Data Migration in Large Scale Heterogeneous Storage Systems with Nodes to Spare

Chadi Kari*, Sixia Chen†, Sepehr Amir-Mohammadian*, and Vivek Pallipuram*,
*School of Engineering and Computer Science, University of the Pacific
†Department of Computer Science, Central Connecticut State University

***Abstract*—In large scale storage systems such as data centers, the layout of data on storage disks needs to be frequently reconfigured for load balancing purposes or in the event of system failure/upgrades. This reconfiguration event, referred to as *data migration*, must be completed efficiently as the system tends to perform sub-optimally during such process. The data-migration problem has been studied extensively in the literature with efficient algorithms presented for homogeneous (all storage disks have similar capabilities) and heterogeneous (storage disks can have different capabilities) cases.**

In this paper, we investigate adding data forwarding to existing algorithms for the heterogeneous data migration problem. In data forwarding, we introduce additional storage nodes (called bypass nodes) during the migration process. Our simulations show that adding as few as 2 bypass nodes with limited capabilities can improve the performance by up to 15% and adding more bypass nodes with heterogeneous capabilities can improve the migration performance by 25%. We then present a novel algorithm that makes intrinsic use of bypass nodes and show that the algorithm can always achieve an optimal migration schedule while adding no more than $\alpha \times n/3$ bypass nodes where n is the numbers of disks and α is a term defined to reflect the heterogeneity factor of disks.

I. Introduction

The amount of data being generated in today's data-intensive applications is growing at a staggering pace. The role of big data is now critical for several important fields such as healthcare [1], genomics [2], and smart cities [3], among others. To leverage superior data processing capabilities, modern cloud infrastructures [1,2] work tirelessly to ensure that the end users seamlessly access and operate upon their data. As pointed by [4] and [5], one of most important big data challenges is to manage the voluminous information in large scale storage systems or data centers.

The performance of such systems depends critically on the data assignment to storage devices such that the load is balanced across all storage devices and/or a cost function is optimized . This assignment of data to storage devices is called a *data layout*. The optimal data layout needs to be reconfigured over time due to disk failures, disk additions, load balancing or changing usage patterns. Once the new data layout is computed, data must be migrated to its new assigned location. It is critical that the migration is done as quickly as possible to minimize power consumption and because the system performs sub-optimally during the migration process.

[1]https://aws.amazon.com/big-data/
[2]https://cloud.google.com/

We can model the input of the data-migration problem using a transfer graph $G = (V, E)$. Each vertex in the graph represents a storage disk and each edge (u, v) represents a data object to be moved from disk u to disk v. There can be multiple data objects that need to be moved between u and v and therefore graph G can be a multigraph (see Figure 1).

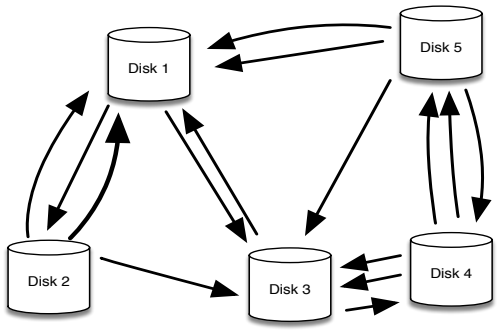

Fig. 1. An example of data transfer instance

We associate a constraint c_v with each node $v \in G$ that represents the number of simultaneous transfers that a disk can handle. A data-migration is complete when all the data objects are moved, i.e., when all the edges in the graph are scheduled. For example, consider the transfer graph in Figure 2. If a disk can participate in one transfer at a time then the data-migration process will take nine rounds of transfers (one disk is always idling) to complete. However, if the disks are able to perform two transfers at a time, then migration can be completed in three rounds.

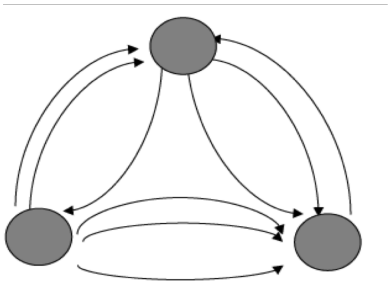

Fig. 2. Transfer Graph with 3 nodes

We assume that any pair of disks can send data to each other directly (i.e. the underlying storage system network

is complete) and all data objects are of the same size (for example, fragment variable-sized objects into unit sized pieces. The time it takes to send unit-sized pieces is the same as the time it takes to send the entire object). Since we consider objects of the same fixed size, we can assume that a migration plan consists of a series of *rounds*. Each round corresponds of a number of compatible transfers.

