $0 < \varepsilon < 1$. Due to space constraints some of the proofs are given in the Appendix.

1.2 Related Work

The data placement problem has been extensively studied (see, e.g., [19, 5, 10, 17, 8] and a comprehensive survey in [9]). The paper [15] 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 [15] 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, 17]).

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]). A survey of known results for the data migration problem is given in [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, 22, 3, 7]). We are not aware

[4] This is often the case in practical scenarios, where the new popularity vector is a *permutation* of the initial vector, that is, the popularity distribution function remains unchanged.

of earlier results for our reconfiguration problem.

2 Hardness Results

We show that the reconfiguration problem is NP-hard even if the system consists of only two servers, and even if the 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* $X = \{x_1, x_2, \dots x_{n_X}\}$ *and* $Y = \{y_1, y_2, \dots, y_{n_Y}\}$*, 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 *such that* $S_X = S_{Y''} + z$*. 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.*

We are now ready to prove the hardness of the reconfiguration problem.

Theorem 2.2 *The reconfiguration problem is NP-hard even if the system consists of only two servers, all replication costs are uniform, and even if the popularity changes are limited 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. Given X, Y, z, such that there exists a subset $Y'' \subseteq Y$ of size n_X such that $S_X = S_{Y''} + z$, 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. $C_1 = n_X,\ L_1 = S_X - z,\ C_2 = n_Y,\ L_2 = S_Y + z$. 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 exactly fulfill the broadcast demands. Assume further that the demands are changed to be the values of the elements in X, Y. Specifically, $D = \langle x_1, x_2, \dots x_{n_X}, y_1, y_2, \dots, y_{n_Y} \rangle$. 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 represents 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.

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

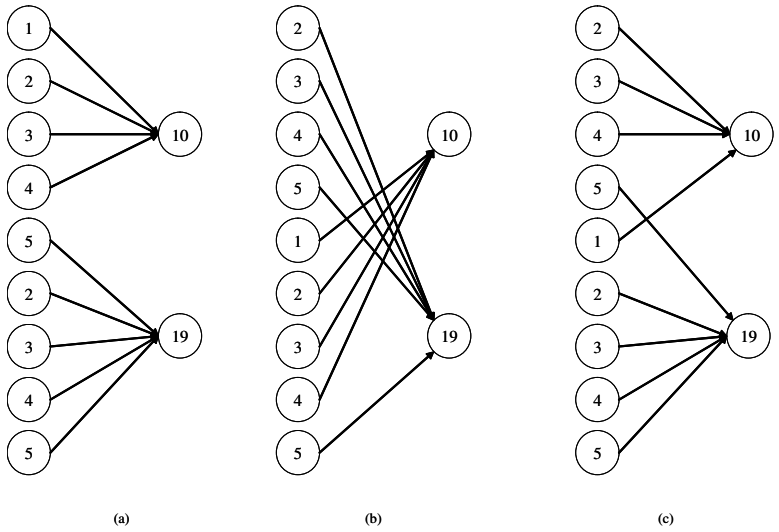

Figure 2: The reconfiguration problem induced by Example 2. (a) initial assignment, (b) reconfiguration having cost 8, and (c) an optimal reconfiguration having cost 2.

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 solving the reconfiguration problem can be used to solve the *smallest subsets with a given difference* problem, implying that the reconfiguration problem is NP-hard. □

3 Minimal Cost Algorithm for Fixed Number of Servers

We present a polynomial time algorithm which finds a minimal-cost reconfiguration for a semi-homogeneous system, assuming that the number of servers, N, is some fixed constant. The algorithm outputs a placement which achieves the optimal cost and uses servers with load capacities $(1 + 3\delta)L$, for a small parameter $\delta \in (0, 1]$.

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

(P_1) For every big movie i, and every server j, the broadcast allocation $B_{i,j}$ of server j to movie i is either 0 or at least δL and an integral multiple of $\delta^2 L$, i.e., $B_{i,j} = k\delta^2 L$, where k is an integer in $[1/\delta, 1/\delta^2]$.

(P_2) Every 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 $j' \neq j$, $B_{i,j'} = 0$.

We show below that if a slight augmentation of the load capacities is allowed, then a minimal

Reconfiguration Algorithm

For each storage and load allocation to the big movies, fulfilling 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 $L_j := L - L_j^b + 2\delta$ 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.

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

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

cost reconfiguration fulfilling the above properties can be found in polynomial time. In addition, limiting the set of reconfigurations to ones fulfilling the properties does not affect the minimal cost. An overview of the algorithm is given in Figure 3.

