

Survey on Data Placement and Migration Algorithms in Distributed Disk Systems

Beomjoo Seo, bseo@usc.edu

ABSTRACT

We survey research work done on (distributed) storage device since last decade, especially arranged into three categories : data allocation, data migration, and heterogeneity support for efficient object storage/retrieval. In brief, round-robin based striping, even its simplicity, lacks in on-line scalability and flexibility, whereas randomized allocation is slowly gaining reputation because of its easy adaptability to a new environment. Data migration problem, recently addressed, finds an efficient migration schedule to move the data from one disk configuration to another. Lastly, separated from data allocation category, heterogeneity support issue is also covered in terms of bandwidth and space utilization.

1. OVERVIEW

As more and more data are archived digitally, it becomes very important to keep data consistent or to make them available whenever necessary. [1] Due to the nature of the medium which stores the data, which deteriorate over time, it is growing needs to migrate the data efficiently. Traditional striping based data placement algorithms widely used in RAID, however, does not scale well because of static assignment of disks. [2] Especially, migrations are more difficult to cope with while maintaining the load balance with minimum efforts in heterogeneous disk systems.

In order to maintain the load balance or keep the higher availability of the data, you may replicate the data or copy a fraction of them. Replication is a good approach to isolate the disks from other system resources rather than to balance the loads. [3] But these replication and caching approaches are out of scope and not covered in this paper. In fact, several researches regarding these topics were already covered in the class.

In this paper we focus on discussing the potentials of the on-line scalability support (disk additions or removals) and the heterogeneity support of existing data placement and migration algorithms in distributed disk systems such as RAID, Network Attached Storage (NAS), Storage Area Network (SAN), and P2P networks.

Primary purposes of this survey are (1) to compare all the related research papers on different data allocation and migration algorithms with their performance, scalability, and heterogeneity, and (2) to search several issues not addressed yet in newly suggested distributed environments. Specifically, in dynamically changing environments, it is not well investigated to place and migrate the data efficiently to provide a high degree of availability. For example, for Content Distribution Networks (CDN) that are based on Distributed Hash Table (DHT) techniques we may distribute the media content by segmenting it into successive data

blocks and placing them on distributed disks randomly. [4]

These survey is expected to clarify the problem more clearly on data migration in a distributed way and to provide a good guide on future research.

The reviewed papers are classified as three areas: data placement, data migration, heterogeneity support. In Section 2, two different methodologies of data placements are presented: striped allocations, and randomized allocations. Traditional clustered storage subsystems, such as disk arrays and disk striping, can achieve dramatic disk I/O performance gains by accessing disks in parallel, but they have significant scalability problems especially when adding new disks or removing old disks. [2, 5] Randomized algorithms assign objects to randomly chosen storage nodes and adapt well in a dynamically changing environment. [6, 7, 8, 4] Data migration problem is to find an efficient plan to move the data from one disk subsystem configuration to another. In Section 3, we introduce data migration concept in a formal way. Because data migration problem is addressed recently, it has little research work done yet. As this problem is reduced to an NP-complete problem, there are two heuristic solutions suggested by introducing by-pass node concept [9, 10] and by allowing multiple replication of data. [11] As mentioned above, heterogeneity issue to support different types of disks is hard to solve, but it is more natural in the real world. In [12], a stochastically optimal allocation with the same speed disks is presented. Bandwidth to Space Ratio (BSR) based policy determines to copy or deletes the replicas of the video objects according to the current load in a heterogeneous disk system. [13] Logical disk modelling approach performs an exhaustive search to find the configuration parameters to utilize the disks maximally. [14]

2. DATA ALLOCATION

For many years many research work has been done to improve the disk I/O performance. Especially, from the early 1990s, these efforts were focused on optimizing the performance of a single disk drive for an efficient media streaming. Various low-level and high-level optimizations are suggested : Grouped Sweeping Scheduling (GSS) [15], SCAN-EDF, intelligent buffer cache mechanisms, reconfiguration of a file system lay-out, and data layout based on disk geometry for continuous media.(see Figure 1) As such systems evolve, scalability issue became main concern. However, compared with other system components such as processor and memory, performance improvement of a disk is much slower. Redundant Arrays of Inexpensive Disk (RAID) is a commercial success story to promise improvements of an order of magnitude in performance, reliability, and scalability. [2]

Disk striping is a new methodology to achieve high access concurrency by spreading out the objects across the disks in a stripe set. Additionally, it prevents load imbalance. This method

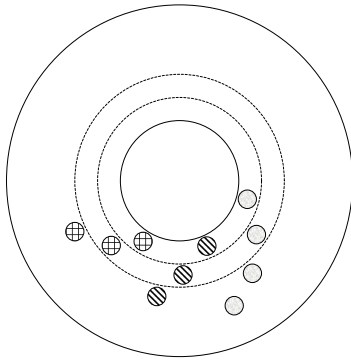


Figure 1: Disk geometry based data placement.

had been well-studied and widely used in high bandwidth and space demanding area. There are two well-known striping policies: round-robin policy, randomized policy. Round-robin policy stripes the data across the disk array in the round-robin order and achieves synchronous parallel access to the disks, resulting in higher throughput. The round-robin order, however, becomes a primary bottleneck when the system reallocates data. Randomized striping policy, disseminating the data across the stripe set randomly, offers an attractive alternative to round-robin striping. Unexpectedly, randomized striping performs well and its performance is comparable to the round-robin striping.

In this section, we specifically discuss the pros and cons of the striping based data placement method.

2.1 Round-robin Striping

The main advantages of striping are :

Load balance Each load of the disks in a stripe set is well balanced statistically.