Most of the previous research has focused on *homogeneous* storage systems where all disks have the same capabilities and it is assumed that a disk can perform one data transfer at a time [6], [7], [8], [9], [10]. If a disk can be involved in only one transfer (either sending or receiving but not both), then the data-migration problem is precisely the multigraph edge coloring problem and therefore it is NP-complete. A legal edge coloring of the graph gives a migration plan for our problem (all edges labeled with the same color can be scheduled in the same round without any conflict).

As storage disks are added overtime to the system, they tend to have different capabilities (for example different read/write speeds, rpms, seek times etc...). In [11], [12], the authors consider the *heterogeneous* data-migration problem, where disks have heterogeneous capabilities and some disks are able to handle multiple simultaneous transfers.

In this paper, we consider the heterogeneous data-migration problem with data forwarding. In data forwarding, data exchanged between disk u and disk v is not necessarily delivered directly from u to v but can be forwarded to other disks that are idling or to additional disks that are added for migration process. We refer to such disks as *bypass nodes* or *bypass disks*. The following example shows that adding bypass nodes could reduce the number of steps for migration. In the left transfer graph, all the nodes have transfer constraint equal to 1 and each edge is duplicated k times. The number of steps needed to complete the migration is $3k$. By adding one bypass node, d, the number of steps can be reduced to $2k$.

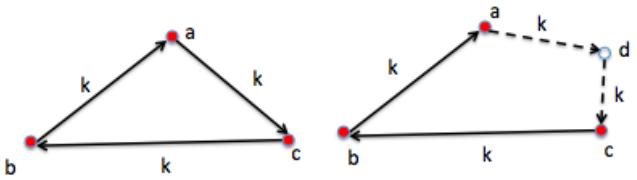

Fig. 3. Performance can be improved by adding a bypass node

We investigate adding data forwarding and bypass nodes to 2 existing algorithms from [11]. Our simulations show that adding a small number of bypass nodes can significantly improve the performance of the data-migration algorithms. We then present a novel algorithm, *Flatten-and-Factor*, that makes intrinsic use of bypass nodes and show that the algorithm can achieve an optimal migration schedule while adding no more than $\alpha \times n/3$ bypass nodes where n is the number of disks and $\alpha = \sum_{v \in G} c_v/(n \times c_{bypass})$ is a term defined to reflect the heterogeneity factor of disks.

The rest of the paper is organized as follows. Section II reviews the current literature. Section III describes the algorithms for data-migration with the addition of data forwarding. The section also presents and analyzes a novel migration algorithm called *Flatten-and-Factor*. Section IV describes the experimental setup, describes the results and gives an analysis and discussion of these results. The paper concludes in Section V with a summary and directions for future research.

II. Related Work

A. Homogeneous Data Migration

Most variants of the data-migration problem are NP-complete. Previous research has focused on the homogeneous data-migration problem with the constraint that each disk can participate in only one migration at a time. Coffman et al. [13] developed optimal algorithms for several special cases such as cycles, trees, and bipartite graphs. Hall et al. [6] studied the homogeneous data-migration problem and developed efficient approximation algorithms.

Coffman et al. [13] and Sanders et al. [14] studied the problem of data-migration with forwarding. Whitehead [15] shows that when forwarding is needed because some of the edges in the transfer graph are not present in the interconnection network, then the problem is NP-complete.

The data-migration problem with forwarding becomes more challenging when constraints on disk space are assumed. Hall et al. [6] developed approximation algorithms with constant factor approximations under the assumption that each disk has one spare unit of space. Anderson et al. [16] performed an experimental study of several algorithms including the ones in [6]. They show that the algorithms in question perform much better in practice than the theoretical bounds suggest.

B. Heterogeneous Data Migration

In the heterogeneous data-migration problem, disks have different capabilities and each disk v has a transfer constraint c_v, which represents the number of simultaneous transfers that the disk can handle. Saia [17] presented a 1.5-approximation algorithm for arbitrary c_v. The algorithm finds a solution by creating c_v copies of each node and distributing the edges incident to the node evenly. The maximum degree of the node now becomes $\lceil d_v/c_v \rceil$ where d_v is the degree of a node v. Shannon's theorem [18] then gives an edge coloring algorithm with at most $1.5\lceil d_v/c_v \rceil$ colors. Kari et al. [11] presented an optimal algorithm in the case where all the disk constraints are even and an efficient approximation algorithm for the general heterogeneous data migration problem with an approximation factor of $1+o(1)$. Roberts et al. [12] empirically evaluated the performance of several algorithms on multiple transfer graphs of varying sizes and described which algorithms are best suited for certain characteristics of transfer graph instances.

