Data Migration in Heterogeneous Storage Systems

Chadi Kari
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269
Email: chadi@engr.uconn.edu

Yoo-Ah Kim
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269
Email: ykim@engr.uconn.edu

Alexander Russell
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269
Email: acr@engr.uconn.edu

***Abstract*—Large-scale storage systems are crucial components in data-intensive applications such as search engine clusters, video-on-demand servers, sensor networks and grid computing. A storage server typically consists of a set of storage devices. In such systems, data layouts may need to be reconfigured over time for load balancing or in the event of system failure/upgrades. It is critical to migrate data to their target locations as quickly as possible to obtain the best performance. Most of the previous results on data migration assume that each storage node can perform only one data transfer at a time. A storage node, however, can typically handle multiple transfers simultaneously and this can reduce the total migration time significantly. Moreover, storage devices tend to have heterogeneous capabilities as devices may be added over time due to storage demand increase.**

In this paper, we consider the *heterogeneous data migration* problem, where we assume that each storage node v has different transfer constraint c_v, which represents how many *simultaneous* transfers v can handle. We develop algorithms to minimize the data migration time. We show that it is possible to find an optimal migration schedule when all c_vs are even. Furthermore, though the problem is NP-hard in general, we give an efficient algorithm that offers a rigorous $(1 + o(1))$-approximation guarantee.

I. Introduction

Large-scale storage systems are crucial components for today's data-intensive applications such as search engine clusters, video-on-demand servers, sensor networks, and grid computing. A storage cluster can consist of several hundreds to thousands of storage devices, which are typically connected using a dedicated high-speed network. In such systems, data locations may have to be changed over time for load balancing or in the event of disk addition and removal. It is critical to migrate data to their target disks as quickly as possible to obtain the best performance of the system since the storage system will perform sub-optimally until migrations are finished.

For example, in a system consisting of parallel disks, data layouts are computed based on many factors such as user demands and disk space constraints to balance the load. Data layouts need to be changed over time according to changes of user demand patterns. In these cases, data need to be quickly moved to their target disks to obtain a new data layout as soon as possible.

Another example of data migration is when we add or remove disks in a storage system. In a large-scale storage system, disk removals, additions and replacements can occur frequently. For example, in a search engine cluster with hundreds of storage devices, such system upgrades and replacements are required as often as every few days [1]. In the event of disk additions and removals, it is necessary to quickly redistribute or recover data so that the system can run in the best performance under the new configuration.

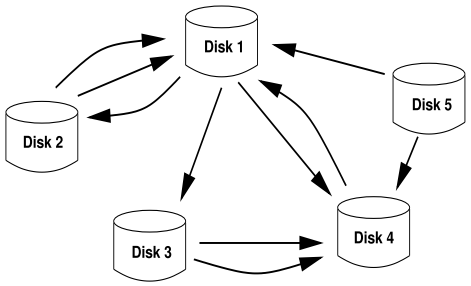

Fig. 1. An example of data transfer instance

The data migration problem can be informally defined as follows. We have a set of disks $v_1, v_2, \ldots, v_n$ and a set of data items $i_1, i_2, \ldots, i_m$. Initially, each disk stores a subset of items. Over time, data items may be moved to another disk for load balancing or for system reconfiguration. We can construct a *transfer graph* $G = (V, E)$ where each node represents a disk and an edge $e = (u, v)$ represents a data item to be moved from disk u to v. Note that the transfer graph can be a multi-graph (i.e., there can be multiple edges between two nodes) when multiple data items are to be moved from one disk to another (for example see Figure 1).

Most of the previous results on data migration assume that each disk can perform only one transfer at a time. A disk, however, may be able to handle multiple transfers simultaneously and this can reduce the total migration time significantly. For example consider Figure 2, suppose that we have three disks in the system and for each pair of nodes we have M items to be moved and it takes one time unit to send one data item from one disk to another. Assuming that each node can participate in only one transfer, it will take $3M$ time units. However, if a disk can perform two transfers at a time by using half of its bandwidth for each transfer, the migration can be done is $2M$ time units. (It requires M rounds but each round takes 2 time units as the bandwidth is split.)

Previous research has focused mainly on homogeneous

1063-6927/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDCS.2011.46

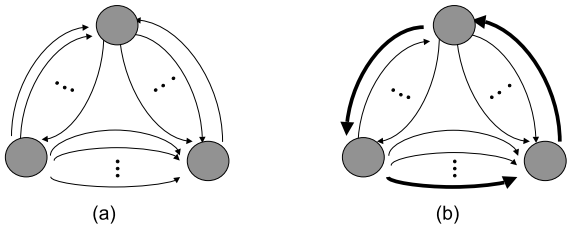

Fig. 2. An example of data migration when each node can handle two transfers simultaneously

storage systems, in which it is assumed that disks in a system all have the same capabilities [2], [3], [4], [5], [6], [7]. In large-scale storage systems, however, storage devices tend to have heterogeneous capabilities as devices are added over time as storage capacity demand increases. Moreover, available bandwidth of each disk can be different depending on current user traffic on different disks. That is, we may not want a disk to be involved in too many migrations if the disk is currently serving many clients. In case each disk has a different bandwidth available for migration, assuming that each node participates in only one transfer will significantly degrade the finish time of the data migration schedule as a slow node can be a bottleneck in the schedule.

here

In this paper, we consider the *heterogeneous data migration* problem where we will assume that each disk has different transfer constraints c_v, representing how many transfers the disk can handle simultaneously and develop algorithms to minimize the data migration time. Coffman et al. [8] considered the problem and developed optimal algorithms for several special cases such as cycles, trees, and bipartite graphs. Saia [9] presented a simple 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 [10] then gives an edge coloring algorithm with at most $1.5\lceil d_v/c_v \rceil$ colors.