Faster Response Time Average response time of the request is faster than conventional data placement implementation.

Aggregated bandwidth Total bandwidth is aggregative by the number of striped disks even though not linearly incremented.

Simplicity Due to a simple placement order, next block location can be easily computed without any lookup operation.

The disadvantages are :

Poor heterogeneity support Allocation is more difficult to handle due to the heterogeneous nature of the disk.

Expensive data migration cost Expensive data migration is required when storage devices join or leave from the stripe set (storage reconfiguration). For example, Figure 2 shows that reconfiguring the round-robin striping from a stripe set 3 to a set 4 requires 75% ($= \frac{18}{24}$) data movement to preserve the round-robin order, which is not acceptable for many applications.

In-adaptability for workload changes It is difficult to find appropriate stripe size, or stripe level because the workload of stored objects is characterized by the static and dynamic load imbalance.

There are two primary criteria to compare various striping techniques: performance and load balance. Performance is measured

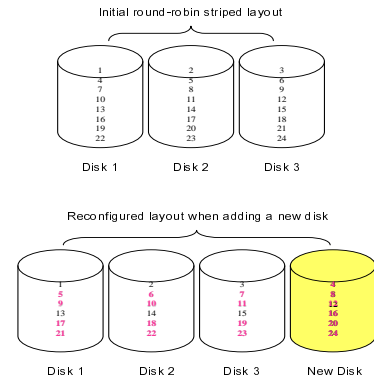


Figure 2: Example of bad reorganization of data layout based on round-robin striping when adding a new disk.

by the number of simultaneous object accesses, or the number of successfully responded I/O requests for a given period; load balance is measured by counting the average queue length of individual disks and comparing the variances of the queue lengths.

However, striped allocation is widely used in streaming area due to its easy deployment to increase the disk I/O bandwidth through a simple disk additions. But it is questionable whether striping can also support small-size file objects as well as large files. In [16], it reports the benchmark result that request size of disk I/O should be less than the stripe depth in order to prevent load imbalance. Nevertheless, it still has sudden load imbalance because of unexpectedness of the load. Hence, the replication may be helpful to prevent the load imbalance.

2.1.1 RAID

Patterson et al. propose five different organizations of disk arrays. [2] Among them, only mirroring and RAID level 5 are commercially deployed. [17] Striping over the disk array which maximizes concurrent I/O capability of the system is more beneficial than individual access of each disk because individual access suffers from high load imbalance although it has higher performance gain than striping. By design, stripe size involves all the disks, which may waste disk bandwidth.

2.1.2 Staggered Striping

Staggered striping, a variant of striping based data placement policies, subdivides a large stripe disk set into several different logical stripe sets, and minimizes the percentage of disk bandwidth that is wasted when the storage server consists of heterogeneous media objects with different bandwidth requirements.

For example, there are 12 striped disks; each disk bandwidth is 20 Mbp; the display requirements of three multimedia objects X, Y, Z are 60 Mbps, 80 Mbps and 40 Mbps respectively. Traditional striping allocates the consecutive data blocks across all 12 disks in a round-robin way. When a serving a request that references object X , it would sacrifice 75% of the available disk bandwidth. Staggered striping creates three different types of logical stripe sets ($G=3$ for X ; 4 for Y ; 2 for Z) per round, reflecting the display requirement of each object. Segments of object X for the first round ($X0.0, X0.1, X0.2$) are allocated at a logical stripe set ($G=3$) individually. For the next round allocation ($X1.0, X1.1, X1.2$), new logical stripe set with the same stripe size is created, shifted and wrapped around along the disks. Figure 3 illustrates staggered data layout of all segmented objects X, Y, Z . The Y axis of Figure 3 implies the round.

Staggered striping retrieves segments from the associated log-

	Disk 0												Disk 11											
Subject 0	Y0.0	Y0.1	Y0.2	Y0.3	X0.0	X0.1	X0.2	Z0.0	Z0.1															
1		Y1.0	Y1.1	Y1.2	Y1.3	X1.0	X1.1	X1.2	Z1.0	Z1.1														
2			Y2.0	Y2.1	Y2.2	Y2.3	X2.0	X2.1	X2.2	Z2.0	Z2.1													
3				Y3.0	Y3.1	Y3.2	Y3.3	X3.0	X3.1	X3.2	Z3.0	Z3.1												
4	Z4.1				Y4.0	Y4.1	Y4.2	Y4.3	X4.0	X4.1	X4.2	Z4.0												
5	Z5.0	Z5.1				Y5.0	Y5.1	Y5.2	Y5.3	X5.0	X5.1	X5.2												
6	X6.2	Z6.0	Z6.1				Y6.0	Y6.1	Y6.2	Y6.3	X6.0	X6.1												
7	X7.1	X7.2	Z7.0	Z7.1				Y7.0	Y7.1	Y7.2	Y7.3	X7.0												
8	X8.0	X8.1	X8.2	Z8.0	Z8.1				Y8.0	Y8.1	Y8.2	Y8.3												
9	Y9.3	X9.0	X9.1	X9.2	Z9.0	Z9.1				Y9.0	Y9.1	Y9.2												
10	Y10.2	Y10.3	X10.0	X10.1	X10.2	Z10.0	Z10.1				Y10.0	Y10.1												
11	Y11.1	Y11.2	Y11.3	X11.0	X11.1	X11.2	Z11.0	Z11.1				Y11.0												
12	Y12.0	Y12.1	Y12.2	Y12.3	X12.0	X12.1	X12.2	Z12.0	Z12.1															

Figure 3: Staggered striping with 12 disks, 3 objects.

ical stripe set, whose size is configured with the number of disk drives necessary for displaying the object with no hiccups. Thus, it does not waste the bandwidth of the disks. [5] Performance result also shows that staggered striping is superior to traditional striping in terms of performance, eliminating the need for the data replication. But load imbalance may happen when there is an overlapping area between two logical stripe sets, which can be avoided by postponing the retrieval requests temporarily until two overlapped sets become disjoint. Thus, start-up latency after issuing a streaming request is increased slightly.