III. Problem Definition and Algorithms

Our model makes the same assumptions as in [6] and [12]. In the Heterogeneous Data-Migration problem, we assume the network topology is fully connected and each disk may send data directly to any other disk. A directed multigraph $G = (V, E)$ is given. Each edge $e = (i, j) \in E$ represents a data

item that needs to be transferred from disk i to disk j. It is assumed that each data item needs the same amount of time to be migrated. Each disk v has a transfer constraint c_v, which represents the number of transfers in which the disk can engage concurrently. We use d_v to denote the degree of v.

Additionally, we are allowed to use bypass nodes during migration. A data item can be forwarded to a bypass node first and transferred to the destination later. Our objective is to minimize the number of rounds to finish the entire data-migration without adding too many bypass nodes.

A. Algorithms

The task of an algorithm for the HETEROGENEOUS DATA-MIGRATION problem is to compute a schedule, in which each edge of the given graph is assigned to some time step, or round. All the data items that are selected in a round can be sent simultaneously.

We revisit two algorithms from [12]: Edge-Ranking and Randomized-Algorithm that address the heterogeneous data-migration problem without using any bypass nodes. These two algorithms use different strategies to select a maximal set of non-conflicting edges to transfer data items in each round. We modify the original algorithms by adding a fixed number of bypass nodes at the beginning of the algorithm and applying two extra steps in each round.

For the sake of self-containment, we include the algorithms from [12].

Algorithm 1 Edge-Ranking

1: For every edge $e = (u,v) \in E$, define its weight to be $d_u/c_u + d_v/c_v$.
2: Sort the edges in descending order of their weights.
3: Select edges in order from the sorted list until a maximal set of nonconflicting edges has been selected.
4: Allocate the selected set of edges to the current time step.
5: Delete the edges that are selected from the list and repeat from step 3 for the following time step if any unallocated edges remain.

Algorithm 2 Randomized Algorithm

1: Randomly order the edges in E.
2: For each $e \in E$, select e if it is nonconflicting with all the edges selected for the current time step.
3: Allocate the selected set of edges to the current time step.
4: Delete the edges that are selected from the list and repeat from step 2 for the following time step if any unallocated edges remain.

To the above two algorithms, we add the following additional steps: At the end of each round, we choose a maximal set of data items that remain from this round to forward to the available bypass nodes. At the beginning of the next round, we choose data items to send from bypass nodes to destination nodes as long as the destinations are available and delete the corresponding edges from the multigraph. The algorithms can be seen in detail below.

The following pseudocode can be adapted to modified versions of Edge-Ranking and Randomized-Algorithm by changing the predefined order in step 2. For Edge-Ranking, for every edge $e = (u,v) \in E$, define its weight to be $d_u/c_u + d_v/c_v$ and the queue is created in descending order of weight of the edges. For modified Randomized-Algorithm, the order is random.

Algorithm 3 Modified Algorithms

1: Let $B = \emptyset$
2: Create a queue, Q, of all edges in a predefined order.
3: Allocate a maximal set of edges from B to the current time step. For each edge allocated in this way, delete the corresponding edge from Q.
4: Select and remove edges from Q until a maximal set of nonconflicting edges has been selected. Allocate the selected set of edges to the current time step.
5: Select (but do not remove) a maximal set of edges from Q to redirect to bypass nodes and, for each such edge (u,v), allocate to the current time step a new edge (u,b) to the corresponding bypass node and add to B a new edge (b,v).
6: Repeat from step 3 if any edges remain in Q.

In the next section, we empirically compare the results of the above two algorithms. Each algorithm is simulated in four ways: the number of bypass nodes is chosen to be either 2 or random, and the transfer capacities of the bypass nodes are either identical or varied.

B. A Third Algorithm — Flatten-and-Factor

We present a novel algorithm called Flatten-and-Factor, which makes use of the concept of *graph factorization*. Specifically, it makes use of well-known algorithms that exist to compute a 2-factor of a graph, which is a set of cycles spanning the entire graph. Our algorithm

1) transforms the transfer graph by expanding each node into a set of nodes with cardinality equal to the original node's capacity (each new node has transfer constraint one),
2) computes a 2-factor decomposition of the new graph, and then
3) adds one bypass node to each odd cycle.