3.1 Analysis of the Algorithm

In analyzing the algorithm, we use the next technical lemmas.

Lemma 3.1 *Limiting the allocation to a one fulfilling* (P_1) *and* (P_2) *may require an increase by at most a factor of* $(1+2\delta)$ *in the servers load capacity, without changing the configuration cost.*

Proof: Given an arbitrary storage allocation, in which the total broadcast allocation of each server is at most L. We show that it can be converted into an allocation fulfilling the properties. The only changes are in the broadcast matrix, B. The assignment matrix A remains with unchanged, therefore, the reconfiguration cost is the same.

Consider the servers one after the other, for each server j and big movie, i if $B_{i,j} \geq \delta L$, then round $B_{i,j}$ up to the next multiple of $\delta^2 L$. Note that there are at most $1/\delta$ such movies, thus, the total overflow on the server due to this step is at most δL. If $0 < B_{i,j} < \delta L$ and the total demand already allocated to movie i on previously considered servers is less than D_i then set $B_{i,j} = \delta L$. Similarly, for the small movies, when server j is considered, if for some small movie i, $0 < B_{i,j} \leq D_i$ and movie i was not assigned its total demand on a previously considered server, then set $B_{i,j} = D_i$. Indeed, the total overflow in the server due to this step might be much larger than δL, because the allowed overflow (in this stage) is per big movie and not per server.

Following the above step, the assignment matrix A remains the same, and both properties P_1 and P_2 hold. However, some servers have a big overflow, of more than $2\delta L$, some are assigned total load capacity less than L, and the rest are assigned total load capacity between L and $(1+2\delta)L$ - with an allowed overflow. Denote the above sets of servers *Overflowed, Vacant,* and *Fair* respectively. Consider a graph whose vertices set is the servers and there is a clique for each movie. That is, two servers j_1, j_2 are connected if for some movie i, $A_{i,j_1} = A_{i,j_2} = 1$. A *balancing step* is defined by a path in this graph. The starting server is overflowed, the middle servers are fine, and the last server is vacant. Server j_2 can follow server j_1 in the path if j_1 is overflowed of fine, j_2 is fine or vacant, and for at least one movie i, shared by the servers, some amount of broadcasts can be migrated from j_1 to j_2. If the edge exists due to a small movie i then D_i broadcasts can be migrated. If it exists due to a big movie, then some amount between δL and $2\delta L$ broadcasts can be migrated. Specifically, the allocation satisfies that $B_{i,j_1} \geq \delta L$. If $B_{i,j_1} = \delta L$ or $B_{i,j_1} \geq 2\delta L$, then exactly δL broadcasts are transmitted. If $B_{i,j_1} \leq 2\delta L$ then B_{i,j_1} broadcasts are migrated. This ensures that the resulting assignment also satisfies (P_1). Balancing steps are performed as long as one exists. Note that we do not care about the time complexity of detecting

and performing balancing steps, as we only need to prove the *existence* of an assignment fulfilling P_1 and P_2. The following claim completes the proof.

Claim 3.2 *If no balancing step exists, then all servers are allocating at most* $(1+2\delta)L$ *broadcasts.*

□

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.*

Lemma 3.4 *Given a configuration of the big movies on the servers, there exists a polynomial time algorithm which finds for the small movies a placement of minimum cost, fulfilling property* (P_2)*, where each server* j *has storage capacity* C_j *and load capacity* $L_j(1+\delta)$*.*

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 assignment will be determined by rounding the solution to the following linear program. The costs $c_{i,j}$ are the given replication costs. Note that in the input for the assignment problem, once the big movies have been assigned to the servers, the servers may have different load capacities; however, by scaling the broadcast requirements of the movies, we may assume, w.l.o.g., that the load capacities of the servers satisfy $L_1 = \cdots = L_N = \hat{L}$. Specifically, for a movie i and server j, define $D_{i,j} = D_i \cdot \hat{L}/L_j$.

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

Claim 3.5 *Given an optimal solution for LP, with the cost* C*, there exists a polynomial time algorithm which finds an assignment of the small movies to servers of load capacity* $\hat{L}(1+\delta)$*, whose total cost is at most* C*.*

Proof: Given an optimal (fractional) solution for LP, we use a rounding technique of Shmoys and Tardos [18]. 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. We show below that the resulting integral solution may use servers whose load capacities are at most $\hat{L}(1+\delta)$.

Formally, sort the small movies in non-decreasing order by load requirements. 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 $v_{i,1}, \ldots, v_{i,\sigma_j}$ correspond to 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 $x_{i,j} \geq 0$, 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 $w(u_i, v_{j,\sigma_j}) = x_{i,j}$.