2.1.3 HP AutoRAID

HP AutoRAID system, a small disk array that supports a two-level hierarchy, migrates the data from existing subset to another subset to provide a full redundancy and excellent performance at higher level, and stripes the data across a small subset of all the disks at lower level. [17] Data layout at lower level can be configured as either RAID level 1 or RAID level 5. As shown in Figure 4, data spaces on the disks are segmented into large objects called Physical EXtents (PEXes). A Physical Extent Group (PEG) consists of several PEXes. At least, three of the PEXes in a PEG should be placed in separate disks in order to provide a redundancy. A PEG is stored as a form of either RAID level 1 or level 5. Segment is the unit of contiguous space (or stripe unit) and replication of a hot-spot object is performed on its segment basis.

This system includes following features which cannot be found at any other RAID systems.

Adaptive object replication for workload changes Active data is copied to mirroring storage and then removed from that storage space when the data becomes less active.

On-line storage capacity expansion When a disk is added to the array, data is gradually balanced across the previous disks and a new one.

From these features, data migration becomes an important performance factor of the system. When copying data from one disk to another, efficient migration plan is computed, which is covered in Section 3.1.

When updating old disks, it removes one old disk at a time, inserting a new disk, and then waiting for the automatic data reorganization and rebalancing to complete without interrupting its operation, which, however, turned out to be a poor decision in terms of total completion time.(see Section 3.3.)

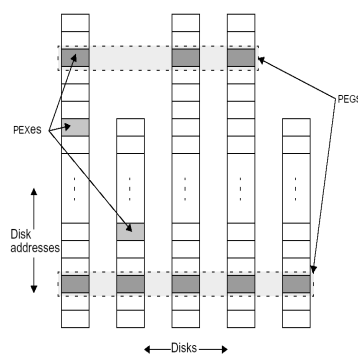


Figure 4: Data layout and the mapping of PEGs and PEXes onto disks of HP AutoRAID.

2.2 Randomized Striping

Randomized allocation across the disk array is now gaining a spotlight in data placement area. It is attributable to on-line scalability, extensibility, and its cost-effectiveness. The RIO [6] system and Yima [18] multimedia server are such examples. In Section 2.2.1, it describes the basic concept of randomized allocation. Section 2.2.2 reports the performance result when randomized allocation is combined with replication. Section 2.2.3 discusses the applicability of disk addition and removal with randomized allocation theoretically. Finally, Section 2.2.4 presents specific pseudo-random function that can be applicable to efficient disk addition and removal scenario. It also reports whether randomness of data can be preserved after the disk scaling operations.

2.2.1 Randomized I/O (RIO)

Unpredictability of data access pattern makes it difficult to keep optimal data lay-out on striped disks and to provide interactive operations for continuous media such as *fast forward*, *fast rewind*, *jump* in round-robin striping. In Randomized I/O (RIO) server system, multimedia objects are divided into fixed-size data blocks and stored at random locations on randomly chosen disks. Randomized allocation which does not rely on a careful data layout simplifies the retrieval scheduling and does not worry about the different types of workload any more. [6] Moreover, this approach is more flexible and scalable when adding new disks, deleting existing disks, or replacing old disks, which will be covered in later section.

Simulation for comparing randomized allocation and round-robin striping says that RIO performs very well, contrary to the common belief that random allocation is associated with poor performance. With double buffering and no replication, RIO is said to support approximately the same number of streams as traditional striping in worst case. With replication, random allocation may outperform striping. Traditional striping outperforms RIO only for small amounts of buffer space and the time when RIO does not take advantage of using replication. But the difference is relatively small. In the system where maximum start up latency is a concern, random allocation has significantly superior performance than striping. [6] The main reason why randomized allocation is comparable to round-robin one is due to the absorption of the variances of the disk service time with increased I/O size, which can take advantage of utilizing the contiguous I/O operation.

It is expected that rapidly changing disk technology will con-

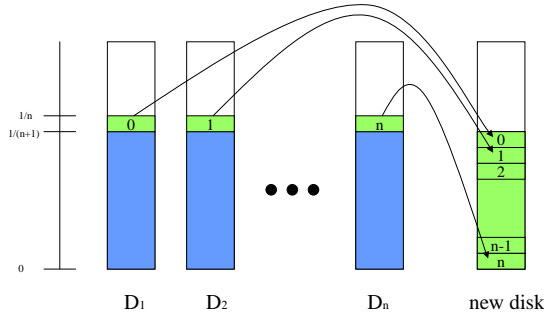


Figure 5: Cut-and-paste based block-to-disk mapping and migration.

tinue to validate this argument.

2.2.2 Random Striping

Random striping¹ scheme, termed in [7], is to combine replication and randomized allocation to randomly place the segment of each content on the striped disk set. The objects are divided into several contiguous segments. Each segment is copied several times and the copies are allocated on randomly chosen available free disks. Even consecutive segments are placed completely independent of each other.