An example of G and the corresponding G' are given in Figure 4.

Analysis of this algorithm yields the following result.

Theorem 1. *If all the bypass nodes have transfer constraint c_{bypass}, Flatten-and-Factor requires at most $2\max_v \lceil d_v/2c_v \rceil$ steps and uses at most $\alpha \times n/3$ bypass nodes on any multigraph G, where $\alpha = \sum_v c_v/(n \times c_{bypass})$.*

Proof. For each 2-factor of the graph, the algorithm uses two step2. So in total the algorithm takes $2\max_v \lceil d_v/2c_v \rceil$ steps to complete. The bypass nodes are only added for the odd cycles, so their number will not exceed $|V'|/3 = \sum_v c_v/3$ since

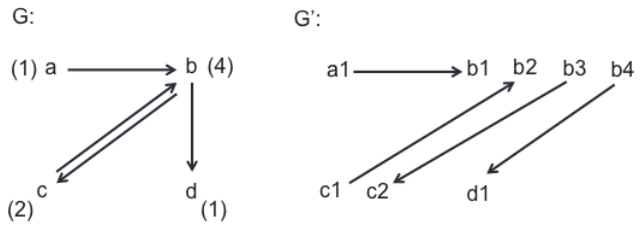

Fig. 4. The graph G is a transfer graph with transfer constrains given in parenthesis. The graph G' is constructed by the Flatten-and-Factor algorithm

Algorithm 4 Flatten-and-Factor

1: For the given graph $G = (V,E)$, define a new graph $G' = (V',E')$ as follows:
   - for every node $v \in V$, create $v_1, v_2, \cdots, v_{c_v}$ in V';
   - for every edge $e = (u,v) \in E$, connect u_i and v_j for some i and j such that for every node $v \in V$ and $v_i \in V'$, $\lceil d(v)/c(v) \rceil \geq d_{v_i} \geq \lceil d_v/c_v \rceil - 1$.
2: Add dummy self-loops and edges to G to make it regular and even degree $(2\max_v \lceil d_v/2c_v \rceil)$.
3: Compute a 2-factoring of G'. This gives $\max_v \lceil d_v/2c_v \rceil$ 2-factors.
4: In each 2-factor, add one bypass node and two edges in each odd cycle so that all cycles are even. Allocate every other edge in each cycle to the first time step and the remaining edges to the second.

the worst case is there are only odd cycles and each one has three nodes. But note that the bypass nodes here all have transfer constraint 1. If the bypass nodes have transfer constraint at least c_{bypass}, then the same bypass node can be used in c_{bypass} cycles. So the number of bypass nodes will not exceed $\sum_v c_v/3c_{bypass}$. □

The number of rounds matches the lower bound of the optimal solution for the heterogeneous data migration problem without using bypass nodes, which is proved to be $max_v \lceil d_v/c_v \rceil$ in [11].

IV. Simulation Results and Analysis

For our experiments, we adapt the simulator developed in [12] to execute Randomized and Edge-Ranking algorithms with and without bypass nodes as shown in Figure 5. The goal of the simulator is to simulate data-migration in a disk farm using the scheduling algorithms described in Section III. After completing the data-migration process for all of the disks, the simulator outputs the number of rounds required to complete the process. In what follows, we describe the operation of the simulator using its four stages. The simulator inputs the graph attribute $((V,E))$, disk attribute (c_v), and a user input, *schedule_select*, to select a scheduling algorithm (Edge-Ranking or Randomized with or without bypass nodes) for data-migration. The disk farm generator stage constructs a model of a disk farm using the graph and disk attributes. In the first round of simulation, the schedule iterator stage inputs the original model of the disk farm and checks if there are any disks available for scheduling. If such disks exist, the schedule iterator increments the *round* counter by one and passes the disk farm to the scheduler stage. The scheduler stage uses the selected scheduling algorithm to mark the scheduled disks and passes the marked disk farm to the data-migration stage. The data-migration stage removes the marked disks from the disk farm (thereby simulating data-migration) and passes the updated disk farm to the schedule iterator stage for the next iteration. When all of the disks are scheduled, the schedule iterator stage terminates and returns the number of scheduling rounds.

