Distributing Orthogonal Redundancy on Adaptive Disk Arrays

J. L. Gonzalez {DAC-UPC},Toni Cortes{DAC-UPC,BSC}
joseluig@ac.upc.es, toni@ac.upc.es

[1] Department of Computer Architecture, Technical University of Catalonia(DAC-UPC)
[2] Barcelona Supercomputing Center(BSC)

Abstract. When upgrading storage systems, the key is migrating data from old storage subsystems to the new ones for achieving a data layout able to deliver high performance I/O, increased capacity and strong data availability while preserving the effectiveness of its location method.
However, achieving such data layout is not trivial when handling a redundancy scheme because the migration algorithm must guarantee both data and redundancy will not be allocated on the same disk. The Orthogonal redundancy for instance delivers strong data availability for distributed disk arrays but this scheme is basically focused on homogeneous and static environments and a technique that moves overall data layout called re-striping is applied when upgrading it.
This paper presents a deterministic placement approach for distributing orthogonal redundancy on distributed heterogeneous disk arrays, which is able to adapt on-line the storage system to the capacity/performance demands by only moving a fraction of data layout.
The evaluation reveals that our proposal achieve data layouts delivering an improved performance and increased capacity while keeping the effectiveness of the redundancy scheme even after several migrations. Finally, it keeps the complexity of the data management at an acceptable level.

1 Introduction

Delivering improved performance I/O, increased capacity and strong data availability, have been the traditional aims when designing storage systems

Scalability of storage systems is as important, or even more important to current applications than the aims above-mentioned mainly because of the constant growth of new data (at an annual rate of 30% [1] and even 50% for several applications [2]) and the rapid decline in the cost of storage per GByte [3], which have led to an increased interest in storage systems able to upgrade their capacity online.

Since applications such as scientific or database are particularly sensitive to storage performance because of their imposing I/O requirements (in some cases can account for between 20 to 40 percent of total execution time [4]), they are requiring storage systems capable of upgrading their bandwidth in order to deliver fast service times.

The upgrade of storage systems means managing new disks that have not been contemplated in the data layout that is used for locating data and redundancy blocks. The key thus is performing rearrangements on the data layout by migrating blocks from old storage subsystems to the new ones in order to make use of the new capacity and bandwidth delivered by the new storage subsystems.

Nevertheless, the following challenges arise when periodically upgrading the data layout of storage systems:

The effectiveness of the redundancy scheme: the storage systems must to guarantee not allocating data and redundancy on the same disk even after several upgrade processes.

The management of data: The storage system must be able to preserve a simple, efficient and compact method for locating data and redundancy blocks even when managing large volumes of information.

The heterogeneity management: the periodical upgrade of storage systems results in heterogeneous storage environments because a constant improvement in disk capabilities has been observed year after year [3] [5]. Therefore, the addition of new disks requires a compromise between taking advantage of disks with higher capabilities [6] and avoiding wasting disk capacity and/or performance.

The disk arrays known as RAID systems [7] have become the most popular storage technique in the market server for meeting both capacity and performance requirements while schemes based on redundancy have been key for achieving data availability even during disk failures [7] [8] [9] [10].

However, RAID systems are not able to face up to above-mentioned challenges because they are essentially homogeneous and non-adaptive

For instance, RAIDX [11] is quite popular in cluster environments because it allows all distributed local disks to be integrated as a distributed-software redundant array of independent disks (ds-RAID) with a single I/O space. Its main advantage comes from its Orthogonal Striping and Mirroring (OSM) redundancy scheme where the bandwidth is enhanced with distributed striping across local and remote disks while its redundancy are writing in a sequential manner [8] .

Nevertheless, RAIDX is not able to face up to the capacity and performance demands in an efficient manner when new disks are added to this system because one completely rearrangement of its data layout is required for guarantying the effectiveness of its redundancy scheme.

As we can see, one of the greatest challenges is to design storage systems that can handle capacity demands by adapting new storage subsystems yet achieve high performance, strong data availability, and simple management when faced with huge volumes of information.

This paper presents an adaptive and deterministic approach for distributing orthogonal redundancy on a set of heterogeneous disks, which is also able to adapt the data layout of disk arrays to the addition of new storage subsystems by using a upgrade strategy based on the segments movement.

This strategy rearranges only a fraction of the data layout and achieves an improved I/O performance and capacity while preserving the effectiveness of redundancy scheme even after several migrations.

Although this proposal has been designed to deliver different orthogonal redundancy schemes even at the same time, we are focusing our proposal on the OSM of RAIDX because it is the scheme imposing most restrictions when performing an upgrade process.

In addition, there is not currently available an efficient method for distributing OSM in dynamically changing storage environments.

By simulating several scenarios and application workloads, the performance of our proposal has been measured before and after adding storage subsystems, we also evaluate the effect of disk heterogeneity and we analyze the complexity in data management after several migrations.

The evaluation reveals that although our rearrangement scheme only moves between 20 and 40 % data, the resultant data layout yields a similar performance to traditional method of re-striping, which rearranges the 100 % of the data layout. It also reveals when a set of heterogeneous disks are used the data layout produced by the rearrangement scheme is able to deliver even better performance and increased capacity in the same comparison.

In addition, the heterogeneity management only needs few KB as metadata. Finally, it keeps the complexity of the data management at an acceptable level.