Simulation report says that single copy of the data in random striping may result in an enormous fraction of hiccups of the continuous display at any load. Random striping with two copies is said to suffer a negligible number of hiccups at the highest load they simulated. [7] This result is disappointing in that randomized allocation requires at least one more copies in order to provide hiccup-free retrieval and display, which will result in significant space wastage. It is also contradictory to the simulation result done in Section 2.2.1. The only difference is the assumption on the buffering mechanism: Randomized I/O uses double buffering, whereas random striping does not consider any buffering. Currently, there is no simulation or experimental result to measure the buffering effect in randomized environment. We conjecture that more buffering, absorbing the deadline misses, may satisfy both contradictory results.

2.2.3 Cut-and-paste

Brinkmann et al. [8] investigates the problem of evenly distributing and efficiently locating data in dynamically changing distributed storage environments such as Storage Area Networks (SAN). It is also based on randomized allocation strategy to place the data across the disks. It assumes that there exists a high-quality hash function to assign all the data blocks in the system into the uniformly distributed real number with high probability, which is mapped to a dedicated storage device.

The placement strategy called *cut-and-paste* cuts off the range $[\frac{1}{(n+1)}, \frac{1}{n}]$ from given n disks, and pastes them to a range $[0, \frac{1}{(n+1)}]$ for a newly added $(n+1)^{th}$ disk as shown in Figure 5. For disk removal, it uses reversing operation which moves all the blocks in disks that will be removed to the other disks.

Cut-and-paste method failed to find an efficient hash function. Fortunately, such a hash function is proposed in [19], which is discussed in Section 2.2.4.

Cut-and-paste approach also argues that the distribution among

¹Within this context, it has different meaning. It is different from randomized allocation nor randomized striping termed in previous section.

the disks is unique in every situation, *which is in doubt*. If the object size is smaller than the stripe set, randomness of the object can be preserved. For a large media object, other paper shows that the hash function, considered as well-randomized, does not preserve the randomness of data per round after several disk additions and removals. [19]

For supporting heterogeneous disk environments, it reduces the problem of placing data in non-uniform (heterogeneous) disks to that of placing data in homogeneous disks by evenly distributing the data across the disks until some of the disks are saturated². If some disks are saturated, all blocks that cannot be stored due to saturated disks are attempted to stripe in the re-configured stripe set which excludes saturated disks at previous allocation step. At each striping step, algorithm uses different independent hash function to avoid the correlation of the data locations of successive steps. However, this greedy policy does not take into consideration on placing data based on different workload, which results in load imbalance. Section 4 presents more intelligent data distribution methods across the heterogeneous disks.

There are several open issues :

- Is there any hash function which preserves the randomness of data after the disk scaling operations?
- If not, what is the upper bound of the number of scaling operations which guarantees to preserve the randomness?

2.2.4 SCALING Disks for Data Arranged Randomly (SCADDAR)

The SCADDAR algorithm [19] assumes that data blocks are equal in size and randomly distributed across all storage devices in a system. The random distribution has great advantages when data needs to be reorganized: blocks to migrate can randomly chosen and move, resulting in a new random distribution after the reorganization. The core of the algorithm lies in its capability of keeping track of the location of the blocks after multiple reorganizations, such that efficient access is still guaranteed, and minimizing the amount of data to be moved. For example, to add one disk to a four disk storage system, only 20% of the data must be moved. However, SCADDAR optimizes for a single operation termed a *disk scaling operation*: either disk additions or disk removals.

With pseudo-random placement, for each block a random number X is generated and the block is placed on disk $(X \bmod N)$, where N is the total number of disks. [19] When adding or removing disks, one approach is to redistribute each block to disk $(X \bmod N_i)$, where N_i is the new number of disks. [19] Unfortunately, this approach may redistribute all the blocks to new locations. In order to minimize the data movement, SCADDAR extends above `modulo` approach by introducing `REMAP` function. `REMAP` computes new random number (X_{j+1}) of any block at $(j+1)^{th}$ disk scaling operation by using old random number (X_j) of that block calculated at j^{th} operation.

Let $q_{j+1} = (X_{j+1} \div N_{j+1})$ and $r_{j+1} = (X_{j+1} \bmod N_{j+1})$. `REMAP` for disk removal operation is

$$X_{j+1} = \begin{cases} q_j \times N_{j+1} + new(r_j) & : \text{if } r_j \text{ not removed} \\ q_j & : \text{otherwise} \end{cases}$$

`REMAP` for disk additions is

$$X_{j+1} = \begin{cases} q_j - (q_j \bmod N_{j+1}) + r_j & : \text{if } (q_j \bmod N_{j+1}) < N_j \\ q_j & : \text{otherwise} \end{cases}$$

As shown in above equations, computing the new location is decided and migrated only when certain conditions are satisfied.

²there is no available free space

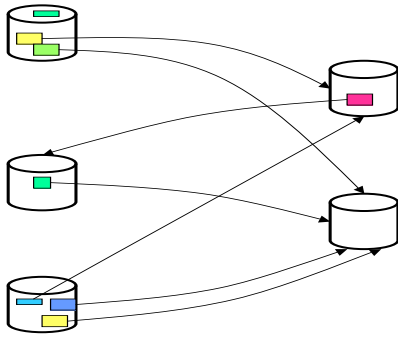


Figure 6: Example of Multi-graph representing data migration.

Otherwise, the data won't be moved, which reduces significant amount of data movement.

This pseudo-random function, however, is said not to preserve the randomness after several disk scaling operations. True randomized hash function which preserves its randomness of data has not been known so far. In the SCADDAR paper, sequences or combinations of operations are not dealt with, which motivates the Disk Replacement Problem (DRP). [4]

The SCADDAR algorithm is applied to the Yima [18] server system which implemented eight disk nodes support and experimented in the real world successfully.