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 $v_{j,k}$. We note that the weight function on the edges of G_B defines a fractional matching, in which any movie vertex U_i is exactly matched to a server vertex $v_{j,k}$, $1 \leq k \leq \sigma_j - 1$. In other words, for any $1 \leq j \leq N$ and $1 \leq k \leq \sigma_j - 1$, $\sum_{i=1}^{M_r} w(u_i, v_{j,k}) = 1$. In addition, for all $1 \leq j \leq N$ and $1 \leq k \leq \sigma_j - 1$,

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

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

1. Given an optimal solution for LP, 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 $\hat{L}(1+\delta)$. Since we defined in G_B a fractional matching of cost C, there exists in G_B an integral matching, H, whose cost is C, such that all the movie vertices are exactly matched. Therefore, the 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 at most the optimal cost C.

Next, we show that the total broadcast requirement assigned on each server is at most $L(1+\delta)$. 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

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

The second inequality follows from (1) and the fact that $D_{i,j} \leq \delta \hat{L}$ for all $1 \leq i \leq M_r$. □

This completes the proof of the lemma. □

Combining the above lemmas, we summarize in the next result.

Theorem 3.6 *Given a system of N servers, each 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 optimal, by using servers with load capacities $L(1+3\delta)$.*

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.*

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 $(2+2\varepsilon)L$ where

$$\varepsilon = \text{argmax}_{\{i|D_i<L\}} D_i/L. \tag{2}$$

4.1 The Algorithm

The following is an overview of the algorithm. (*i*) Partition the instance to *big* and *small* movies; movie i is big if $D_i \geq L$, else movie i is small. For any big movie i let $k_i = \lceil D_i/L \rceil$. Define the instance $\tilde{I}$ in which the broadcast demand of any big movie i is $\tilde{D}_i = k_i L$, and the broadcast demand of any small movie i is $\tilde{D}_i = D_i$. (*ii*) Solve a linear programming relaxation of the reconfiguration problem for $\tilde{I}$ with the constraints that each small movie is assigned a single copy and each big movie i is allocated the load capacity $\tilde{D}_i$. (*iii*) Round the (fractional) solution of the linear program to obtain an integral solution whose cost is optimal. (*iv*) Use the integral solution to assign movie copies to N servers, where server j has storage capacity C_j and load capacity $(2+2\varepsilon)L$.

Solving the LP: Let $x_{ij} \in \{0,1\}$ be an indicator for storing big movie i on server j (on which it is allocated L broadcasts). Also, $y_{ij} \in \{0,1\}$ is an indicator for the assignment of small movie i on server j. Finally, $c_{i,j}$ is the given replication cost (which depends on the initial configuration). We first find an optimal solution for LP', the linear programming relaxation of the problem, in which, $x_{ij} \in [0,1]$ denotes the fraction of the load capacity of server j allocated to movie i.

Constraints (3) ensure that the total load capacity used by copies of the big movies and by the small movies on each server is at most $(2+\varepsilon)L$.[5] Constraints (4) ensure that the total storage required on server j is at most C_j. The constraints (5) and (6) reflect the bounds on the number of copies of each (big or small) movie. Together with constraints (3), it is guaranteed that each movie is allocated $\tilde{D}_i$ broadcasts.

Rounding the Fractional Solution: To obtain an integral solution from a given optimal solution for LP', replace first the variables $x_{i,j}$, $1 \leq j \leq N$, $i \in Big$ by the following set of variables. For each variable $x_{i,j}$, let $f_i \geq 1$ be the smallest integer such that $\tilde{D}_i/f_i \leq \max_{i|D_i<L} D_i$. We define $k'_i = f \cdot k_i$ variables $x_{i,j,r}$, $1 \leq r \leq k'_i$, where $x_{i,j,r} = x_{i,j}/k'_i$. Intuitively, we partition

[5] We refer to this constraint in Lemma 4.3.

$$
\begin{aligned}
\textit{(LP')}: \quad \text{minimize} \quad & \sum_{i\in Big}\sum_{j=1}^{N} x_{i,j}\cdot c_{i,j} + \sum_{i\in Small}\sum_{j=1}^{N} y_{i,j}\cdot c_{i,j} & & \\
\text{subject to:} \quad & \sum_{i\in Big} x_{i,j}\cdot L + \sum_{i\in Small} y_{i,j}\cdot D_i \le (2+\varepsilon)L & \text{for } 1\le j\le N & (3)\\
& \sum_{i\in Big} x_{i,j} + \sum_{i\in Small} y_{i,j} \le C_j & \text{for } 1\le j\le N & (4)\\
& \sum_{j=1}^{N} x_{i,j} = k_i & \text{for } i\in Big & (5)\\
& \sum_{j=1}^{N} y_{i,j} = 1 & \text{for } i\in Small & (6)\\
& 0\le x_{i,j}\le 1 & \text{for } 1\le j\le N,\quad i\in Big & \\
& 0\le y_{i,j}\le 1 & \text{for } 1\le j\le N,\quad i\in Small &
\end{aligned}
$$