2 Related Work

The proposals for Adaptive storage systems have mainly been focused on:

Distributing data blocks in a random manner: Hash tables are defined for handling the metadata when the system has been upgraded, which allows rearranging only a fraction of the data layout when upgrading storage systems [12] [13]. However, it requires huge efforts for keeping benefits such as high performance, balanced load and efficiency data locating after several upgrade processes.

Defining a solution at file system level: This solution can choose between using either sequential or random data placement method while using name / metadata servers for locating data [14] [15] [16]. The performance for this case depends on algorithms for balancing load per disk. Nevertheless, the abovementioned solutions are focused on the optimization of either capacity or performance not data availability.

The current research for providing data availability in constantly changing environments is only available for schemes based on mirrors [17], chained declustering [18] and replicas [19] not for orthogonal striping.

3 An Adaptive Redundancy Placement Method

Our placement method makes use of two kinds of algorithms:

- *Algorithm for Handling Orthogonal Redundancy on Heterogeneous Disks.*
- *Algorithm for adding new storage subsystems in distributed disk arrays.*

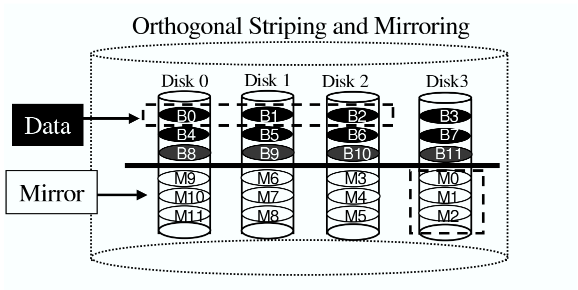


Fig. 1. *An example of a OSM in a RAIDX system*

3.1 An Algorithm for Handling Orthogonal Redundancy on Heterogeneous Disk Arrays

An array of mid-ranged RAID is considered as orthogonal one [10] whenever it can deliver data availability not only when a disk fails but also even when a RAID system fails.

We will use the term **SS** in the rest of this paper for identifying a Storage Subsystem which means a SS can be either a single disk or RAID system. We also will use the term **matrix** for identifying an orthogonal disk array.

Orthogonal Striping and Mirroring Overview In the OSM scheme each SS is logically divided into two parts: the top part for data and the bottom one for mirrors. It is focused on using as much bandwidth as possible in the top part by distributing stripes on **C** SS such as RAID0 does while the mirroring is performed in sequential manner in vertical groups of **G** blocks in bottom part (where C represents the number of SSs of the array and G = C-1).

Figure 1 shows an example of a RAIDX using the OSM for distributing blocks and mirrors on C=4 and G = 3. B0 means Block 0 and M0 mirror 0.

As we can see, this method has been designed for homogeneous environments handled by a fixed data layout, which involves the following two restrictions:

1. OSM assumes all SS having the same number of disks.

2. OSM requires the distribution of G stripes of data on the top part of the array for delivering a balanced load per disk.

In figure 1 three stripes have been distributed in top part in order to deliver a uniform load per disk.

The first assumption avoids the expansion of the system by the addition of SS with different number of disks and the second the usage of SS with different capabilities.

Distributing OSM Redundancy on Heterogeneous SS We use *heterogeneous repetition patterns* [6] for distributing data blocks because of their capability to capture the disk's behave by assigning a utilization factor per SS or *UF*. The *UF* parameter is a value between 0 and 1 that defines the way in which the algorithm will distribute both data and redundancy on a SS. For example, in

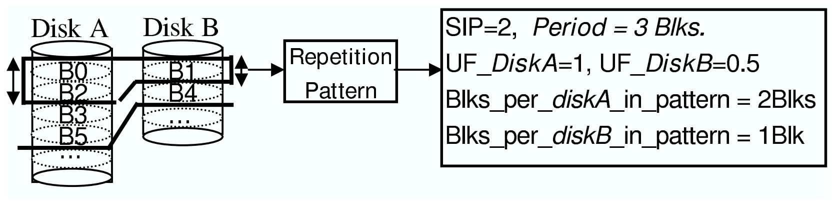
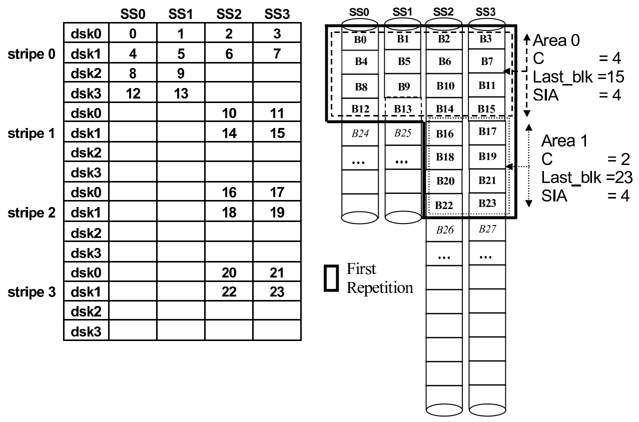

Disk A
Disk B
B0
B2
B3
B5
...
B1
B4
...
Repetition Pattern
SIP=2, *Period = 3 Blks.*
UF_*DiskA*=1, UF_*DiskB*=0.5
Blks_per_*diskA*_in_pattern = 2Blks
Blks_per_*diskB*_in_pattern = 1Blk

Fig. 2. *An example of a repetition pattern*



		SS0	SS1	SS2	SS3
	dsk0	0	1	2	3
stripe 0	dsk1	4	5	6	7
	dsk2	8	9		
	dsk3	12	13		
	dsk0			10	11
stripe 1	dsk1			14	15
	dsk2				
	dsk3				
	dsk0			16	17
stripe 2	dsk1			18	19
	dsk2				
	dsk3				
	dsk0			20	21
stripe 3	dsk1			22	23
	dsk2				
	dsk3				

Fig. 3. *The block distribution per disks on the matrix (left) and the usage of heterogeneity areas (right)*

a matrix of two SS: *SS_A* of 540GB and *SS_B* of 250GB the administrator would like to make use twice of the *SS_A* , which means the algorithm should assign it *UF*=1 while UF=0.5 to *SS_B*. This results in the algorithm distributing two blocks on *SS_A* per only one on *SS_B*. See this example in figure 2.