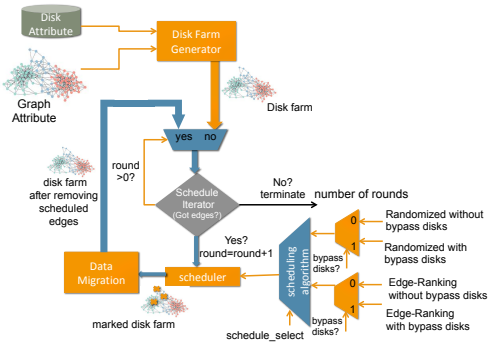

Fig. 5. Data-migration simulator with its four primary stages: disk farm generator, schedule iterator, scheduler, and data-migration.

We consider transfer graphs with c_v equal to 2. Figure 6.(a) provides an analysis of the number of rounds taken by the Randomized algorithm without the bypass disks (highlighted by blue squares) and with two bypass disks (highlighted by red, right facing triangles). As seen in the same figure, adding two bypass disks results each with $c_v = 1$ reduces the number of rounds across the graph sizes (up to 15% for some instances). For a very few cases, the addition of two bypass disks does not reduce the number of rounds, but the performance is never worse than the case without any bypass disks. Figure 6.(b) shows a similar trend for the Edge-Ranking algorithm, where an improvement up to 14% is shown.

Figures 6.(c) and (d) provide the results for the two migration algorithms when the number of bypass disks is allowed to exceed 2. As seen in these figures, adding more than 2 bypass disks results in significantly less number of rounds versus the case where no bypass disks are considered (upwards of 25% for some instances). The performance benefit is particularly perceived for graphs with larger number of disks, highlighting the strengths and scalability of our algorithms.

V. Conclusion

In this paper, we have explored adding data forwarding to existing algorithms for the heterogeneous data migration problem. Our results show that adding a few bypass nodes even with limited capabilities can have a significant improvement

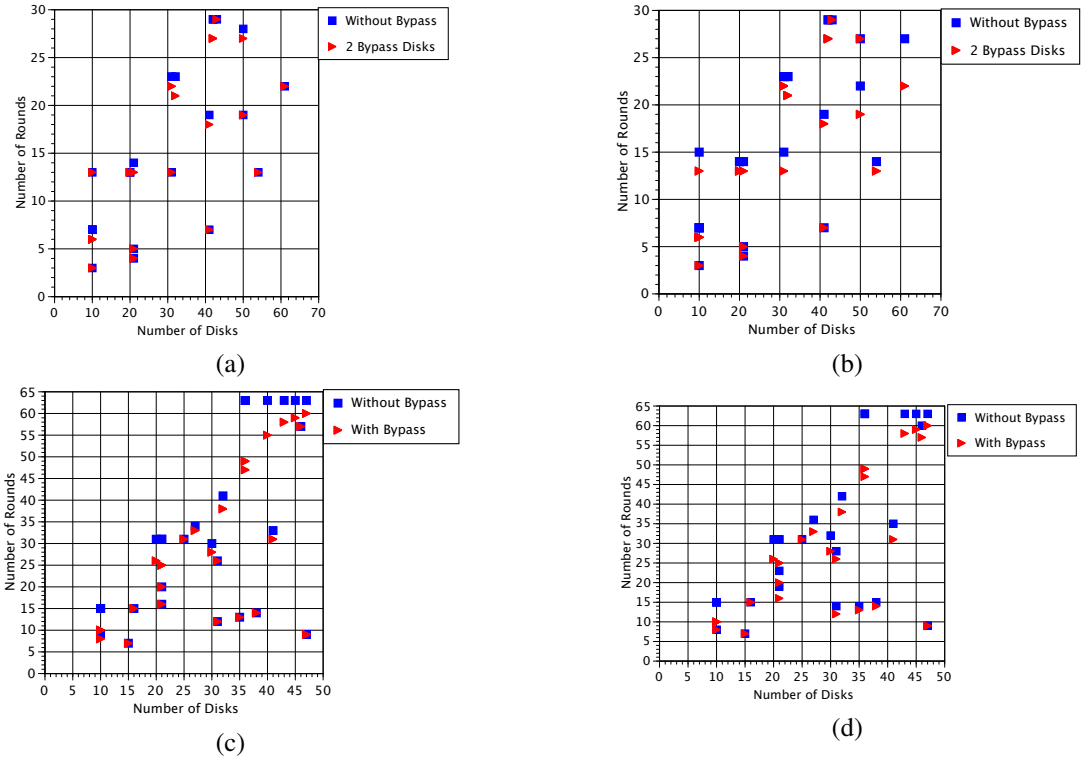

Fig. 6. Comparison of the number of rounds taken by Randomized and Edge-Ranking algorithms with and without bypass disks: (a) Randomized; bypass disks=2;c_v=2, (b) edge-ranking; bypass disks=2; c_v=2, (c) randomized; random bypass disks; c_v=2, (d) edge-ranking; random bypass disks; c_v=2

on the performance of the data migration algorithms. We also presented and analyzed Flatten-and-Factor, an algorithm that makes intrinsic use of bypass nodes and have provable performance guarantees.