We show that it is possible to find an *optimal* migration schedule when all c_v's are even. This can be done by decomposing the transfer graph into $max_v\lceil d_v/c_v \rceil$ components where in each component at most c_v edges are incident to v. For arbitrary c_v, we go on to show that there is an algorithm that uses at most $OPT + \sqrt{OPT}$ colors. This is a $(1+o(1))$-algorithm and the approximation factor is close to 1 when OPT increases.

The remainder of this paper is organized as follows. We first describe the related work in Section II. In Section III, we formally define the problem, along with two lower bounds on the optimal solution. In Section IV, we present a polynomial time algorithm that finds an optimal migration schedule when c_v's are even. Finally, in Section V we give a $(1 + o(1))$-approximation algorithm for the general case.

II. Related work

Several approximation algorithms have been developed for data migration in local area networks [5], [11], [7], [12], [13], [14]. In their ground-breaking work, Hall et al. [4] studied the problem of scheduling migrations given a set of disks, with each storing a subset of items and a specified set of migrations. A crucial constraint in their problem is that each disk can participate in only one migration at a time. If both disks and data items are identical, this is exactly the problem of edge-coloring a multi-graph. That is, we can create a transfer graph $G(V, E)$ that has a node corresponding to each disk, and a directed edge corresponding to each migration 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. Computing a solution with the minimum number of colors is NP-hard [15], but several approximation algorithms are available for edge coloring [16], [17], [18], [19], [20]. With space constraints on disks, the problem becomes more challenging. Hall et al. [4] showed that with the assumption that each disk has one spare unit of space, very good constant factor approximations can be obtained. The algorithms use at most $4\lceil \Delta/4 \rceil$ colors with at most $n/3$ bypass nodes,[1] or at most $6\lceil \Delta/4 \rceil$ colors without bypass nodes where Δ is the maximum degree of the transfer graph and n is the number of disks.

For multi-graph edge coloring, a simple generalization of Vizing's theorem gives a solution using at most $\Delta + \mu$ colors where Δ is the maximum degree of a node and μ is the multiplicity. When $\mu > \Delta/2$, a solution using $3\Delta/2$ can be obtained [10]. The best approximation ratio for the problem have long been $1.1OPT + 0.7$ [17], [19]. Recently, Sanders and Steurer obtained $1 + o(1)$-approximate solution. We show that when each color can be used c_v times for each node and c_v are even for all v, then there is a polynomial time algorithm to find an optimal solution. We obtain the matching approximation ratio for the general cases where each node has arbitrary c_v.

Coffman et al. [8] and Sanders et al. [21] studied the problem of data migration with forwarding (data exchanged between disk i and disk j is not necessarily delivered directly from i to j but can be forwarded over other disks). Whitehead [22] shows that when forwarding is needed because some of the edges in the transfer graph are not present in the

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

interconnection network, then the problem is NP-complete. In this paper, we assume that transfers between disks can be done through a very fast network connection dedicated to support a storage system and any two disks can send data to each other directly.

In the data migration problem with cloning, each data item may have several copies in different disks [6], [3], [13], [11]. To handle high demand for popular items or for fault-tolerance, multiple copies may have to be created and stored on different disks [23], [24]. In the problem, each data item i initially belongs to a subset of source disks S_i and needs to be moved to another subset of destination disks D_i. It is also assumed that each data transfer takes the same amount of time and each disk can participate in only one transfer — either send or receive. The objective is to minimize the time taken to complete the migration. It is shown that this problem is NP-hard by a reduction from edge coloring and a polynomial-time algorithm with approximation factor of 9.5 is developed [6], which was further improved to 6.5 + o(1) recently [11].

Another interesting objective function in data migration is to minimize the total completion time. Minimizing the sum of weighted job completion times is one of the most common measures in the scheduling literature since it can reflect the priorities of jobs. Kim [14] developed a LP-based approximation algorithm with approximation ratio of 9, which was later improved to a 5.06-approximation by Gandhi et al. [25]. On the other hand, in a system where storage devices can be free to do other tasks as soon as their own migrations are complete, minimizing the sum of completion times over all storage devices is interesting since the performance of a device is degraded while it is involved in the migration. For the objective, Kim [14] developed 10-approximation algorithm and Gandhi et al. improved the ratio to 7.682 [26].