The algorithm distributes enough blocks for achieving the *UF* of each SS, which results in the definition of a heterogeneous pattern. Once it has been defined it is repeated until saturating all disks.

Handling Orthogonal Systems of different number of SSs When the SSs are RAID systems and they have different number of disks, our data placement method applies a UF per RAID system instead of per disk for distributing data and redundancy blocks.

Figure 3 shows a matrix of four SS: *SS_0* and *SS_1* of 250GB with four disks each while *SS_2* and *3* of 540GB with two disks each. In this figure, all disks of all SSs have been used according its *UF* (left).

Defining Heterogeneity Areas Different heterogeneity levels are handled according to the types of disk used in the matrix, thus the idea is to divide the

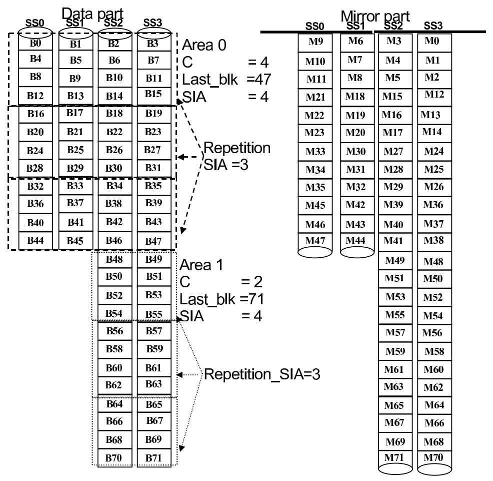


Fig. 4. *An example of how the algorithm adjusts the SIA of the heterogeneity areas*

heterogeneous pattern into as many areas as heterogeneity levels we are handling in that pattern. The areas works as a homogeneous RAID because they have its *C*, which means the algorithm can keep the effectiveness of the OSM redundancy. The parameter called *last_blk* indicates the limit of this area and the beginning of the next one if any. Figure 3 (right) shows how the pattern is divided into two areas: the area 0 with *C*=3 and area 1 with *C*=2; both areas having a *SIA* = 4. Where *SIA* represents the number of *Stripes In Area.*

The *area* structure is used when locating blocks within the areas of a pattern and it includes the following fields:

C_area: this field keeps the number of disks of that heterogeneity area.

Last_blk: this field keeps the number of blocks of that heterogeneity area.

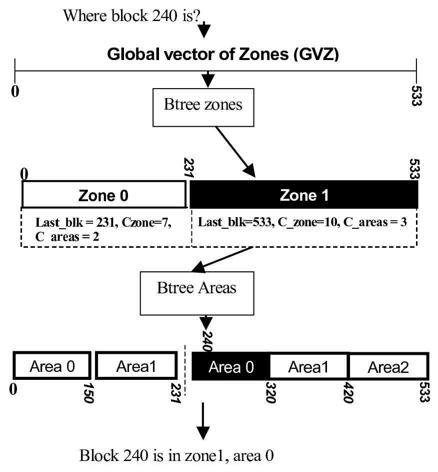
Dsk_per_area: this table has a size of C_area and allows locating the disks such as RAID system does.

Blks_in_pattern_per_area: This is a table of C_area size, which keeps the number of blocks per disk in this area. This table allows locating the block position into a disk.

The algorithm adjusts each area for guarantying a balanced load per SS both in top and bottom part of matrix by modifying the SIA value, which keeps the behave of OSM redundancy scheme for all heterogeneity areas. (See figure 4)

Locating blocks in Adaptive Disk Arrays Our adaptive placement method uses a Single Address of Space (SAS) called Global Vector of Zones (GVZ) [17], which has been modified in our proposal for locating blocks within heterogeneity areas.

GVZ handling Heterogeneity Areas The GVZ is a tree of zones growing zone by zone according to the capacity/performance demands, the intuitive idea is starting with zone 0 and when the system is upgraded with the addition of



Where block 240 is?
Global vector of Zones (GVZ)
0
533
Btree zones
0
231
533
Zone 0
Zone 1
Last_blk = 231, Czone=7, C areas = 2
Last_blk=533, C_zone=10, C_areas = 3
Btree Areas
240
Area 0
Area1
Area 0
Area1
Area2
0
150
231
320
420
533
Block 240 is in zone1, area 0

Fig. 5. *The algorithm uses B-trees for determining data location within GVZ*

new SS, the GVZ also is increased by zones able to allocate as many blocks as capacity delivered by the new SS.

In this approach a *zone* is big horizontal part of the matrix on which the blocks are striped as in a traditional RAID. As a *zone* is an abstraction used for allocating/locating blocks within the matrix by using a heterogeneous pattern, a zone can handle several heterogeneity areas.