the load requirement of movie i to k'_i, so we can now consider k'_i sub-movies, where each needs to be allocated $D^S_{max} = \max_{i|D_i<L} D_i$ broadcasts and can have a single copy. Note that the values of the variables $x_{i,j,r}$ and $y_{i,j}$ form an optimal solution for the linear program LP".

Thus, it is possible to apply to the variables $y_{i,j}$, $i \in Small$, $1 \le j \le N$ and $x_{i,j,r}$, $i \in Big$, $1 \le j \le N$ and $1 \le r \le k'_i$ the rounding technique of Shmoys and Tardos [18], as described in the proof of Claim 3.5 (by taking $\delta = \varepsilon$). Use the resulting integral solution to assign a single copy of small movie 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, let $\sum_{r=1}^{k'_i} x_{i,j,r} = h_{i,j}$, $0 \le h_{i,j} \le \min(C_j, 3)$ be the number of copies of the sub-movies of i assigned to server j. If $h_{i,j} > 0$ then assign a copy of big movie i on server j, i.e., $A_{i,j} = 1$ and $B_{i,j} = h_{i,j}L$.

4.2 Analysis

Let $OPT(I)$ denote the cost of some optimal solution for I.

Lemma 4.2 *The constraint that any small movie is assigned a single copy may increase the total load capacity used on any server at most by εL.*

The proof is similar to the proof of Lemma 3.1 (with $\delta = \varepsilon$). We omit the details.

Lemma 4.3 *There exists a solution for $\tilde{I}$ where each small movie is assigned a single copy on servers with load capacities $(2+\varepsilon)L$, such that the total cost is $OPT(I)$.*

Proof: By Lemma 4.2, the requirement that each small movie is assigned a single copy may add at most εL to the broadcast requirement of server j, $1 \le j \le N$. Also, rounding up the broadcast requirement of the big movies increases the requirement of each movie at most by factor of 2. Thus, given a solution for I, we can obtain a valid solution for $\tilde{I}$ by doubling the load allocated to any copy of big movie i on server j. This may increase the total load capacity used on server j at most by L. Thus, we get an overall demand of at most $(2+\varepsilon)L$ on each server. The running time of the algorithms is polynomial, since it involves solving and rounding the solution of an LP program with a polynomial number of variable and constraints. □

Theorem 4.4 *The above algorithm outputs in polynomial time a solution of cost at most $OPT(I)$.*

$$
\begin{aligned}
\textit{(LP")}: \quad \text{minimize} \quad & \sum_{i\in Big}\sum_{r=1}^{k'_i}\sum_{j=1}^{N} x_{i,j,r}\cdot c_{i,j} + \sum_{i\in Small}\sum_{j=1}^{N} y_{i,j}\cdot c_{i,j} \\
\text{subject to:} \quad & \sum_{i\in Big}\sum_{r=1}^{k'_i} x_{i,j,r}\cdot D^S_{max} + \sum_{i\in Small} y_{i,j} D_i \le (2+\varepsilon)L \qquad 1\le j\le N \\
& \sum_{i\in Big}\sum_{r=1}^{k'_i} x_{i,j,r} + \sum_{i\in Small} y_{i,j} \le C_j \qquad 1\le j\le N \\
& \sum_{j=1}^{N} x_{i,j,r} = 1 \qquad \text{for} \quad i\in Big \\
& \sum_{j=1}^{N} y_{i,j} = 1 \qquad \text{for} \quad i\in Small \\
& x_{i,j,r}\ge 0 \qquad \text{for} \;\; 1\le j\le N, i\in Big, \;\; 1\le r\le k'_i \\
& y_{i,j}\ge 0 \qquad \text{for} \;\; 1\le j\le N, \;\; i\in Small
\end{aligned}
$$

The movies can be placed on N servers with storage capacities $C_1,\ldots,C_N$ and load capacities $(2+2\varepsilon)L$, where ε is defined in (2).