III. Problem Definition

In the Heterogeneous Data Migration problem, we are given a transfer graph $G = (V, E)$. Each node in V represents a disk in the system and each edge $e = (i, j)$ in E represents a data item that need to be transferred from i to j. We assume that each data item has the same length, and therefore it takes the same amount of time for each data to migrate. Note that the resulting graph is a multi-graph as there may be several data items to be sent from disk i to disk j.

We assume that each disk v can handle multiple transfers at a time. Transfer constraint c_v represents how many parallel data transfers the disk v can perform simultaneously.

Our objective is to minimize the number of rounds to finish all data migrations. As noted above, we assume disks send data to each other directly.

A. Lower Bounds

We have the following two lower bounds on the optimal solution.

$$LB_1 = \Delta' = \max_v \lceil d_v/c_v \rceil \quad (1)$$

$$LB_2 = \Gamma' = \max_{S \subseteq V} \frac{|E(S)|}{\lfloor \frac{\sum_{v \in S} c_v}{2} \rfloor} \quad (2)$$

where $E(S) = \{(u, v) \in E \text{ s.t. } u, v \in S\}$ is the set of edges both of which endpoints are in S.

LB_1 follows from the fact that for a node v, at most c_v data items can be migrated in a round. When $\forall v \in V, c_v$ is even, $LB_1 \leq LB_2$ and, in fact, we show that there is a migration schedule that can be completed in LB_1 rounds. The following lemma proves that LB_2 is a lower bound on the optimal solution.

Lemma 3.1: LB_2 is a lower bound on the optimal solution.

Proof: An optimal migration is a decomposition of edges in E into $E_1, E_2, \ldots, E_k$ such that for each E_i and a vertex v, there are at most c_v edges incident to v in E_i. For a subset $S \subseteq V$, let $d_i(v, S)$ be the number of edges in $E_i(S)$ incident to v. Then $2|E_i(S)| = \sum_{v \in S} d_i(v, S)$. As $d_i(v, S) \leq c_v$, we have $|E_i(S)| \leq \lfloor \frac{\sum_{v \in S} c_v}{2} \rfloor$. As E_i's cover all edges in $E(S)$, the lemma follows. ∎

IV. The Cases for Even Transfer Constraints

In this section, we describe a polynomial time algorithm to find an optimal migration schedule when each node v has even transfer constraint c_v. We show that it is possible to obtain a migration schedule using Δ' rounds.

A. Outline of Algorithm

We first present the outline of our algorithm when c_v is even for any v. The details of each step is given in Section IV-B.

(1) Construct G' so that every node has degree exactly $c_v\Delta'$ by adding dummy edges.
(2) Find a Euler cycle (EC) on G'.
(3) Construct a bipartite graph H by considering the directions of edges obtained in EC. That is, for each node v in G', create two copies v_{in} and v_{out}. For an edge $e = (u, v)$ in G', if the edge is visited from u to v in EC, then create an edge from u_{out} to v_{in} in H.
(4) We now decompose H into Δ' components by repeatedly finding a $c_v/2$-matching in H.
(5) Let $M_1, M_2, \ldots, M_{\Delta'}$ be the matching obtained in Step (4). Then schedule one matching in each round.

B. Detailed Description and Analysis

We now describe the details and show that the algorithm gives an optimal migration schedule.

Step (1)-(3): The first three steps are a generalization of Peterson's theorem. G' can be constructed as follows. For any node v with degree less than $c_v\Delta'$, we add self-loops until degree of the node becomes at least $c_v\Delta' - 1$. Note that after

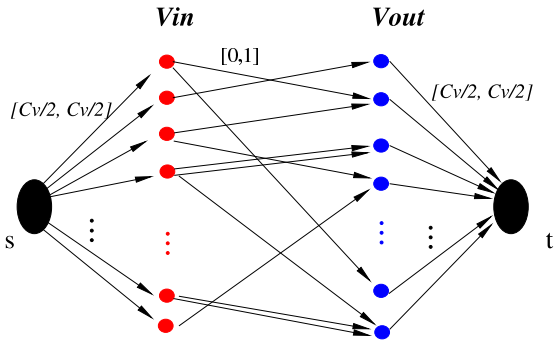

Fig. 3. Flow Network for Step (4)

the modification, the number of nodes with degree $c_v\Delta' - 1$ is even as c_v's are even. Pair the nodes and add edges so that every node has degree $c_v\Delta'$.

Note that each node in G' has even degree as all c_v's are even. Therefore, we can find a Euler cycle EC on G'. Note that for each node v, there are $c_v\Delta'/2$ incoming edges and $c_v\Delta'/2$ outgoing edges in EC .

We construct a bipartite graph H by considering the directions of edges obtained in EC. For each node v in G', create two copies v_{in} and v_{out}. For an edge $e = (u, v)$ in G', if the edge is visited from u to v in EC, then create an edge from u_{out} to v_{in} in H. As each node v in G' has $c_v\Delta'/2$ incoming edges and $c_v\Delta'/2$ outgoing edges in EC, the degrees of v_{in} and v_{out} in H is also $c_v\Delta'/2$.

Step (4): We now find a $c_v/2$-matching in H where exactly $c_v/2$ edges are matched for each v_{in} and v_{out}. We show the following lemma.