Locating Blocks within Heterogeneity Areas: The algorithm must to determine the zone and area where the information was distributed by applying the following procedure:

1. The algorithm uses a b-tree for determining the zone or *chosen zone* where the information was allocated.
2. The algorithm determines the number of block into that zone (BZ). if *chosen zone = 0* then BZ=B Otherwise **BZ** = block(B)-(last_block(chosen zone-1)+1)
3. Once the BZ has been defined, the algorithm determines in which repetition of the pattern (module_p) that block has been allocated. Afterwards, it determines the Blk_pattern. Where:
   Blk_pattern= BZ-(module_p* last_block(chosen zone-1))
   module_p = BZ/last_block(chosen_zone-1)
4. The algorithm applies another b-tree using it for determining the area where Blk_pattern is.

Figure 5 shows an example when locating block 240 in a storage system handling two zones. As we can see, the access to the area where blocks have been allocated is quite simple and hierarchy.

Now the algorithm has to determine the position of the Blk_pattern into the chosen area or (**Blk_area**) by using the *areas table*, which has been full-filled in advance when the pattern of this area was designed.

The following formulas are used for locating the disks where the block and it redundancy have been allocated by our placement method:

Disk_Data = $dsk\_per\_area\_table[chosen\_area][(blk\_area\%areas\_table.C)]$

Disk_Redundancy = $dsk\_per\_area\_table[chosen\_area][areas\_table.G - Mirror]$

Pos_blk = $(module\_p * blks\_in\_pattern\_per\_dsk\_table[dsk\_per\_area[chosen\_area][Disk\_Data]])$

$+blks\_in\_pattern\_per\_area\_table[Disk\_Data] + (blk\_area/areas\_table.C)$

Redundancy_pos $= (module\_p$
$*blks\_in\_pattern\_per\_dsk\_table[dsk\_per\_area[chosen\_area][Disk\_Redundancy]])$
$+blks\_in\_pattern\_per\_area\_table[Disk\_Redundancy]$
$+(module\_area * areas\_table.G) + (blk\_area\%areas\_table.G)$

WHERE

Mirror $= mod(blk\_area, (areas\_table.G * areas\_table.C))$

Module_area $= blk\_area\%(areas\_table.G * areas\_table.C)$

3.2 Algorithm for Assimilating New SS in Distributed Disk Arrays

The following strategies are used for taking advantage of the space and bandwidth of the new SS.

1. Designing a new data layout including the new SS. The migration algorithm defines a data layout including the new SS added to the storage system (NSS in the rest of this paper). The migration algorithm delivers a trade-off between achieving balanced load per disk, improved performance, increased capacity and strong data availability and reducing the time employed when migrating data.

2. Reducing the spend time when migrating data Once the new data layout has been defined, the algorithm starts by calculating *IL*, which represents the Ideal Load that each disk should have after the migration process (new disks included):

IL = Old_Capacity/(Old_Capacity+Capacity_New_SS)
MC = (1 - IL)* Old_Capacity

IL delivers a % that is used for determining *MC* , which represents the amount of blocks that the algorithm must migrate.

3. A Migration Algorithm Based on Data Layout Segmentation and Zones Expansion: This algorithm performs a logical segmentation of zones that results in large chunks of data or redundancy that are called segments.

The intuitive idea is moving large segments of data instead of independent blocks in order to meet both IL and MC requirements.

As each segment consist of a huge number of blocks, the usage of segments reduces the effects of the migration process on data access because technically this big part of data layout only is changing to another SS. This guarantee, at least, that that big part of data layout still having the same behave and can be handled in similar manner in the new SS.

For example, the blocks of a file that have been allocated in adjacent manner in a SS will keep this behave in the new SS where they have been migrated. The same applies with redundancy blocks, which guarantees the effectiveness of almost all schemes based on striping. In the best case, this big part of the data layout (segment) will observe better services times or even more parallelism in its new position that in its old SS.

Defining the Segmentation Levels The algorithm defines the following types of segmentation:

Diagonal Segmentation: This segmentation divides the data layout into C_zone* C_zone chunks, which are used in a diagonal manner where C_zone is the number

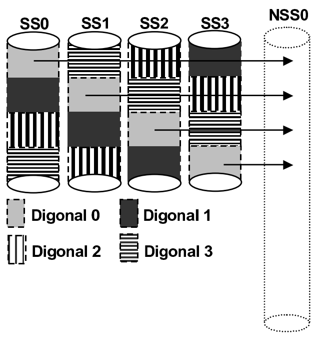


Fig. 6. *An example of a zone with Diagonal segmentation*

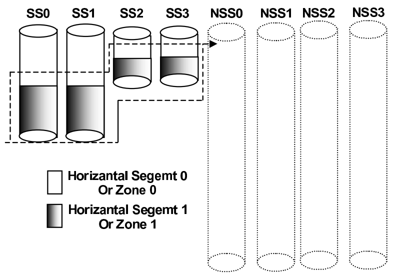


Fig. 7. *An example of a zone with Horizontal segmentation*

of disk used in the zone. Figure 6 shows how the algorithm performs a diagonal segmentation on zone 0. As we can see, the algorithm could migrate the diagonal 0 to the NSS0. This technique avoids migrating data and its redundancy to the same SS.

Horizontal Segmentation: This kind of segmentation divides the zone into two horizontal parts. The algorithm thus can choose any part for migrating it to the NSSs. This kind of segmentation produces a new zone for handling the migrated part. Figure 7 shows how the algorithm has performed a horizontal segmentation in zone 0. As we can see, the algorithm could migrate any horizontal segment to the New SSs.