For future work, we would like to explore adding the use of bypass nodes to other existing algorithms in [11], [12] and empirically compare their performance to Flatten-and-Factor. Prior research on data migration algorithms [12], [16] suggests that these algorithms perform much better in practice than their theoretical bounds and it would be interesting to explore if this remains true when adding forwarding. Another interesting and challenging future direction is to study the heterogeneous data migration problem with forwarding when constraints on disk space is considered.

ACKNOWLEDGMENT

The authors gratefully acknowledge use of the services, support and research credits provided by Google Cloud Platform.

REFERENCES

[1] M.-T. Kechadi, "Healthcare big data: Challenges and opportunities," in *Proceedings of the International Conference on Big Data and Advanced Wireless Technologies*, BDAW '16, (New York, NY, USA), pp. 3:1–3:1, ACM, 2016.

[2] K.-C. Wong, *Big Data Analytics in Genomics*. Springer, 2016.

[3] Y. Sun, H. Song, A. J. Jara, and R. Bie, "Internet of things and big data analytics for smart and connected communities," *IEEE Access*, vol. 4, pp. 766–773, 2016.

[4] Z. Lv, H. Song, P. Basanta-Val, A. Steed, and M. Jo, "Next-generation big data analytics: State of the art, challenges, and future research topics," *IEEE Transactions on Industrial Informatics*, vol. 13, pp. 1891–1899, Aug 2017.

[5] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Commun. ACM*, vol. 57, pp. 86–94, July 2014.

[6] J. Hall, J. Hartline, A. Karlin, J. Saia, and J. Wilkes, "On algorithms for efficient data migration," in *SODA*, pp. 620–629, 2001.

[7] S. Khuller, Y. Kim, and Y.-C. Wan, "Algorithms for data migration with cloning," in *22nd ACM Symposium on Principles of Database Systems (PODS)*, pp. 27–36, 2003.

[8] L. Golubchik, S. Khuller, Y. Kim, S. Shargorodskaya, and Y. J. Wan, "Data migration on parallel disks," in *12th Annual European Symposium on Algorithms (ESA)*, 2004.

[9] S. Khuller, Y.-A. Kim, and Y.-C. J. Wan, "On generalized gossiping and broadcasting," in *12th Annual European Symposium on Algorithms*, 2004.

[10] S. Khuller, Y. Kim, and A. Malekian, "Improved algorithms for data migration," in *APPROX*, 2006.

[11] C. Kari, Y. Kim, and A. Russell, "Data migration in heterogeneous storage systems," in *31st International Conference on Distributed Computing Systems*, pp. 153–160, 2011.

[12] G. Robert, S. Chen, C. Kari, and V. Pallipuram, "Data migration algorithms in heterogeneous storage systems: A comparative performance evaluation," in *16th IEEE International Symposium on Network Computing and Applications*, pp. 105–108, 2017.

[13] E. Coffman, M. G. Jr., D. Johnson, and A. Lapaugh, "Scheduling file transfers," *SIAM J. Computing*, vol. 14, no. 3, pp. 744–780, 1985.

[14] P. Sanders and R. Solis-Oba, "How helpers hasten h-relations," *Journal of Algorithms*, vol. 41, pp. 86–98, 2001.

[15] J. Whitehead, "The complexity of file transfer scheduling with forwarding," *SIAM J. Comput.*, vol. 19, no. 2, pp. 222–245, 1990.

[16] E. Anderson, J. Hall, J. Hartline, M. Hobbs, A. R. Karlin, J. Saia, R. Swaminathan, and J. Wilkes, "An experimental study of data migration algorithms," in *International Workshop on Algorithm Engineering*, pp. 145–158, Springer, 2001.

[17] J. Saia, "Data migration with edge capacities and machine speeds," tech. rep., University of Washington, 2001.

[18] C. Shannon, "A theorem on colouring lines of a network," *J. Math. Phys.*, vol. 28, pp. 148–151, 1949.