Lemma 4.1: There exists a $c_v/2$-matching in H and it can be found in polynomial time.

Proof: We construct a flow network as shown in Figure 3. Create a source s and sink t. There are edges from s to all nodes v_{out} and also from all nodes v_{in} to t. The capacities of edges are $c_v/2$. The capacities of edges from v_{in} and v_{out} are bounded above by 1 and bounded below by 0.

We show that there is a fractional flow from s to t with total flow $\sum_v c_v/2$. Then by the integrality theorem, we can find an integral flow with the same total flow using, e.g., the Ford-Fulkerson algorithm.

Consider a fractional flow sending $1/\Delta'$ flow through each edge from v_{out} to v_{in}. As each node v_{in} has $c_v\Delta'/2$ incoming edges, we may send $c_v/2$ flow to v_{in}; this satisfies the capacity constraints for edges from s. We can show that the capacity constraints for edges to t are also satisfied in the same way.

Since we send $c_v/2$ for each edge from s to v_{in}, the total is $\sum_v c_v/2$. The lemma follows. ■

Lemma 4.2: We can decompose H into $M_1, M_2, \ldots, M_{\Delta'}$ so that each M_i is a $c_v/2$-matching in H.

Proof: Once we find a $c_v/2$-matching in H, we remove the matched edges from H. Let M_1 denote the removed matched edges. Note that in the modified H, each node has the degree exactly $c_v(\Delta' - 1)/2$. Then we can show that there is a fractional matching by sending $1/(\Delta' - 1)$ flow through each edge from v_{out} to v_{in}. In general, after removing $M_1, M_2, \ldots, M_i$ from H, each node has the degree $c_v(\Delta' - i)/2$ and sending $1/(\Delta' - i)$ flow through each edge from v_{out} to v_{in} gives a $c_v/2$-fractional matching. Therefore, we can find a $c_v/2$ matching in each iteration. As each node has degree of $c_v\Delta'/2$, we can repeat this Δ' times and obtain $M_1, M_2, \ldots, M_{\Delta'}$. ■

Step (5): Each component M_i can be scheduled in one round and therefore, we have an optimal migration schedule for even capacities.

Lemma 4.3: Each component M_i can be scheduled in one round.

Proof: Each node v in G' has two copies in H — v_{in} and v_{out}. As v_{in} and v_{out} both have $c_v/2$ edges incident to them in M_i, the total number of edges that are matched in M_i and incident to v in G' is c_v. Therefore, each component M_i can be scheduled in one round. ■

Theorem 4.1: We can find an optimal migration schedule when each node has even c_v.

V. General Cases

In this section, we consider the case that each node v has an arbitrary c_v. The problem is NP-hard as it is NP-hard even when $c_v = 1$ for all nodes. We develop an algorithm that colors edges of the given graph so that the transfer constraints c_v of the nodes are satisfied. The coloring defines a data migration schedule. As the number of colors used determines the number of rounds in our schedule, we would like our coloring algorithm to minimize the number of colors needed. We obtain an algorithm that uses at most $OPT + \sqrt{OPT}$

A. Outline of the Algorithm

We first give an overview of the coloring algorithm. Our algorithm generalizes recent work for multi-graph edge coloring by Sanders and Steurer [20]. Our algorithm uses three particular subgraph structures, balancing orbits, color orbits and edge orbits defined in section V-B. The latter two structures — color orbits and edge orbits — are generalizations of the structures used by Sanders and Steurer [20].

The algorithm starts with a naive partial coloring of $G = (V, E)$ and proceeds in two phases. In the first phase, we use three structures and color edges until we produce a simple uncolored subgraph G_0 (Section V-C1) consisting of small connected components (Section V-C2); in the second phase we color G_0 and show that $O(\sqrt{d_v(G_0)/\min c_v})$ new colors are enough to obtain a proper coloring in G_0 (Section V-C3).

B. Preliminaries

We first introduce some definitions. Let $|E_i(v)|$ be the number of edges of color i adjacent to a vertex v.

Definition 5.1 (Strongly/lightly missing color): Color c is *saturated* at vertex v if $|E_c(v)| = c_v$. The color c is *missing* at vertex v if $|E_c(v)|$ is less than c_v; in this case we distinguish two possibilities:

- c is *strongly missing* if $|E_c(v)| < c_v - 1$.

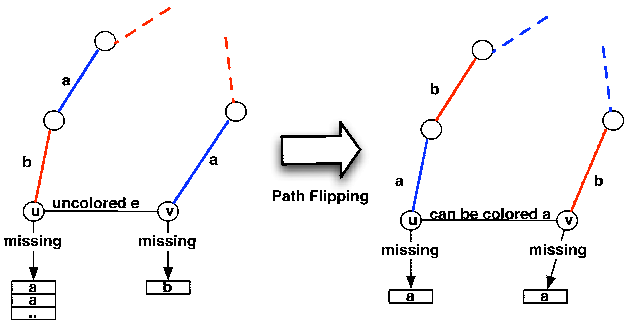

Fig. 4. u strongly missing a and path P ends at v, we can color e with a

- c is *lightly missing* if $|E_c(v)| = c_v - 1$.