Choosing Available Segments: The following restrictions defining the policies used for preserving the OSM effectiveness after upgrade process.

Diagonal segment: 1) Two diagonal segments cannot be migrated to the same NSS. 2) A diagonal segment that has been migrated can be migrated again.

Horizontal Segment: 1) This segment cannot be migrated to one single SS. 2) A horizontal segment can be migrated to another place only when the number of $C\_NSS => C\_zone$ where C_NSS represents the number of new SS added to the system.

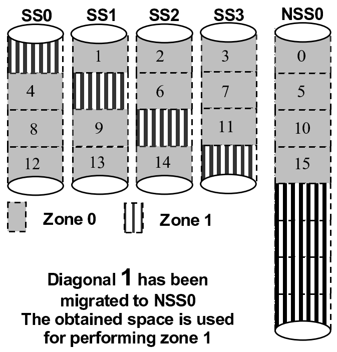

SS0
SS1
SS2
SS3
NSS0
1
2
3
0
4
6
7
5
8
9
11
10
12
13
14
15
Zone 0
Zone 1
Diagonal 1 has been migrated to NSS0
The obtained space is used for performing zone 1

Fig. 8. *First migration. The algorithm migrates one diagonal segment to the NSS0.*

Zone Extension: When the chosen segments do not meet the *IL* and *MC* requirements, the algorithm applies a zone extension. By performing a re-striping of that zone, the algorithm upgrades the *C_zone* value of the stripes with *C_zone + C_NSS*.

3. The Following algorithm is used for selecting available segments:

- If (C_storage_system>=NSS) then making horizontal segmentation.
  In this case, both the IL and MC parameters as well as the parallelism and redundancy requirements are easily achieved.
  - The algorithm calculates the first_blk_mig and all zones beyond this block, if any, are migrated as a single new zone.Where first_blk_mig = Old_Capacity - MC_AdaptiveZ.
- Otherwise
  - The algorithm calculates the load per SS that could be achieved by extending the last zone and asks If all SS has IL+(range* IL) or IL-(range* IL) then
    - The algorithm makes horizontal segmentation of last zone and performs a re-striping of that zone for achieving an approximation to IL

    .
  - Otherwise, the algorithm ask for zones with a diagonal segmentation. If any
    - if available diagonal segments of that zone >= NSS then Selecting NSS diagonal segments.
      - The algorithm calls to the algorithm defined for matching the available segments with MC and IL parameters
    - Otherwise, Selecting the available diagonal segments and looking for NSS-available diagonal segments of vertical segments.
    - The algorithm calls to the algorithm defined for matching the available segments with MC and IL parameters
- Otherwise, the algorithm chooses the zone with more SS and data blocks for performing diagonal segmentation. Afterwards, the algorithm selects C_NSS diagonal segments.
- The algorithm calls to the algorithm defined for matching the available segments with MC and IL parameters.

The following algorithm has been defined for matching the available segments with MC and IL parameters:

- The algorithm asks if the sum of capacity of chosen segments is < MC+(range*MC).
  - if (load per SS < IL load+(IL load*range)) and (load per SS < IL load-(IL load*range)) then The algorithm can migrate the selected segments.
  - Otherwise determines whether the best option is choosing vertical segments or expanding the last zone, once defined it the algorithm selects the segments and it can migrate them.
- Otherwise: The algorithm must migrate the selected segments.

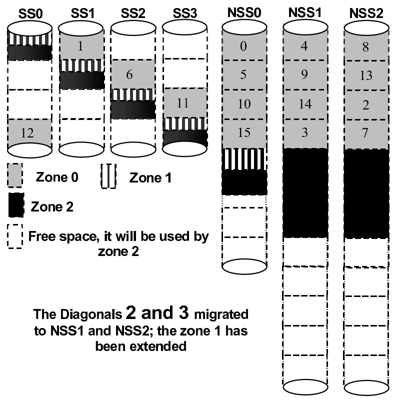

SS0
SS1
SS2
SS3
NSS0
NSS1
NSS2
1
6
11
12
0
5
10
15
4
9
14
3
8
13
2
7
Zone 0
Zone 1
Zone 2
Free space, it will be used by zone 2
The Diagonals 2 and 3 migrated to NSS1 and NSS2; the zone 1 has been extended

Fig. 9. *Second migration. The algorithm migrates diagonal 1 and 2 to NSS1 and NSS2. The algorithm also has extended the Zone 1.*

Figure 8 shows a matrix of 4 SS where one SS (NSS0) has been added to the matrix and the algorithm has decided to migrate diagonal 0 to NSS0.

Figure 9 shows the example from figure 8 but now two SS have been added to the matrix. In this figure, we can see how the algorithm has decided to migrate the two next diagonals to the NSS1 and NSS2. In addition, the algorithm has decided to perform a re-striping with the rest of MC in zone 1.

In the second migration, figure 8 represents the original data layout while figure 9 represents the data layout that the algorithm has designed with chosen segments.

Note that when the algorithm migrates all diagonals, it can decide whether to make a diagonal segmentation of another zone or make a re-striping of the last zone according to the MC and IL.

Migrating Blocks The algorithm starts the migration process for converting the original data layout to the final one.

