On Data Movement and Migration*

Samir Khuller † Yoo-Ah Kim ‡ Yung-Chun (Justin) Wan §

Abstract

Our work was motivated by the problem of managing data on storage devices, typically a set of parallel disks. Due to dynamic changes in load, one needs to develop algorithms for quickly changing one storage configuration into another. We refer to this problem as the data migration problem.

1 Introduction

To handle high demand, especially for multimedia data, a common approach is to replicate data objects within the storage system. Typically, a large storage server consists of several disks connected using a dedicated network, called a *Storage Area Network*. Disks typically have constraints on storage as well as the number of clients that can access data from a single disk simultaneously.

Approximation algorithms have been developed [25, 26, 12, 19] to map known demand for data to a specific data layout pattern to maximize utilization[1]. In the layout, we not only compute how many copies of each item we need, but also a layout pattern that specifies the precise subset of items on each disk. The problem is NP-hard, but there are polynomial time approximation schemes [12, 26]. Hence given the relative demand for data, the algorithm computes an almost optimal layout. Recently, in [19] we have been able to extend the algorithms even when the data objects do not have the same size.

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

A special case of this was studied in [14] - here they compute a movement schedule, but this *does not allow* the creation of new copies of any data object. It addresses only the data *movement* problem. (So for example, we cannot create extra copies of any data item, we can just change which disks they are stored on.) The problem studied in [14] is formally defined as follows: given a set of disks, with each storing a subset of items and a specified set of move operations (each move operation specifies which data object needs to be moved from one disk to another), how do we schedule these move operations? If there are no storage constraints, then this is exactly the problem of edge-coloring the following multi-graph. We create a graph that has a node corresponding to each disk, and a directed edge corresponding to each move operation that is specified. Algorithms for edge-coloring multigraphs can now be applied to produce a migration schedule since each color class represents a matching in the graph that can be scheduled simultaneously. With space constraints on the disk, the problem becomes challenging. In [14] it is shown that with the assumption that each disk has one spare unit of storage, very good constant factor approximations can be developed. The algorithms use at most $4\lceil\Delta/4\rceil$ colors with at

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. This research was supported by NSF Awards CCR-9820965 and CCR-0113192.

†E-mail : samir@cs.umd.edu.

‡E-mail : ykim@cs.umd.edu.

§E-mail : ycwan@cs.umd.edu.

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

most $n/3$ bypass nodes, or at most $6\lceil\Delta/4\rceil$ colors without bypass nodes [2].

On the other hand, to handle high demand for popular objects, new copies will have to be dynamically created and stored on different disks. This means that we crucially need the ability to have a "copy" operation in addition to "move" operations. In fact, one of the crucial lower bounds used in the work on data migration [14] is based on a degree property of the multi-graph. For example, if the degree of a node is Δ, then this is a lower bound on the number of rounds that are required, since in each round at most one transfer operation involving this node may be done. For copying operations, clearly this lower bound is not valid. For example, suppose we have a single copy of a data item on a disk. Suppose we wish to create Δ copies of this data item on Δ distinct disks. Using the transfer graph approach, we could specify a "copy" operation from the source disk to each of the Δ disks. Notice that this would take at least Δ rounds. However, by using newly created copies as additional sources we can create Δ copies in $\lceil\log(\Delta + 1)\rceil$ rounds, as in the classic problem of broadcasting by using newly created copies as sources for the data object. **undefined term as yet**

EDIT

Each data item that initially belongs to a subset of disks, needs to be moved to another subset of disks. (We might need to create new copies of this data item and store it on an additional set of disks.) If each disk had exactly one data item, and needs to copy this data item to every other disk, then it is exactly the problem of gossiping. Different communication models can be considered based on how the disks are connected. We use the same model as in the work by [14, 1] where the disks may communicate on any matching; in other words, the underlying communication graph is complete.