Definition 5.2 (alternating path): An *ab-path* between vertices u and v where a and b are colors, is a path connecting u and v and has the following properties:

- Edges in the path have alternating colors a and b.
- Let $e_1 = (u, w_1)$ be the first edge on the path and suppose e_1 is colored a; then u must be missing b and not missing a.
- If v is reached, for the first time, by an edge colored b then v must be missing a but not missing b; similarly, if v is reached, for the first time, by an edge colored a then v must be missing b and not missing a.

A *flipping* of an ab-path is a recoloring of the edges on the path such that edges previously with color a will be recolored with color b and vice versa.

Note that unlike the case when $c_v = 1$, an alternating path may not be a simple path in our problem as there can be multiple edges with the same color incident to a node.

1) Balancing Orbits: We first define balancing orbits as follows.

Definition 5.3 (balancing orbit): A *balancing orbit* O is a node induced subgraph such that all nodes $V(O)$ are connected by uncolored edges and the following property holds

- A vertex $v \in V(O)$ is strongly missing a color.

The following lemma shows that if we have a balancing orbit, we can color an uncolored edge and eventually remove any balancing orbits.

Lemma 5.1: If there is a balancing orbit in G, then we can color a previously uncolored edge.

Proof: Let O be the balancing orbit. That is, a node $u \in V(O)$ is strongly missing some color. Suppose u is strongly missing color a and let $e = (u, v)$ be the uncolored edge adjacent to u. There are three cases

- v is missing color a: in this case we can simply color e with a.
- v is missing color b and not missing color a: let P be the ab-path starting at u. P ends at a node $u' \neq v$: flipping P will make b missing at u and thus we can color e with b.
- P ends at v: flipping P will make a missing at v and since a was strongly missing at u, u is still missing a, so color e with a (see Figure 4).

■

2) Color Orbits and Edge Orbits: In this section, we define two subgraph structures: a *color orbit* and an *edge orbit*.

Definition 5.4 (Color orbit): A *color orbit* O is a node induced subgraph such that all nodes $V(O)$ are connected by uncolored edges and the following property holds

- There are at least two nodes $u, v \in V(O)$ lightly missing the same color.

Lemma 5.2: [20] If there exists a color orbit in G then we can color a previously uncolored edge.

By Lemma 5.1 and 5.2, whenever we find a balancing orbit or color orbit, we can color a previously uncolored edge and make progress. If neither of properties in Definition 5.3 and 5.4 hold, we call O a *tight color orbit*.

Our goal at the end of Phase 1 of the algorithm is to get a *simple* uncolored graph G_0 consisting of small connected components. That is, in G_0 there cannot be more than one uncolored edges between two nodes. In order to eliminate parallel uncolored edges the following subgraph structure is used.

Definition 5.5 (Lean and bad edges): If an edge e is colored and all its parallel edges are colored then e is a *lean* edge. If e is uncolored and has a parallel uncolored edge then e is a *bad* edge.

Definition 5.6: An *edge orbit* is a subgraph consisting of two uncolored parallel edges (called the *seed* of the edge orbit) and then is inductively defined as follows: Let $e = (x, y)$ be an edge in the edge orbit O, let a and b be missing colors at x and y respectively and let P be the alternating path starting at x then $O \cup P$ is an edge orbit if

- no edge of color a or b is contained in O.
- $\exists v \in P$ that was not in the vertex set of O.

If edge orbit O has a lean edge then O is called a weak edge orbit otherwise O is a tight edge orbit. A color c is free for an edge orbit O if O does not contain an edge with color c.

The following lemma from [20] states that if in some coloring of G, there exists a weak edge orbit then we make progress toward our goal of obtaining G_0 by either coloring a previously uncolored edge or by uncoloring a lean edge and coloring a bad edge.

Lemma 5.3: If a coloring of G contains a weak edge orbit then we can either color a previously uncolored edge or we can uncolor a lean edge and color a bad edge.

A tight edge orbit does not have lean edges so its vertex set is connected by uncolored edges and thus a tight edge orbit is one of the following — a balancing orbit, color orbit or a tight color orbit. When it is a tight color orbit, we cannot make progress toward G_0 and we call it a *hard orbit*. Note that no vertex in a hard orbit is strongly missing a color, no

two nodes are lightly missing the same color, and no edge in a hard orbit is lean.

3) Growing Orbits: A color c is *full* in a hard orbit O if c is saturated on all vertices of $V(O)$ but at most one vertex in $V(O)$ is lightly missing c or equivalently if $|E_c(v) \cap E(V(O))| \geq \lfloor \frac{\sum_{v \in V(O)} c_v}{2} \rfloor$. So if color c is full in a hard orbit O it cannot be used to color uncolored edges whose endpoints are in O.

Definition 5.7 (Lower bound witnesses): A hard orbit is a Δ'-witness if all missing colors at some node are non-free. It is a Γ'-witness if all free colors of the orbit are full.