3. DATA MIGRATION

The *data migration* problem is the problem of computing an efficient schedule to move data stored on storage devices in a network from one to another. [9, 11] The performance of data migration schedule is estimated to measure the elapsed time to complete the migration task as to its plan. In other words, the performance is to find a migration schedule using the minimum number of rounds to minimize the total amount of time to finish the schedule. The performance of large storage systems depends critically on having an assignment of data to storage devices that balances the load across devices or that optimizes a more complex cost function. Unfortunately, the optimal data layout changes over time when the workload for the storage requests may change, or when storage devices are added, removed, or replaced. Consequently, it is common to compute a new layout of data based on newly predicted workloads and storage specifications. Once the new assignment is computed, the data must be migrated from its old configuration to its new configuration.

All the following papers on data migration assume that the size of data objects are equal and the amount of time to move the data from one disk to another disk is same.

3.1 Bypass Migration

In [9], it suggests some algorithms to find a efficient migration plan in the minimum amount of time. This problem for networks of arbitrary topology is NP-complete even if all objects are the same size and each storage device has only one object that must be moved off of it. It can be viewed as a directed multi-graph $G = (V, E)$ without self-loops where each vertex corresponds to a storage device, and each directed edge $(u, v) \in E$ represents an object that must be moved from storage device u in the initial configuration to storage device v in the final configuration. [9](see Figure 6.)

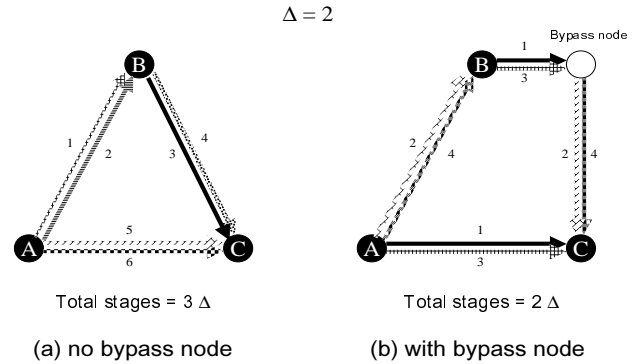


Figure 7: Examples of the number of edge color with no bypass node and with bypass node.

Above multigraph problem turns out to be the *multigraph edge coloring problem*, which is, of course, NP-complete. In [9], the bypass node, with no degree at the graph, is introduced to send an object to a storage device other than its final destination, assuming it would reduce the number of stages in the migration plan. A bypass node is an extra storage device that can be used to store objects temporarily. After the migration, temporary objects stored in the bypass node is removed.

For example, Figure 7 shows two data migration graphs whose lower bounds of edge color are identical ($\Delta=2$). Node A moves two items to B and C , respectively. Node B moves two items to C . Figure 7(a) shows the original data migration graph; Figure 7(b) is a modified multigraph with a bypass node³. Because Figure 7(a) performs a single transaction per stage, it requires 6 stages to complete the migration. But Figure 7(b) has two sets of parallel transfer operations (one set of (A to B) and (bypass node to C); the other set of (A to C) and (B to bypass node)), which finishes the migration at 4 stages. As illustrated, newly allocated bypass nodes increase the parallelism.

Assuming Δ be the lower bound to complete migration, they show that migration will be completed at most $2\lceil\frac{\Delta}{2}\rceil$ stages using at most $\frac{n}{3}$ where n is the number of vertices in G when there are no capacity constraints on the storage devices. Under the constrained capacity environments, it remains unclear how much harder this problem is than standard edge coloring, but provides at most $4\lceil\frac{\Delta}{4}\rceil$ algorithm with at most $\frac{n}{3}$ bypass nodes, or at most $6\lceil\frac{\Delta}{4}\rceil$ colors without bypass nodes.

3.2 Cloning Based Migration

Khuller et al. [11] argues that new copies may be dynamically created to handle high demand for popular objects. The maximum degree (Δ) of a node might be a lower bound on the number of stages that are required, since in each stage at most one transfer operation is involved. However, copying operations defer this assumption. For example, Figure 8(a) illustrates a particular operation, drawing a data migration example where original object in one disk is copied to three different disks without removing the original one. From this graph, it requires at least three edge colors to complete the task. However, as shown in Figure 8(b), copied object may be reused at later stage, which reduces the degree by 2. Migration graph with multiple copies reduces to a sequence of data migration graphs with no cloning. Therefore, data migration problem with cloning is harder to solve than data migration problem without cloning.

³a node filled with white circle

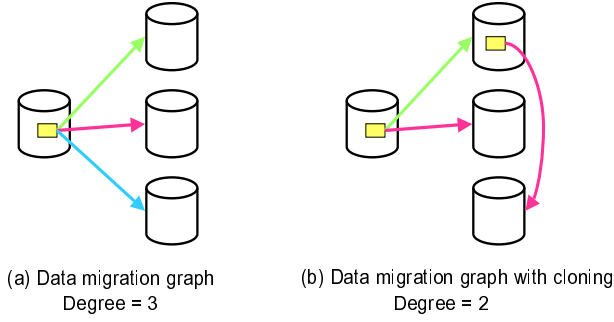


Figure 8: Example of Transition from data migration graph without cloning to data migration graph with cloning where max degree is 2.