The *most general problem* of interest is the **data migration problem** when data item i resides in a specified (source) subset S_i of disks, and needs to be moved to a (destination) subset D_i. We show that this problem is NP-hard and develop a polynomial time 12-approximation algorithm for it. We have also implemented the algorithm, and compare its performance to several other heuristics for the problem.

For all our algorithms, we move data only to disks that need the data. Thus we use no bypass nodes.

1.1 Relationship to Gossiping and Broadcasting

The problems of Gossiping and Broadcasting have been the subject of extensive study [20, 16, 17, 3, 4, 18, 9, 11]. These play an important role in the design of communication protocols in various kinds of networks. The *gossip problem* is defined as follows: there are n individuals. Each individual has an item of gossip that they wish to communicate to everyone else. Communication is typically done in rounds, where in each round an individual may communicate with at most one other individual. There are different models that allow for the full exchange of all items of gossip known to each individual in a single round, or allow the sending of only one item of gossip from one to the other (half-duplex) or allow each individual to send an item to the individual they are communicating with in this round (full-duplex). In addition, there may be a communication graph whose edges indicate which pairs of individuals are allowed to communicate in each round. (In the classic gossip problem, also called the telephone model, communication may take place between any pair of individuals; in other words, the communication graph is the complete graph.) In the *broadcast problem*, one individual needs to convey an item of gossip to every other individual. The two parameters typically used to evaluate the algorithms for this problem are: the number of communication rounds, and the total number of telephone calls placed.

The problems we study are generalizations of the above mentioned gossiping and broadcasting problems. The basic generalizations we are interested in are of two kinds (a) each item of gossip needs to be communicated to only a subset of individuals, and (b) several items of gossip may be known to an individual.

The communication model we use is the half-duplex model, where only one item of gossip may

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

be communicated between two communicating individuals during a single round. Each individual may communicate (either send or receive an item of data) with at most one other individual in a round. *This model best captures the connection of parallel storage devices that are connected on a network and is most appropriate for our application.* *** We should also add references for this model as well as relationship of this model with full duplex ***

1.2 Contributions and Outline of Paper

In Section 2 we define the basic model of communication and the notation used in the paper.

This is the first work that relates broadcasting and gossiping with the data migration problem. In Section 4 we develop a 12 approximation algorithm for the general data migration problem. The data migration problem will be shown to be NP-hard in the Appendix.

1.3 Related Work

The paper by Liben-Nowell [22] considers a problem very similar to multi-source multicast. However, the model that he uses is different than the one that we use. In his model, in each telephone call, a pair of users can exchange all the items of gossip that they know. The objective is to simply minimize the total number of phone calls required to convey item i of gossip to set D_i of users. In our case, since each item of gossip is a data item that might take considerable time to transfer between two disks, we cannot assume that an arbitrary number of data items can be exchanged in a single round. Several other papers use the same telephone call model [2, 7, 15, 18, 30].

Other related problems that have been studied are the set-to-set gossiping problem [21, 24] where we are given two possibly intersecting sets A and B of gossipers and the goal is to minimize the number of calls required to inform all gossipers in A of all the gossip known to members in B. Liben-Nowell [22] generalizes this work by defining for each gossiper i the set of relevant gossip that they need to learn. This is just like our multi-source multicast problem with $\Delta = N$, except that the communication model is different, as well as the objective function.

2 Models and Definitions

In the *data migration problem*, we have N disks and Δ data items. For each item i, there is a subset of disks S_i and D_i. Initially only the disks in S_i have item i, and all disks in D_i want to receive i. Note that after a disk in D_i receives item i, it can be a source of item i for other disks in D_i that have not received the item as yet. Our goal is to find a migration schedule using the minimum number of rounds, that is, to minimize the total amount of time to finish the schedule. We assume that the underlying network is fully connected and the data items are all the same size, in other words, it takes the same amount of time to migrate an item from one disk to another. The crucial constraint is that each disk can participate in the transfer of only one item - either as a sender or receiver. Moreover, as we do not use any bypass nodes, all data is only sent to disks that desire it.