We have already proposed an method for migrating data blocks during idle periods [20], [21] and [17]. This method has been designed for reducing the effects produced by the migration process by proposing a mechanism able to take gradually advantage of NSS bandwidth. The bandwidth of the NSSs is used for gradually reducing the overhead suffered by the upgrade operations until a gradual improvement in their services times is observed even during the migration process.

Locating Blocks When Using the Final Data Layout Since the algorithm has moved segments from SSs to NSSs, the algorithm easily can locate the new location of segments that have been migrated by using the indirection concept.

An indirection can be determined by using the *diagonal* structure, which has been full-filled during of segments selection and includes the following fields:

	Migration 1	**Migration 2**	**Migration 3**
Arrival	9 ms arrival time	7 ms arrival time	5 ms arrival time
(ON/OFF model)	500 ms OFF periods	400 ms OFF periods	300 ms OFF periods
% of read requests	70 Uniform distribution	(for all migrations)	
Request size	8Kb Poison distribution	(for all migrations)	
Request location	Uniform distribution 35% sequential	(for all migrations)	

Table 1. *Specifications of workload used in each migration*

C_diagonal: represents the number of diagonals that have been migrated.

S_diagonal: represents the size of the diagonal.

N_diagonal: is a table that indicates the new SS where a diagonal segment has been migrated.

4 Methodology

As our proposal is able to guarantee the effectiveness of redundancy schemes in heterogeneous environments even after several upgrade processes, we are going to study the effects on the performance yielded by the resultant data layout when taking advantage of disk heterogeneity and moving only a fraction of it.

By simulating several workload applications, we have performed a set of tests to make a performance comparison between the final data layout used by our adaptive redundancy placement method and the data layout produced when completely changing the data layout such as is currently performed in RAIDX systems.

4.1 Simulation and Workload issues

In order to perform this evaluation, we have implemented both homogeneous and heterogeneous data placement using OSM redundancy on HRaid [22], which is a storage-system simulator that allows us to simulate storage hierarchy. We also have implemented on HRaid simulator the algorithm to adapt the data layout to the addition of new SSs.

Simulating the following scenarios performed the comparisons:

The Storage System Before and After Migration(BA Migration scenario) By using synthetic workloads, we can go forwards in time and perform several migrations in order to measure migration-by-migration both the service times both for traditional RAIDX, AdaptRaidX (our heterogeneous version) and our adaptive redundancy placement.

This experiment makes sense because we want increased bandwidth and/or capacity when the storage system is not able to address an increment on demand. Using synthetic workloads, with increases in both their access and address space, can simulate this.

We have performed one workload per migration in order to simulate a data base system workload. The workload information has been extracted from [23] and their features have been increased according to [24].Table 1 shows the features of the workload used in each migration.

	HAWK1	*hp 97560*	ST15230W	*90871u2maxtor*	WD204BB	ST136403LC
Formatted capacity (GB)	1	1.28	4	8	18.7	33.87
Block Size (bytes)	1024	1024	512	512	512	512
Sync Spindle Speed(RPM)	4002	5400	5400	7200	7200	10000
Average Latency (ms)	8.2	5.7	5.54	5.54	4.16	2.99
Buffer	64KB	128KB	512Kb	1MB	2MB	4MB
DskID	A	B	C	D	E	F

Table 2. *Specifications of used disks*

The Storage System after several Migrations (AS Migrations scenario) : By using real financial traces (OLTP [25]), we can go backwards in time and then start to perform migration after migration until we arrive at the year when the trace was created and until the address space of the real trace is accomplished. Afterwards, we measure the effects of data manageability, capacity usage and performance.

This experiment makes sense because a real trace cannot be manipulated in order to predict new accesses, but we can use it on an environment that has been upgraded many times and observe the performance of both Adaptive and static placement methods after several migrations.

4.2 Configurations studied

In both scenarios we are always adding between 30% and 50% [3] [2] of the current capacity and use disks according to technology's trends of hard disks [3] corresponding to each migration.

Table 2 shows the features of the disks used by the SS in both scenarios. The DiskID was included in table 2 for identifying each kind of disk in the rest of chapter.

BDA Migration scenario: We have simulated a storage system of ten SS with five disks type E each. We have performed three migrations to scale the capacity of the storage system from 935 GB to 2,45 TB.

Table 3 shows disks and SS used in this scenario as well as details about the capacity added and the overall capacity of the storage system in each migration. The SS_ID means several SS have the same features.

We have tested the following configurations per migration for both Adaptive and static methods:

RAIDX Start: This is the current configuration of the storage system.

RAIDX Final This is the resultant configuration once the new SS have been added and a re-striping (homogeneous) has been performed.

AdaptRaidX Final: Similar to **RAIDX Final** but the re-striping has been performed using all SS according their capabilities.

AdaptiveX Final: This is the resultant configuration once the new SS by using our adaptive proposal.

AS Migrations Scenario: We have performed 10 migrations, resulting in two upgrades of the storage system at year until achieving the address space of the real trace.