Following data migration algorithm can be simply described as follows: There are N disks and Δ data items. Each item i are stored in a subset of disks S_i (**sources**). And all disks in D_i want to obtain i (**destinations**). After a disk in D_i receives object i , it can be a source of item i for other disks in D_i that have not received the item as yet. [11] Define β_j as the number of different sets D_i that a disk j belongs to, and β as an upper bound on the number of items a disk may need ($=\max_{j=1,\dots,N}\beta_j$). In other words, β is a lower bound on the optimal number of stages. After choosing optimal source nodes for each object individually, the algorithm divides each destination group, namely D_i , into two groups – one group having $|D_i| \geq \beta$; the other group having $|D_i| < \beta$ – to draw a migration graph in a separate way. After merging two separate transfer graphs, it calculates the number of edge colors of merged graph, which provides the upper bound on the number of stages required to ensure that each disk in D_j gets item j .

Algorithm 1 Data Migration Algorithm with Cloning

- 1: For each object i , choose a unique source from S_i where minimizes the maximum number of objects that a disk may be a source or destination for.
 - 2: **if** $|D_i| \geq \beta$ **then**
 - 3: For each object i , select a disjoint collection of subsets G_i where $G_i \subseteq D_i$ and $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$.
 - 4: Create a directed data transfer graph where each disk is a vertex and edges drawn from disks in G_i to $(\beta - 1) \lfloor \frac{|D_i|}{\beta} \rfloor$ disks in $\frac{D_i}{G_i}$ such that the out-degree of each source vertex at most $(\beta - 1)$ and the in-degree of each destination is 1.
 - 5: Redefine D_i as the set of disks whose degree is 0 in the transfer graph.
 - 6: **else**
 - 7: Repeat Step 1 with a newly defined D_i to find a new source s_i .
 - 8: Add a directed edge from the new source of object i to all disks in $\frac{D_i}{\{s_i\}}$.
 - 9: **end if**
 - 10: Find a edge coloring of the transfer graph.
-

Above algorithm is a polynomial-time approximation algorithm, whose worst case bound is 9.5-approximation.

3.3 Disk Replacement Problem (DRP)

The *Disk Replacement Problem* (DRP), a subset problem of data migration problem, is the challenge of finding a sequence of disk drives removals and additions to obtain a final, target storage system while minimizing the data migration cost. [4] The migration cost may depend on the requirements of the application and could either be the total amount of data moved during the disk replacement operation, or the total elapsed time to complete the task.

When the system has enough empty disk slots to add all new disks the operation is termed *unbounded*. Otherwise, it would be *bounded*. DRP model is proven to be a variation of the single-source shortest path problem. Thus, the upper bound of the complexity of finding the optimal sequence is polynomial. For unbounded disk scaling requests, they proposed *ABBD* and *ABD* algorithms. *ABBD* is the native implementation of scaling sequences with two atomic operations - disk additions and disk removals. The shortest sequence to minimize both the space and time cost is to add all new disks first and then to remove all disks to be deleted. After the installation or de-installation of the disks, data would be migrated from old disk configuration to new disk configuration to balance the load. The *ABD* algorithm improves upon *ABBD* by combing two independent data re-balancing steps into one. For bounded disk scaling requests, an exhaustive search algorithm based on a divide-and-conquer approach, termed *D&C*, is proposed to minimize both the time and space costs. To reduce its search space size, two space cost minimization heuristics, *D&C + EB* and *SPACE*, and two time cost minimization heuristics, *TIME* and *LMIN*, are suggested and simulated. The *D&C + EB* algorithm uses the *EB* component only when the system has at least one empty disk slot and the number of new disks is smaller than or equal to the number of disks to be removed. The *SPACE* algorithm finds the optimal data moving sequence in linear time. *TIME* reduces the *D&C* search space by filling all empty disk slots with new disks at once and then running the *D&C* algorithm. Finally, scaling sequences found by *LMIN* are not guaranteed to be optimal but the search space is again reduced compared with *TIME*. All the algorithms except *LMIN* and *ABBD* are always guaranteed to answer the optimal solutions for minimizing the space cost or the time cost.

Consider the following example. A storage system consists of ten disk drives. Each disk stores 100 GB of data; its bandwidth is 20 MB/s; and the data is evenly balanced across all the devices. A disk administrator wants to replace two old disks in the current configuration and also add ten new devices to increase the total storage space. At the end, the data should be load balanced again across all twenty disk drives. A naive storage administrator determines the following sequence: copy the data of the two disks to be replaced onto the other eight disk drives, and then remove the two old disks, add two new ones and restore the data. Finally, he adds ten disks and distribute the data evenly across all the disks. This scenario requires 900 GB data movement. If you take the following scenario, data will move 600 GB – 33 % improvement. In terms of time cost, while the naive approach takes 3.47 hours, optimal solution takes only 2.08 hours. It also implies that disk replacement method proposed at HP AutoRAID as discussed in Section 2.1.3 is incorrect.

It is also proven that multiple disk additions or removals are preferred over a single disk addition or removal mathematically. This means that it would be advantageous to perform data re-organization periodically after "batching" enough disk adds and removes together, rather than on an individual basis.

4. HETEROGENEITY SUPPORT

A disk drive is characterized by bandwidth, space and its com-

bination. Rapid advances of modern disk manufacturing technologies enable much larger capacity and slightly faster disk service time, making heterogeneous disk drives coexist in a system. At the time of object placement to disks, data lay-out should be carefully determined to avoid load imbalance across the disks and to maximize the utilization of both bandwidth and space. Wrong decision to place frequently accessed objects on slower disks would lead to significant performance degradation.

4.1 Heterogeneous Declustering

Declustering is a strategy to enhance I/O parallelism by distributing the data among n parallel disks so that data which is likely to be requested together by a query is allocated to different disks. In [12], adaptive declusterization methods which works efficiently in heterogeneous disks and server systems are proposed.