Our algorithms make use of known results on edge coloring of multi-graphs. Given a graph with max degree Δ and multiplicity μ the following results are known (see [6] for example). Let χ' be the edge chromatic number of G.

THEOREM 2.1. *(Vizing [31]) If G has no self-loops then $\chi' \leq \Delta + \mu$.*

THEOREM 2.2. *(Shannon [27]) If G has no self-loops then $\chi' \leq \frac{3\Delta}{2}$.*

3 The Data Migration Problem

The algorithm is based on ideas developed in the previous section. However dealing with multiple data items sharing common disks causes some difficulty. Moreover, if many copies of an item are available initially, our lower bound of $\max_i \lceil \log |D_i| \rceil$ does not hold any more. Hence we cannot afford to simply broadcast item i to the group G_i, as this may take $\log |G_i|$ steps. This makes Step 2 somewhat complicated.

We first define β as $\max_{i=1\ldots N} |\{j | i \in D_j\}|$. In other words, β is an upper bound on the number of different sets D_j, that a disk i may belong to. Note that β is a lower bound on the optimal number of rounds, since the disk i that attains the max, needs at least β rounds to receive all the items j such that $i \in D_j$, since it can receive at most one item in each round.

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

Algorithm Data Migration.

1. For an item i such that $|S_i| > 1$, decide a unique source $s_i \in S_i$ so that $\alpha = \max_{j=1,\ldots,N} |\{i|j = s_i\}|$ is minimized. In other words, α is the maximum number of items that a disk may be a source for. Note that α is also a lower bound on the optimal number of rounds. In Lemma 3.1 will show how we can find a source for each item.

2. Find a transfer graph for items such that $|D_i| \geq \beta$.

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

   (b) We have each item i sent to the set G_i as shown in Lemma 3.4.

   (c) We now create a transfer graph as follows. Each disk is a node in the graph. We add a directed edge from each disk in G_i to $(\beta-1)\lfloor\frac{|D_i|}{\beta}\rfloor$ disks in $D_i \backslash G_i$ such that the out-degree of each node in G_i is at most β and the in-degree of each node in $D_i \setminus G_i$ is 1. Note that s_i is not involved since we satisfy only $(\beta - 1)\lfloor\frac{|D_i|}{\beta}\rfloor$ disks in $D_i \setminus G_i$ in this step. We redefine D_i as a set of the remaining $|D_i \setminus G_i| - (\beta - 1)\lfloor\frac{|D_i|}{\beta}\rfloor (< \beta)$ disks so that they can be taken care of in Step 3.

3. Find a transfer graph for items such that $|D_i| < \beta$.

   (a) For each item i, find a new source in D_i. A disk j can be a source for several items as long as $\sum_{i \in I_j} |D_i| \leq 2\beta$ where I_j is a set of items of which j is a new source. See Lemma 3.5 for the details of this step.

   (b) Send each item from s_i to its new source.

   (c) Create a transfer graph. We add a directed edge from the new source of item i to all disks D_i. The out-degree of a disk will not exceed 2β since $\sum_{i \in I_j} |D_i| \leq 2\beta$.

4. We now find an edge coloring of the transfer graph obtained by merging two transfer graphs in Step 2(c) and 3(c). The number of colors used is an upper bound on the number of rounds required to ensure that each disk in D_j gets item j. In Lemma 3.7 we derive an upper bound on the number of required colors.

LEMMA 3.1. *We can find a source $s_i \in S_i$ for each item so that $\max_{j=1,\ldots,N} |\{i|j = s_i\}|$ is minimized, using a flow network.*

Proof. We create a flow network with a source s and a sink t. We have two set of nodes corresponding to disks and items. Add directed edges from s to nodes for items and also directed edges from item i to disk j if $j \in S_i$. The capacities of all those edges are one. Finally we add edges from nodes for disks to t and want to find the minimum capacity α of these edges for the maximum flow of the network to be Δ. We can do this by checking if there is a flow of Δ with the capacity starting from one and increasing by one until it is satisfied. If there is outgoing flow from item i to disk j, then we set j as s_i.