Proof: By Lemma 4.3, the cost of an optimal solution for LP' is at most $OPT(I)$. As shown in the proof of Claim 3.5, 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 in the solution output by the algorithm. By Lemma 4.3 we get an increase of at most $(2+\varepsilon)L$ in the load capacity of the servers due to rounding up the load requirements of the big movies, and since we assign a single copy to any small movies. Finally, rounding the fractional solution may increase the used load capacity on each server by εL. This results in a total increase of $(1+2\varepsilon)L$ in the servers load capacities. □

Extension to Heterogeneous severs: We show that with slight change, the above algorithm can be applied to a system where each server has arbitrary load capacity. Let $L_{min} = \min_{1\le j\le N} L_j$ and $L_{max} = \max_{1\le j\le N} L_j$ be the minimal and maximal load capacities of any server, respectively. We assume below that the ratio L_{max}/L_{min} is *bounded* by some polynomial in the input size. Consider the above algorithm with the following change. We say that movie i is *big* if $D_i \ge L_{min}$ and *small* otherwise. Also, define $k_i = \lceil D_i/L_{min}\rceil$. Round up the broadcast demand of any big movie i to $\tilde{D}_i = k_i L_{min}$. Let $D^S_max < L_{min}$ be the maximal broadcast demand of any small movie. As before, we define k'_i such that for any big movie i $\tilde{D}_i \le k'_i D^S_{max}$. Next, solve LP' and define for any big movie i $x_{i,j,r} = x_{i,j}/k'_i$. Thus, we replace each big movie i by k'_i sub-movies whose broadcast demands are D^S_{max}. For these movies, as well as for the small movies, a single copy needs to be assigned. Using an analysis similar to the analysis in Section 4.2, we get the following result.[6]

Theorem 4.5 *There is a polynomial time algorithm which finds a min-cost reconfiguration for general bounded-ratio instances, using servers whose load capacities are increased at most by factor of $2+2\varepsilon$, for some $\varepsilon\in(0,1)$.*

[6]We give the detailed proof in the full version of the paper.

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*, 2003.

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

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

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

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

[19] 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.

[20] 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 Conference on Comp. Systems*, 2006.

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

[22] X. Zhou and C.Z. Xu, Optimal video replication and placement on a cluster of video-on-demand servers. Proc. of the *2002 International Conference on Parallel Processing* (ICPP'02), pp. 547–555.

A Appendix - Some Proofs

Proof of Lemma 2.1: 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 such that $S_X = S_{Y''} + z$, 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 our assumptions (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 A.1 *The set X has a subset of k elements having total size w if and only if there is a positive answer to the above decision problem of* smallest subsets with a given difference.

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 if and only if $S_{X'} = w = k + z = S_{Y'} + z$. that is, if and only if 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 $S_X - n_X + k = S_{Y'} + z$. □

□

Proof of Claim 3.2: Let $M(j)$ be the set of movies that are allocated broadcasts from server j, that is, $i \in M(j)$ if and only if $B_{i,j} > 0$. Assume that no balancing step exists but there exists an overflowing server j. Since the allocation fulfills (P_1) and (P_2), then for every $i \in M(j)$, if i is big then $B_{i,j} \geq \delta L$, and if i is small then $B_{i,j} = D_i$. Clearly, j can migrate broadcasts to any of his neighbors in the cliques corresponding to the movies in $M(j)$. After such a migration is done, any server j' in each such clique can similarly migrate broadcasts to any server in the cliques corresponding to movies in $M(j) \cup M(j')$. By repeating this process, we calculate the set of all servers that might be allocated additional broadcasts due to a balancing path starting at server j. Note that this set of servers forms a connected component in the graph, which is servicing some set of movies. If all the servers in this component are overflowed or fine then the total broadcast allocation to movies stored on this component is larger than their allocation in the given assignment - a contradiction. Otherwise, there must be a vacant server, v, in the component and by the way the component was defined, there is a balancing path from j to v.

A single balancing step might involve migrations of up to $2\delta L$ broadcasts, as well as migrations of small amounts of broadcasts (D_i for some small movie i). Therefore, a fine server might become overflowed by participating in a balancing path. However, this new overflowed server will be taken care of by another balancing step. The process must end with no overflowed server since, as shown

above, it is guaranteed that as long as there is an overflowed server, there is also a vacant server that can get additional broadcasts. □

Proof of Lemma 3.3: 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 $1/\delta$ positive entries in the corresponding column, each has a value $k\delta^2 L$, for an integer $k \in [1/\delta, 1/\delta^2]$. Thus, there are at most $(M \cdot 1/\delta^2)^{1/\delta}$ possible ways to fill each column in B, and $(M/\delta^2)^{N/\delta}$ possible ways to determine the allocation to big movies. This value is polynomial since N is fixed. □

Proof of Theorem 4.1: 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. □