	Starting Configuration	*Migration 1*	Migration 2	*Migration 3*
Starting Capacity(GB)	467.5	654.5	993.2	1331.9
Added Capacity(GB)	0	187	338.7	338.7
Added SS	5	2	2	2
Disks per SS	5	5	5	5
Used disks (dskID from table2)	E	E	F	F

Table 3. *Specifications for* ***BDA Migration Scenario***

		Mig 1	Mig 2	Mig 3	Mig 4	Mig 5	Mig 6	Mig 7	Mig 8	Mig 9	Mig 10
Starting Cap(GB)	56	68.8	99.51	124.07	148.63	180.77	220.94	277.04	333.14	434.75	536.36
Added Cap(GB)	0	12.8	30.72	24.56	24.56	32.14	40.17	56.1	56.1	101.61	101.61
Added SS	7	1	2	1	1	1	1	1	1	1	1
Disks per SS	8	10	6	6	6	4	5	3	3	3	3
Used disks (dskID)	A	B	B	C	C	D	D	E	E	F	F

Table 4. *Specifications for* ***AS Migrations scenario***

Table 4 shows the same information shown in table 3 but for 10 migrations. We have tested Start and Final configurations for traditional RAIDX and our adaptive redundancy placement method.

Since several migrations on the time means the usage of different kind of disks, the tests performed allow us to observe almost all cases when upgrading storage subsystems.

5 Experimental results

In this section we present the results obtained by the simulation of the tests proposed for both above-mentioned scenarios.

5.1 Evaluating *BDA Migration scenario*

Evaluating the effects when handling disk heterogeneity

The addition of new disks improve the service times of disk array, but a good enough management of disks with higher capabilities can also improve them.

Figure 10 shows the percentage of improvement delivered by the data layout produced when applying the re-striping technique. Four configurations are shown

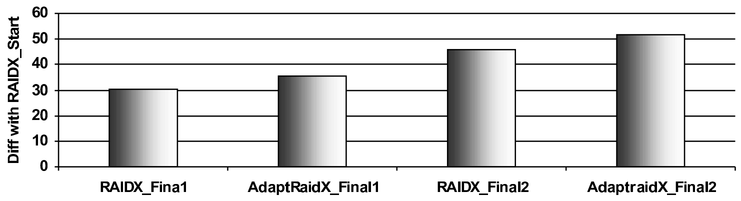


Fig. 10. *The difference in performance of RAIDX Final and AdaptRaidX Final with RAID Start.*

	Start	*Migration 1*	*Migration 2*	Migration 3
Metadata Cost moving segments (KB)	**3.43**	**5.88**	**8.41**	**12.16**

Table 5. *The metadata cost when managing segments for BDA Migration scenario*

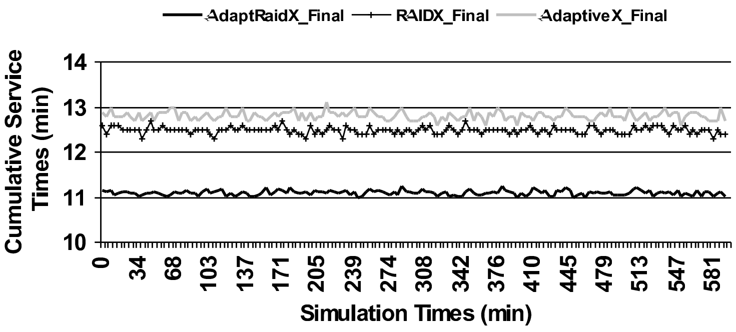


Fig. 11. *The service times for the configuration RAIDX_Final, AdaptRaidX_Final and AdaptiveX_Final*

RAIDX Final1 the homogeneous, *AdaptRAIDX Final 1* (our heterogeneous proposal), *RAIDX Final 2* and *AdaptRAIDX Final 2* for second migration.

In all experiments the services times were measured in clusters of 50x10e3 requests Figure 10 shows a mean values for four configurations using the trace *mig1* in Final 1 and *mig2* in Final 2.

As we can see, the service times are improved in mean when taking advantage of the disks with higher capabilities.

Evaluating the impact of the rearrangement methods on services times In this case we have evaluated our Adaptive proposal (AdaptiveX) that move only a fraction data.

Figure 11 shows the service times observed by *RAIDX Final*, *AdaptRaidX Final* (both upgraded with re-striping) and *AdaptiveX Final* moving segments. The three configuration are simulated by using *mig1* synthetic trace.

The horizontal axis represents the simulation time, (measured every 50x10e3 requests). The vertical axis represents the cumulative service times for these requests, to compare easily each point in the three lines.

AdaptiveX Final only moves between 20 and 40 % data but its resultant data layout yields a similar performance to traditional method of re-striping, which move 100% of data layout.

In fact, we can observe that the performance difference with RAIDX was only of 4% and 16 % with our *AdaptRaidX* version, which also moves all data.

The *AdaptRaidX Final* configuration yields better services times because it can make use of the disk heterogeneity while AdaptiveX do not in this experiment because it is using only diagonal segments.

When increasing the address of space for managing more capacity in migration 2 and 3, only AdaptRaidX and AdaptiveX can manage such increment.

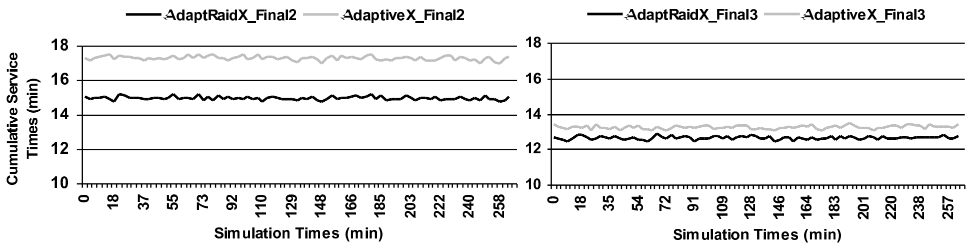


Fig. 12. *The service times for the configuration AdaptRaidX_Final and AdaptiveX_Final in migration 2 (left) and migration 3 (right)*