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

LEMMA 3.2. *If m' be the smallest number such that we can construct a solution of $F_{m'}$ that satisfies all demands $|G_i|$, then $M \geq m'$.*

Proof. Suppose that $M < m'$. Given a optimal schedule of Step 2(b), we can construct a solution of the flow network F as follows. If a disk j sends item i to a disk in G_i at round $t \leq M$ and it makes f copies in G_i subsequently, we send a flow f from x_{jt} to y_i. Note that f cannot be more than 2^{M-t} and therefore, it does not violate the capacity constraint. Since all disks in G_i receive item i after M rounds with this schedule, the corresponding flow satisfies all demands $|G_i|$. This is a contradiction to the assumption that m' is the smallest number to satisfy all demands $|G_i|$.

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

LEMMA 3.3. *Given a solution of $F_{m'}$, we can convert it to a solution satisfying the following properties.*

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

Proof. First, we define a variable z_{ijk} for an edge from x_{jk} to y_i and set $z_{ijk} = f_{ijk}/c_{ijk}$ where f_{ijk} is the flow through e_{ijk}. For each node y_i in V, we create nodes $y_{il}(l = 1 \ldots \lfloor \sum_{j,k} z_{ijk} \rfloor)$. We distribute edges having nonzero flow to y_i as follows. Sort edges in non-increasing order of their capacities. Assign edges to y_{i1} until the sum of z values of assigned edges is greater than or equal to one. If the sum is greater than one, we split the last edge (denote as $e_{ij'k'}$) into $e_{ij'k'_1}$ and $e_{ij'k'_2}$. Assign $e_{ij'k'_1}$ to y_{i1} and define $z_{ij'k'_1}$ so that the sum of z values of edges assigned to y_{i1} is exactly one. Set $z_{ij'k'_2} = z_{ij'k'} - z_{ij'k'_1}$. We repeat this so that for all nodes y_{il}, the sums of z values of the assigned edges are one.

In the resulting bipartite graph with U and $V' = \{y_{il}\}$, z makes a *fractional* matching which matches all vertices in V'. Therefore, we can find an integral matching that matches all vertices in V' and the matching satisfies the first property in the lemma. Now we merge nodes y_{il} into y_i. Then each y_i matches exactly $\lfloor \sum_{j,k} z_{ijk} \rfloor$ edges. Using an analysis similar to that in [28], we can show that the sum of capacities of edges matched to y_i is at least $|G_i| - c_{max}$ where c_{max} is the maximum capacity of edges. Since $c_{max} = 2^{m'-1}$, the second property is satisfied.

LEMMA 3.4. *Step 2(b) can be done in $\alpha + 2m' + O(1)$ rounds.*

Proof. We can do this with the following schedule. First we choose a disk $s'_i \in G_i$ arbitrarily. It takes $\alpha + 3$ rounds to send item i from s_i to s'_i since a disk can have outdegree α and indegree one and also the multiplicity is 2 (see Theorem 2.1).

Next we choose $\lfloor \sum_{j,k} z_{ijk} \rfloor$ disks in $G_i - s'_i$ and disk j sends item i to one of these disks if edge e_{ijk} is matched for some k. It is easy to see that it takes at most $m' + 2$ rounds.

Finally since $\lfloor \sum_{j,k} z_{ijk} \rfloor + 1$ nodes in G_i have item i and $|G_i|/(\lfloor \sum_{j,k} z_{ijk} \rfloor + 1) \leq 2^{m'-1}$, we can make all disks in G_i have item i in additional m' rounds.

The following lemma will show how Step 3(a) works.