The intuition behind the witnesses is the following. Suppose very few colors are used in hard orbit O, in the case of Γ'-witness almost all color classes are full in O and in the case of a Δ'-witness almost all available colors are strong on some node $v \in V(O)$. So a witness in some coloring using q colors indicates that it is *almost* impossible to color an additional uncolored edge using the available q colors and thus the number of available colors needs to be increased.

Lemma 5.4: [20] Given a hard orbit in some coloring we can either find a witness or compute a larger edge orbit.

C. Algorithm

The algorithm proceeds in two phases. The outcome of the first phase would be G_0, a simple uncolored graph with no large components. The following procedure for the first phase eliminates all the bad edges in G (Section V-C1) and reduces the size of connected components (Section V-C2), which gives G_0 with the desired properties. In the second phase (Section V-C3), we color the remaining subgraph G_0.

1) Eliminating bad edges: Given a partial coloring using q colors, we iterate over a list of bad edges and we execute the following steps.Given an edge orbit O

(1) If nodes of O form a balancing or color orbit, apply Lemma 5.1 or 5.2.
(2) If O is weak, apply Lemma 5.3.
(3) If O is a hard orbit, apply Lemma 5.4.
   a) If Lemma 5.4 gives a larger edge orbit $O \cup P$, repeat with $O = O \cup P$.
   b) If Lemma 5.4 gives a witness then increase q by one color and color the bad edges in the seed with the additional color.

The output of this procedure is a simple subgraph G' of G induced by uncolored edges. In Lemma 5.5 and Lemma 5.6, we show an upper bound on the number of used colors if there is a Δ' or Γ'-witness. The next procedure reduces the size of the connected components of G' whenever G' has balancing or color orbits.

2) Reducing size of connected components: For every connected component U of G',

1) If U contains a vertex that is strongly missing a color then use Lemma 5.1 to color an uncolored edge.
2) If U contains two or more vertices that are lightly missing the same color use Lemma 5.2 to color an uncolored edge.

So at the end of the first phase we have the simple subgraph G_0 where for every connected component U of G_0, no vertex is strongly missing a color and no two vertices of U miss the same color. In Lemma 5.7, we show that the size of G_0 is no more than $\frac{q+2}{q-\Delta'+2}$.

3) Coloring G_0: Phase 2 colors G_0. We use only $\max_v \lceil \frac{d_v(G_0)}{c_v} \rceil + 1$ colors. The procedure goes as follows:

1) Create c_v copies of each vertex v and distribute the edges over the copies so that each vertex is adjacent to at most $\lceil \frac{d_v(G_0)}{c_v} \rceil$ edges where $d_v(G_0)$ represents the degree of v in G_0.
2) Use Vizing's algorithm to properly color each component. We need at most $\max_v \lceil \frac{d_v(G_0)}{c_v} \rceil$ + 1 colors.
3) Contract the copies back to v getting a coloring where for any node v there is no more than c_v edges of the same color.

D. Analysis

In the following q denotes the total number of colors available for the algorithm. We show that the algorithm colors all the edges of G using at most $q = OPT + \Theta(\sqrt{OPT})$ colors. We first bound the number of used colors when there is a Δ' or Γ'-witness.

Lemma 5.5: Let O be a hard orbit. If O is a Δ'-witness then $q \leq \Delta' + \frac{2|V(O)|-4}{c^-}$ where $c^- = \min_{v \in V(O)} c_v$.

Proof: For a trivial edge orbit all q colors are free. Other edge orbits are constructed by adding an alternating path with new colors to another edge orbit. Every path adds at least a new node to the edge orbit and reduces the number of free colors by at most 2. So the total number of free colors for an edge orbit is at least $q - 2(|V(O)| - 2)$.

If O is a Δ'-witness then some node $u \in V(O)$ misses no free color. Since u is not strongly missing any color and is incident to at least two uncolored edges, the number of missing colors for u is at least $c_u q - d_u + 2$. As the number of free colors and missing colors of u are disjoint then

$$\begin{aligned} q - 2|V(O)| + 4 + c_u q - d_u + 2 &\leq q \\ c_u q &\leq d_u + 2|V(O)| - 6 \\ q &\leq \Delta' + \frac{2|V(O)| - 4}{c^-} \end{aligned}$$

■

Lemma 5.6: Let O be a hard orbit. If O is a Γ'-witness then $q \leq \Gamma' + 2|V(O)| - 4 - \frac{2}{c^+}$.

Proof: If O is a Γ'-witness then all free colors of O are full. Since O is a hard orbit there is at most $|E(V(O)) - V(O)|$ colored edges between vertices in $V(O)$. So the number of full colors is at most

$$\frac{|E(V(O)) - V(O)|}{\lfloor \frac{\sum_{v \in V(O)} c_v}{2} \rfloor} \leq \Gamma' - \frac{V(O)}{\lfloor \frac{\sum_{v \in V(O)} c_v}{2} \rfloor}.$$

Let $c^+ = \max_{v \in V(O)} c_v$. Then the number of full colors is at most $\Gamma' - 2/c^+$. As the total number of free colors for an edge orbit is at least $q - 2(|V(O)| - 2)$, the lemma follows.