In case where all disks are infinite in capacity, optimal declustering scheme that minimizes the response time for the query requests is to have the amount of data stored on each disk (a_i) be proportional to the bandwidth of each disk (B_i), $a_i = \text{data size} \times \frac{B_i}{\sum_{k=1}^n B_k}$. If disks are finite in capacity, linearity between bandwidth and space becomes invalid.

Assume we have three disks with 1 GB available space; average transfer rates of each disk are 3 MB/s, 2 MB/s, and 1 MB/s. And we need to store the data of 2.5 GB in the system. Above optimal declustering scheme would decluster data in a 3:2:1 ratio (proportional to the speed of three disks) regardless of the available disk spaces. If data is declustered according to the bandwidth, each disk will store 1GB, 833.33 MB, 416.67 MB of data respectively. Because of 1 GB limitation of each disk, the total allocated amount is short of 250 MB and maximum bandwidth is not achievable. The declustering paper [12] suggests Max-Bandwidth algorithm that places 1 GB on disk 1, 1 GB on disk 2, and 0.5 GB on disk 3, resulting in a maximized bandwidth of 5 MB/s.

A dataset of C Bytes on a system of n disks with each disk i having a capacity of C_i and a transfer rate B_i , such that the overall bandwidth B of the system is maximized. Let

$$T = \frac{C}{B}$$

$$f(i, T) = \min(T \times B_i, C_i)$$

$$g(T) = \sum_{i=1}^n f(i, T)$$

$f(i, T)$ indicates the maximum amount of data that can be stored on disk i , subject to the retrieval time constraint of T . $g(T)$ is the total volume of data that can be stored on the system subject to the time constraint T . Finally, the maximal bandwidth, B_{opt} should satisfy the following constraint: $C = g(T_{opt})$ where $T_{opt} = \frac{C}{B_{opt}}$.

Algorithm 2 Max-Bandwidth Algorithm

- 1: Compute $B_{max} = \sum_{i=1}^n B_i$.
- 2: Do a binary search between 0 and B_{max} to find a value B that satisfies $0 \leq |g(T) - C| \leq \epsilon$ where ϵ is a acceptable error.
- 3: Store the i^{th} disks with $f(i, \frac{C}{B})$.

The complexity of this Algorithm is $O(n \times \log_2 \frac{B_{max}}{\text{error margin}})$. Because its running time to compute the optimal bandwidth is less than one second, optimal solution can be obtainable immediately and thus applicable to various heterogeneous disk subsystems. However, this algorithm does not take care of non real-time

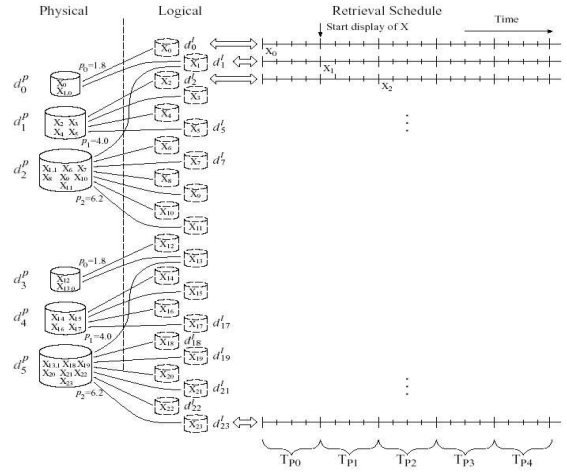


Figure 9: Disk Merging.

objects such as regular file because the assumption on T cannot be made.

4.2 BSR

Two parameters, the bandwidth (the number of objects that can be served) and the space (the number of objects that is stored on the disk), characterizes heterogeneity nature of a disk. The Bandwidth to Space Ratio (BSR) value of a disk represents the degree of similarity with other disks in terms of disk characteristics. The BSR policy is proposed to provide load balance across the disks and maximize the utilization of both bandwidth and space in heterogeneous disk storage systems. [13] The idea of this policy is to place the video objects with higher popularity on a faster disk. Thus, the expected load of the object and its required bandwidth for creating the replica are primary considerations.

The algorithm consists of four steps:

Algorithm 3 BSR Policy

- 1: Decide whether it requires additional copies of the data.
- 2: If required, select additional disks on which new copies are placed.
- 3: Allocate the expected load to all selected disks to match the BSR values of the devices.
- 4: Delete the some of the unused or infrequently accessed existing copies when the policy cannot place all of the expected load.

This straight-forward greedy policy assumes that the expected load on video objects is predictable and changes infrequently. This assumption, however, is hard to achieve because of a large uncertainty in the future demands for video objects.

4.3 Disk Merging

Disk merging technique suggested by [14] separates the concept of logical disk from the physical disks. It is designed to support an arbitrary mix of physical disks and to guarantee a continuous display of each video object.

For example, as shown in Figure 9, six physical disk drives (d_0, \dots, d_5) are configured such that the physical disk $d_0(d_3)$ supports 1.8 logical disks, $d_1(d_4)$ supports 4.0, and $d_2(d_5)$ supports 6.2. The fractions of logical disks can be combined to form additional logical disks. Therefore, the total number of logical disks in Figure 9 can add up to 24. The way to find the mapping

from physical disk to logical disk is initially determined by the bandwidth ratio, which is later fined-tuned through the iterative re-computation.

As expected, there are a lot of space wastage if the BSR of physical disks are totally different because the allocation size of a system is limited by the size of the smallest logical disk⁴. Decreasing the number of logical disks on some of the physical disks to maximize the disk space results in a wastage of the disk bandwidth. To find an optimal configuration to minimize the unused fractions of logical disks, which has a proportional to maximize the utilization of both bandwidth and space of physical disks, is exponential by varying the initial value of p_0 , and hence an exhaustive search is impractical in the real world. As a consequence, this technique is useful when the BSR values of the disks are similar.