LEMMA 3.5. *For each item i we wish to choose a source disk from D_i. Let I_j be the set of items for which disk j is a source. There is a way to choose the sources such that the following properties hold*

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

Proof. We use the result by Shmoys and Tardos [28] for this step. For example, we can create an instance of the problem of scheduling machines with costs. Items correspond to jobs and disks correspond to machines. For each item i we define a cost function as follows. $C(i,j) = 1$ if and only if $j \in D_i$, otherwise it is a large constant C. Processing time of job i (corresponding to item i) is $|D_i|$ (uniform processing time on all machines). The scheduling algorithm in [28] finds a schedule that assigns each job (item) to a machine (disk) to minimize the makespan. They show that the makespan is at most the makespan in a fractional solution plus the processing time of the largest job. Moreover, the cost of their solution is at most the cost of the optimal solution, namely the number of items. We cannot assign an item (job) to a disk (machine) if the disk is not in the destination set for the item.

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

LEMMA 3.6. *Step 3(b) can be done in $\frac{3}{2}(\alpha+\beta)+O(1)$ rounds.*

Proof. Since j can be a new source for item i in Step 3(a) only if $j \in D_i$, $|I_j| \leq \beta$. Therefore, a disk j may need to send α items, and receive β items. That means the maximum degree is $\alpha+\beta$ and this transfer can make a multi-graph. Given a multi-graph with maximum degree δ, we can find an edge coloring using $\lceil\frac{3}{2}\delta\rceil$ colors (see Theorem 2.2 [27]) and the lemma follows.

LEMMA 3.7. *The number of colors we need for the final transfer graph in Step 4 is $\frac{3}{2}\cdot 4\beta+O(1)$.*

Proof. The out-degree of a disk j can be at most $3\beta-1$ ($\beta-1$ in Step 2 and 2β in Step 3). The in-degree is at most β by definition.

THEOREM 3.1. *The total number of rounds required for the data migration is at most $\alpha+2m'+\frac{3}{2}(\alpha+5\beta)+O(1)$.*

COROLLARY 3.1. *Our algorithm is 12-approximation for the data migration problem.*

4 Heuristics

4.1 Edge-Coloring on Transfer Graph

We can find a transfer schedule using edge coloring on a transfer graph. Since a disk can get the same item from different source disks, we want to find a good way of selecting source disks. We build a flow network with a source s and a sink t. In addition we have two sets of nodes corresponding to items and source disks. We add edges from s to node i with capacity $|D_i|$ for all item i, and edges from source disks j to t with capacity $c-\beta_j$. Finally, we add edges from item i to source disk j if $j \in S_i$.

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

Now we find an edge coloring of the transfer graph (which may be a multigraph) to obtain a valid schedule [6], and the number of colors used is an upper bound on the total number of rounds.

We claim that the approximation ratio of this method is $\Omega(\frac{\Delta|D_i|}{\Delta+\log|D_i|})$. Consider the problem single source multicast, all items are initially stored at a single disk s, D_i's equally divide all remaining disks. As there is only one source, the out-degree of the source disk equals to the total demand, which is $\Delta|D_i|$, in the transfer graph. This heuristic takes at most $\frac{3}{2}\Delta|D_i|$ rounds. The optimal solution is for the source to send item i to a disk in D_i at i-th round, and in parallel doubling item i among disks in D_i thereafter. This would take $O(\Delta+\log|D_i|)$. *** Or use single source broadcast? ***

4.2 Broadcast Items One by One

In this algorithm, we deal with one item at a time. We process each item i sequentially, and satisfy the demands by doubling the item in each round, which takes $\log(\lceil\frac{D_i}{S_i}\rceil+1)$. The total number of rounds is $\sum_i \log(\lceil\frac{D_i}{S_i}\rceil+1)$.

We claim that the approximation ratio of this method is $\Omega(\Delta)$. Consider the case where $|S_i|$ is 1 and $|D_i|$ is 3 for all i, and all S_i and D_i are disjoint. The optimal solution is to double all items in parallel, which takes 2 rounds. The algorithm takes 2 rounds for each item, and the total is 2Δ.

4.3 Heuristics Using Matching

The algorithm is as follows.