■

We now bound the size of G_0.

Lemma 5.7: Let O be a tight color orbit. Then $|V(O)| \leq \frac{q+2}{q-\Delta'+2}$.

Proof: Since no two nodes in $V(O)$ share a missing color, the number of missing colors at the nodes of O is less than q. Every node u in $V(O)$ misses at least $c_u q - d_u (\geq q - \Delta')$ colors as no node in $V(O)$ is strongly missing a color. The uncolored edge adjacent to a node $u \in V(O)$ implies an additional color is missing at u. As there are $|V(O)| - 1$ uncolored edges in O there are at least $2(|V(O)|-1)$ additional colors missing at nodes in $V(O)$. Thus we have

$$|V(O)|(q - \Delta') + 2(|V(O)| - 1) \leq q\,,$$
$$q|V(O)| - \Delta'|V(O)| + 2(|V(O)| - 1) \leq q\,, \text{ and thus}$$
$$|V(O)| \leq \frac{q+2}{q-\Delta'+2}\,.$$

■

Corollary 5.1: If $q = \lfloor (1+\epsilon)\Delta' \rfloor - 1$, the size of a hard orbit O is at most $1 + \frac{1}{\epsilon}$.

Proof: A hard orbit is also a tight color orbit and thus

$$|V(O)| \leq \frac{q+2}{q-\Delta'+2} \leq 1 + \frac{\Delta'}{\lfloor \epsilon\Delta' \rfloor + 1} \leq 1 + \frac{1}{\epsilon}\,.$$

■

The following corollary follows from Lemma 5.5, 5.6, and Corollary 5.1,

Corollary 5.2: If $q = \lfloor (1+\epsilon)\Delta' \rfloor - 1$ and there is a witness then $q \leq OPT + \frac{2}{\epsilon} - 2$

The following lemma provides a bound on the number of required colors for G_0.

Lemma 5.8: Suppose that the size of the largest component of G_0 is bounded by C. Then coloring G_0 requires at most $\lceil \frac{C-1}{c^-} \rceil + 1$ colors.

Proof: By Vizing's theorem for simple graphs, at most $\max \lceil \frac{d_v(G_0)}{c_v} \rceil + 1$ new colors will be used in Phase 2 as described in Section V-C3. This procedure colors G_0 using at most $\lceil \frac{C-1}{c^-} \rceil + 1$ as $d_v(G_0) \leq C - 1$.

■

Theorem 5.1: Given a transfer graph G, we can compute a coloring of the edges using at most $OPT + O\left(\sqrt{OPT}\right)$ colors.

Proof: We start our coloring algorithm with $\lfloor (1+\epsilon)\Delta' \rfloor - 1$ colors. At the end of Phase 1, all edges that remain uncolored create a simple subgraph G_0 (a collection of hard orbits). Furthermore, the number of colors has increased only if there was a witness. By Corollary 5.2, the number of colors used is at most $OPT + \frac{2}{\epsilon} - 2$. Thus at the end of phase 1, at most $\max(\lfloor (1+\epsilon)\Delta' \rfloor - 1, OPT + \frac{2}{\epsilon} - 2)$ colors have been used. In Phase 2, the algorithm colors the connected components of G_0 which are hard orbits. By Corollary 5.1 their size is at most $1 + 1/\epsilon$. G_0 is a simple graph and has degree no more than $1/\epsilon$ thus the algorithm in Section V-C3 yields a coloring of G_0 using at most $1 + 1/(\epsilon \cdot c^-)$ additional colors by Lemma 5.8. So the total number of colors used in the coloring algorithm is at most the maximum of $\lfloor (1+\epsilon)\Delta' \rfloor + \frac{1}{\epsilon c^-}, OPT + \frac{2}{\epsilon} - \frac{1}{\epsilon c^-} - 1)$. Choosing $\epsilon = \frac{\sqrt{2}}{\sqrt{OPT}}$ gives the theorem. ■

We start with $\Delta' + \sqrt{\Delta'}$ colors and proceed as described in Section V-C. The overall running time of the algorithm is $O(|E|\sqrt{\Delta'}(\Delta + |V|))$ by lemma 5.9

Corollary 5.3: The coloring algorithm uses at most $OPT + O(\sqrt{OPT})$ colors, which implies an approximation factor of $1 + o(1)$ as OPT increases.

Lemma 5.9: Let C be such that no connected component of G_0 has size exceeding C, then the runtime of the algorithm is

$$O(|E|C(\Delta + |V|))$$

Proof: As the size of connected components of G_0 is less than C, the missing colors at each node of G_0 can be determined in $O(C\Delta)$. Flipping a path can be done in time proportional to the number of vertices on that path, so at most V. This is iterated at most C times in G_0. So the total time for Lemmas 5.1, 5.2 and 5.3 is $O(C(|V| + \Delta))$. Thus the first 2 steps of procedure V-C1 and procedure V-C2 are $O(C(|V| + \Delta))$. Step 3(b) of procedure V-C1 is done in constant time and in step 3(a) $O(\Delta + |V|)$ steps are needed to grow a hard orbit (Lemma 5.4). As we iterate over all bad edges the overall runtime of the algorithm is $O(|E|C(\Delta + |V|))$ ■

Note that the running time of the algorithm described above is polynomial in $|E|$, $|V|$ and Δ. The authors in [20] develop a solution exploiting that a graph with even edge multiplicities can be colored by coloring a graph with halved edge multiplicities and then using each color twice. The same transformation applies to the above algorithm to make it polynomial in $|V|$ and in the number of bits needed to encode edge multiplicities.