5. DIRECTION OF FUTURE RESEARCH

There are many research issues in distributed storage systems. Data migration problem has a framework to be extended as a generalized network problem such as multi-source message distribution and delivery. Content Distribution Network (CDN) is another application to use data migration techniques. The CDN, based on Distributed Hash Table (DHT) technique, may distribute the media content by segmenting it into successive data blocks and placing them on distributed disks randomly. The deployment of many data migration approaches into the real network has not been studied yet.

Another interesting topic is to support the storage device whose failure rate is much higher than traditional disk based storage system. In Peer-to-Peer network, each peer node may have heterogeneous storage space, which is being served for object sharing, and joins/leaves frequently. In such an environment, assumption on low failure rate of each system is not valid any more. Therefore, intelligent data replication and migration to maintain higher availability and performance is also necessary.

In data allocation area, widening the stripe size extremely also has an interesting feature. DHT based random allocation is such an example, where the stripe size will change rapidly as well.

All these issues are expected to be applicable to both Storage Area Network and Network Attached Storage architecture (NAS) in the real world.

6. REFERENCES

- [1] Serge Abiteboul et al. "The Lowell Database Research Self-Assessment Meeting Report," Lowell Massachusetts, May 2003.
- [2] David A. Patterson, Garth Gibson, and Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *SIGMOD*, 1988, pp. 109–116.
- [3] Cheng-Fu Chou, Leana Golubchik, and John C. S. Lui, "Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication," in *International Conference on Distributed Computing Systems*, 2000, pp. 64–71.
- [4] Beomjoo Seo and Roger Zimmermann, "Efficient Disk Replacement and Data Migration Algorithms for Large Disk Subsystems," University of Southern California, 2003.
- [5] Steven Berson, Shahram Ghandeharizadeh, Richard Muntz, and Xiangyu Ju, "Staggered Striping in Multimedia Information Systems," in *SIGMOD*, 1994, pp. 79–90.
- [6] Jose Renato Santos, Richard R. Muntz, and Berthier A. Ribeiro-Neto, "Comparing random data allocation and data striping in multimedia servers," in *Measurement and Modeling of Computer Systems*, 2000, pp. 44–55.
- [7] J. Alemany and J. S. Thathachar, "Random Striping News on Demand Servers," Tech. Rep. TR-97-02-02, University of Washington, 1997.
- [8] Andre Brinkmann, Kay Salzwedel, and Christian Scheideler, "Efficient, Distributed Data Placement Strategies for Storage Area Networks (extended abstract)," in *ACM Symposium on Parallel Algorithms and Architectures*, 2000, pp. 119–128.
- [9] Joseph Hall, Jason D. Hartline, Anna R. Karlin, Jared Saia, and John Wilkes, "On Algorithms for Efficient Data Migration," in *Symposium on Discrete Algorithms*, 2001, pp. 620–629.
- [10] Eric Anderson, Joseph Hall, Jason D. Hartline, Michael Hobbs, Anna R. Karlin, Jared Saia, Ram Swaminathan, and John Wilkes, "An Experimental Study of Data Migration Algorithms," in *Proceedings of the 5th International Workshop on Algorithm Engineering*. 2001, pp. 145–158, Springer-Verlag.
- [11] Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan, "Algorithms for Data Migration with Cloning," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2003, pp. 27–36, ACM Press.
- [12] Ling Tony Chen, Doron Rotem, and Sridhar Seshadri, "Declustering Databases on Heterogeneous Disk Systems," in *Proceedings of the 21st International Conference on Very Large Data Bases*, September 1995, pp. 110–121.
- [13] A. Dan and D. Sitaram, "An Online Video Placement Policy based on Bandwidth to Space Ratio (bsr)," in *ACM SIGMOD International Conference on Management of Data*, May 1995, pp. 376–385.
- [14] Roger Zimmermann and Shahram Ghandeharizadeh, "Continuous Display using Heterogeneous Disk-subsystems," in *Proceedings of the fifth ACM international conference on Multimedia*. 1997, pp. 227–238, ACM Press.
- [15] Philip S. Yu, Mon-Song Chen, and Dilip D. Kandlur, "Grouped sweeping scheduling for dasdbased multimedia storage management," *ACM Multimedia Systems*.
- [16] Gregory R. Ganger, Bruce L. Worthington, Robert Y. Hou, and Yale N. Patt, "Disk Subsystem Load Balancing: Disk Striping vs. Conventional Data Placement," in *Proceedings of the Hawaii International Conference on System Sciences*, 1993, pp. 40–49.
- [17] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan, "The HP AutoRAID hierarchical storage system," in *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, Hai Jin, Toni Cortes, and Rajkumar Buyya, Eds., pp. 90–106. IEEE Computer Society Press and Wiley, New York, NY, 2001.
- [18] Roger Zimmermann, Kun Fu, Cyrus Shahabi, Didi Shu-Yuen Yao, and Hong Zhu, "Yima: Design and evaluation of a streaming media system for residential broadband services," *Lecture Notes in Computer Science*, vol. 2209, pp. 116–??, 2001.
- [19] Ashish Goel, Cyrus Shahabi, Didi Shu-Yuen Yao, and Roger Zimmermann, "SCADDAR: An Efficient Randomized Technique to Reorganize Continuous Media Blocks," in *ICDE*, 2002.

⁴the size of the smallest logical disk unit is termed p_0