1. Build a weighted undirected graph as follows.

Each disk is a node in the graph. Let the cost $c(i,v,w)$ of sending item i from disk $v \in S_i$ to disk $w \in D_i$ be $\frac{|D_i|}{|S_i|}$. Intuitively, if disk v and w are matched, we would like to send the item with greatest cost, and either v or w can be the sender. The weight between v and w in the graph is $\max(\max_i c(i,v,w), \max_i c(i,w,v))$.

2. Find a maximum-weight matching on this graph. For each matched v and w, suppose $c(i,v,w)$ is the weight between v and w, send item i from v to w. This takes 1 round to send items in the matched pairs. We then update the S_i's and D_i's.

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

Worse case example. *** try to find a non-constant worse case example. *** *** other weight functions ***

References

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

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

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

[4] J. Bermond, L. Gargano, A. A. Rescigno and U. Vaccaro. Fast gossiping by short messages. *International Colloquium on Automata, Languages and Programming*, 1995.

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

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

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

[8] G. J. Chang and Y.J. Tsay. The Partial Gossiping Problem. *Discrete Mathematics*, 148:9–14, 1996.

[9] E. J. Cockayne, A. G. Thomason. Optimal Multi-message Broadcasting in Complete Graphs. *Utilitas Mathematica*, 18:181–199, 1980. 62

[10] Y. Dinitz, N. Garg and M. Goemans. On the single-source unsplittable flow problem, *Proc. of FOCS*, 1998.

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

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

[13] M. Grigni and D. Peleg. Tight bounds on minimum broadcast networks. *SIAM J. on Discrete Math.*, 4(2):207–222, 1991.

[14] J. Hall, J. Hartline, A. Karlin, J. Saia and J. Wilkes. On Algorithms for Efficient Data Migration. *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, 2001.

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

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

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

[18] C. A. J. Hurkens. Spreading Gossip Efficiently. *Nieuw Archief voor Wiskunde*, 5/1(2):208–210, 2000.

[19] S. Kashyap and S. Khuller. Algorithms for Non-Uniform Size Data Placement on Parallel Disks. *Manuscript*, 2002.

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

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

[22] D. Liben-Nowell. Gossip is Synteny: Incomplete Gossip and an Exact Algorithm for Syntenic Distance. *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[23] C. H. Papadimitriou. Computational complexity. *Addison-Wesley*, 1994.

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

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

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

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

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

[29] M. Stonebraker. A Case for Shared Nothing. *Database Engineering*, 9(1), 1986.

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

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

Appendix: NP-hardness

We will prove the multi-source multicasting problem is NP-hard by showing a reduction from a restricted version of 3SAT. Papadimitriou [23] showed that 3SAT remains NP-complete even for expressions in which each variable is restricted to appear at most three times, and each literal at most twice. We denote this problem as 3SAT(3).

We assume that each literal appear at least once in the given instance. If not, we can always simplify it such that all literal appear at least once (or the instance is always true).

Given a 3SAT(3) instance, we create a multi-source multicast instance such that the 3SAT(3) instance is satisfied if and only if the corresponding multi-source multicast instance can transfer all items in 3 rounds.

Part I. For each variable x_i, we create (i) a source disk having item x_i, (ii) a set of destination disks X_i of size 3 which need item x_i, (iii) a source disk having item $\overline{x}_i$, (iv) a set of destination disks $\overline{X}_i$ of size 3 which need item $\overline{x}_i$, (v) a source disk having item s_i, (vi) a disk w_i (we call it a switch disk) which wants to receive items x_i, $\overline{x}_i$ and s_i, and (vii) 6 disks which need item s_i.

Part II. For each clause j, we create (i) a source disk having item c_j, and (ii) a set of destination disks C_j of size 2 which need item c_j. Moreover, for each literal in the clause j, randomly pick one disk in the set of destination disks corresponding to the literal, and that disk, which originally need an item corresponding to the literal, will also need item c_j. For example, if clause j is $x_p \vee \overline{x}_q \vee x_r$, then one disk in X_p, one disk in $\overline{X}_q$ and one disk in X_r need item c_j.