VI. Conclusion

In this paper, we consider the data migration problem where each storage node has different transfer constraint c_v, representing how many transfers the node can simultaneously handle. Our objective is to minimize the data migration time. We show that it is possible to find an optimal migration schedule when c_v is even for all v. For the general case, we formulate the problem as a variant of multi-edge coloring problem in which each color can be used c_v times at node v. Furthermore, though the problem is NP-hard, we give an efficient algorithm that offers a rigorous $(1+o(1))$-approximation guarantee.

References

[1] H. Tang and T. Yang, "An efficient data location protocol for self-organizing storage clusters," in *Proceedings of the International Conference for High Performance Computing and Communications (SC)*, 2003.

[2] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes, "An experimental study of data migration algorithms," in *Workshop on Algorithm Engineering*, 2001.

[3] 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.

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

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

[6] S. Khuller, Y. Kim, and Y.-C. Wan, "Algorithms for data migration with cloning," in *22nd ACM Symposium on Principles of Database Systems (PODS)*, 2003, pp. 27–36.
[7] B. Seo and R. Zimmermann, "Efficient disk replacement and data migration algorithms for large disk subsystems," *ACM Transactions on Storage Journal (TOS)*, vol. 1, no. 3, pp. 316–345, AUG 2005.
[8] E. Coffman, M. G. Jr., D. Johnson, and A. Lapaugh, "Scheduling file transfers," *SIAM J. Computing*, vol. 14, no. 3, pp. 744–780, 1985.
[9] J. Saia, "Data migration with edge capacities and machine speeds," University of Washington, Tech. Rep., 2001. [Online]. Available: citeseer.ifi.unizh.ch/article/saia01data.html
[10] C. Shannon, "A theorem on colouring lines of a network," *J. Math. Phys.*, vol. 28, pp. 148–151, 1949.
[11] S. Khuller, Y. Kim, and A. Malekian, "Improved algorithms for data migration," in *APPROX*, 2006.
[12] C. Lu, G. Alvarez, and J. Wilkes, "Aqueduct:online data migration with performance guarantees," in *Proceedings of the Conference on File and Storage Technologies*, 2002.
[13] S. Khuller, Y.-A. Kim, and Y.-C. J. Wan, "On generalized gossiping and broadcasting," in *12th Annual European Symposium on Algorithms*, 2004.
[14] Y.-A. Kim, "Data migration to minimize the average completion time," *J. of Algorithms*, vol. 55, 2005.
[15] I. Holyer, "The np-completeness of edge-coloring," *SIAM J. on Computing*, vol. 10, no. 4, 1981.
[16] D. Hochbaum, T. Nishizeki, and D.B.Shmoys, "Better than "best possible" algorithm to edge color multigraphs," *J. of Algorithms*, vol. 7, no. 1, 1986.
[17] T. Nishizeki and K. Kashiwagi, "On the 1.1 edge-coloring of multigraphs," *SIAM J. on Discrete Math.*, vol. 3, 1990.
[18] M. Goldberg, "Edge-colorings of multigraphs: Recoloring techniques," *Journal of Graph Theory*, vol. 8, pp. 122–136, 1984.
[19] A. Caprara and R. Rizzi, "Improving a family of approximation algorithms to edge color multi-graphs," *Information Processing Letters*, vol. 68, pp. 11–15, 1998.
[20] P. Sanders and D. Steurer, "An asymptotic approximation scheme for multigraph edge coloring," in *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005, pp. 897–906.
[21] P. Sanders and R. Solis-Oba, "How helpers hasten h-relations," *Journal of Algorithms*, vol. 41, pp. 86–98, 2001.
[22] J. Whitehead, "The complexity of file transfer scheduling with forwarding," *SIAM J. Comput.*, vol. 19, no. 2, pp. 222–245, 1990.
[23] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, "Staggered striping in multimedia information systems," in *ACM SIGMOD International Conference on Management of Data*, 1994, pp. 79–90. [Online]. Available: citeseer.ist.psu.edu/berson93staggered.html
[24] M. Stonebraker, "The case for shared nothing," *Database Engineering Bulletin*, vol. 9, no. 1, pp. 4–9, 1986. [Online]. Available: citeseer.ist.psu.edu/stonebraker86case.html
[25] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai, "Improved results for data migration and open shop scheduling," in *ICALP*, 2004, pp. 658–669.
[26] R. Gandhi, M. Halldórsson, G. Kortsarz, and H. Shachnai, "Improved bounds for sum multicoloring and scheduling dependent jobs with minsum criteria," in *WAOA*, 2004, pp. 68–82.