LEMMA 4.1. *If the 3SAT(3) instance is satisfiable, there exists a valid schedule to finish all data transfers in* 3 *rounds.*

Proof. It is easy to see that all seven disks demanding item s_i can be scheduled in three rounds. In particular, we schedule switch disk w_i to receive s_i in round 3 for all i. If variable x_i is **true**, we schedule switch disk w_i to receive $\overline{x}_i$ and x_i in round 1 and 2 respectively. x_i can be sent to a disk in X_i in round 1, making X_i receive items faster than $\overline{X}_i$. After round 2, 2 disks in X_i received item x_i, while only 1 disk in $\overline{X}_i$ received item $\overline{x}_i$. In round 3, the source disk of x_i can satisfy the last disk in X_i which has not received x_i. Note that the remaining 2 disks in X_i are idle and they can receive item c_j from other disks. On the other hand, the remaining 2 disks in $\overline{X}_i$ can be satisfied in round 3 by the source and one disk in $\overline{X}_i$. Note that all disks in $\overline{X}_i$ and the source of $\overline{x}_i$ are busy in this round. Thus, all requested items appeared in **Part I** are satisfied. If the variable is **false**, we schedule the switch disk to receive x_i in round 1, then $\overline{x}_i$ in round 2. As a result, 2 disks in $\overline{X}_i$ are idle in round 3, while all disks in X_i are busy in round 3.

We claim that all 2 disks in C_j, for all j, can be satisfied too. For example, if clause j is $x_p \vee \overline{x}_q \vee x_r$, and suppose x_p is **true** while $\overline{x}_q$ and x_r are **false** in a satisfactory assignment. From the argument above, there exists a schedule such that the disk in X_p, which needs x_p and c_j in X_p, is idle in round 3. However, the disk in $\overline{X}_q$, which needs $\overline{x}_q$ and c_j, and the disk in X_r, which needs x_r and c_j, are busy in round 3. A valid schedule can send item c_j from the source to one disk in C_j in round 1. In round 2, we now have 2 copies of c_j to satisfy disks in $\overline{X}_q$ and X_r. In round 3, without the help of disks in $\overline{X}_q$ and X_r, we can satisfy 2 more disks, which are the second disk in C_j and the disk in X_p. Thus, all requested items appeared in **Part II** are satisfied too.

LEMMA 4.2. *If there is a valid schedule to finish* *all data transfers in 3 rounds, then the 3SAT(3) instance is satisfiable.*

Proof. Since there are 7 disks need item s_i, if we have to finish all transfers in 3 rounds, once a disk receives s_i, it will become busy until round 3. Note that all switch disks have to receive s_i, x_i and $\overline{x}_i$. All switch disk have to receive item x_i and $\overline{x}_i$ in the first two rounds, and s_i in the round 3. If switch disk i receives item x_i in round 1, we set literal $\overline{x}_i$ to be **true**. Otherwise, we set literal x_i to be **true**. Consider the former case, disks in X_i receive item x_i starting at round 2, meaning that all disks in X_i should be busy in round 3 to send or receive x_i. Suppose literal x_i appears in clause j and k. 2 disks in X_i have to receive item c_j and c_k in the first 2 rounds. Thus, our construction restricts that if a literal x_i is set to **false**, disks in X_i cannot receive item c_j in round 3.

Consider a clause j, for instance, $x_p \vee \overline{x}_q \vee x_r$, a disk in X_p, a disk in $\overline{X}_q$, a disk in X_r, and all 2 disks in C_j need item c_j. If all three literals are **false**, it is possible to satisfy the first three disks in the first 2 rounds. However, since all these three disks are busy in round 3, the source of c_j cannot satisfy both disks in C_j, which is a contradiction. Therefore, the clause j should be satisfied.

THEOREM 4.1. *The multi-source multicasting problem is NP-hard*

Proof. It is easy to see that the reduction is polynomial, together with Lemma 4.1 and Lemma 4.2, the problem is NP-hard.