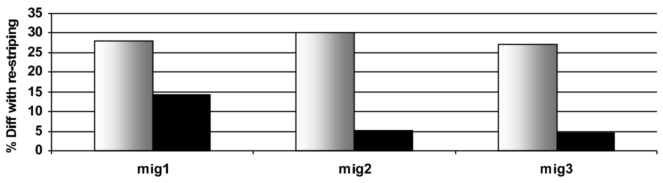


Fig. 13. *The difference of performance observed with AdaptRaidX (black) and the capacity migrated by AdaptiveX (gray).*

Figure 12 shows the service times for AdaptRaidX and AdaptiveX in migration 2 (left) and migration 3 (right). As we can see, the difference of performance observed with AdaptRaidX has been reduced during the three migrations from 16% to 4 % while the capacity moved by AdaptiveX, is 28% in mean.

Figure 13 shows the capacity moved by rearmament performed by AdaptiveX and how the difference of performance observed with AdaptRaidX has been reduced during the three migrations.

Finally, the metadata used when managing segments through the three migrations has been only few KBs as we can see in table 5.

Evaluating the AS Migrations Scenario

As 10 migrations were performed here we focus first on the capacity that both homogeneous re-striping as well as our adaptive scheme based on segments have migrated.

Figure 14 shows the capacity migrated by both techniques during 10 upgrade processes. As que can see, both techniques follows an exponential behave but it is dramatic in the re-striping case.

Evaluating service times: In figure 15 we can observe that *AdaptiveX Final* configuration yields service times very close to *AdaptRaid Final* for this scenario even after several migrations.

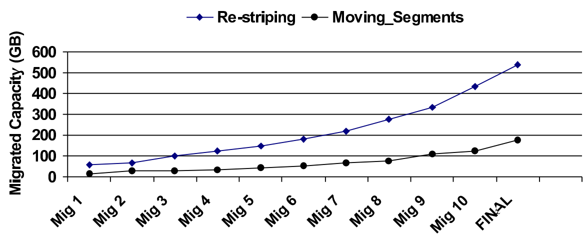


Fig. 14. *The capacity migrated by the techniques re-striping and segments movement.*

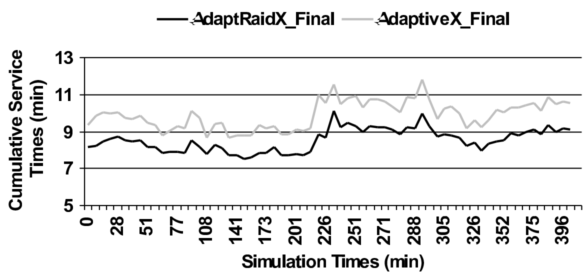


Fig. 15. *The services times of AdaptRAIDX Final and AdaptiveX Final after 10 migrations.*

6 Conclusions

We have presented a proposal for managing OSM in distributed heterogeneous disk arrays (AdaptRaidX), which delivers better service times than homogeneous RAIDX. We also have presented AdaptiveX rearrangement scheme, which only moves between 20 and 40 % data, the resultant data layout yields a similar performance to traditional method of re-striping, which rearranges the 100 % of the data layout. It also reveals when a set of heterogeneous disks are used the data layout produced by the rearrangement scheme is able to deliver even better performance and more capacity is achieved in the same comparison.

In addition, the heterogeneity management only needs few KB as metadata and the migration scheme used by this approach produces a low overhead within an acceptable time. Finally, it keeps the complexity of the data management at an acceptable level.

References

1. P. Charles N. Good L. L. Jordan P. Lyman, H. R. Varian and J. Pal. How much information? 2003? *http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/printable_report.pdf*, 2004.

2. K. G. Coffman and A. M. Odlyzko. Internet growth: is there a "moore's law" for data traffic? pages 47–93, 2002.
3. E. Grochowski and R. D. Halem. Technological impact of magnetic hard disk drives on storage systems. *IBM Syst. J. IBM Corp.*, pages 338–346, 2003.
4. Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and David A. Wood. Dbmss on a modern processor: Where does time go? pages 266–277, 1999.
5. Ron Yellin. The data storage evolution. has disk capacity outgrown its usefulness? *Terada magazine*, 2006.
6. T. Cortes and J. Labarta. Taking advantage of heterogeneity in disks arrays. *In Journal of Parallel and distributing Computing Volume 63*, pages 448–464, 2003.
7. G. Gibson D.A.Patterson and R.H. Katz. A case for redundant arrays of inexpensive disks (raid). pages 109-116. *SIGMOD*, pages 109–116, 1988.
8. R.S.C. Kai Hwang; Hai Jin; Ho. Orthogonal striping and mirroring in distributed raid for i/o-centric cluster computing. *PDS, IEEE Transactions 2002*, pages 26 – 44, 2002.
9. M. Holland and G. Gibson. Parity declustering for continuous operation in redundant disk arrays. *ASPLOS-V*, pages 23–35, 1992.
10. Garth A. Gibson and David A. Patterson. Designing disk arrays for high data reliability. *Journal of Parallel and Distributed Computing*, pages 4–27, 1993.
11. R.S.C. Kai Hwang; Hai Jin; Ho. Raid x. *Parallel and Distributed Systems, IEEE Transactions on Volume 13, Issue 1 2002*, pages 26 – 44, 2002.
12. J. Renato Santos and R. Muntz. Performance analysis of the rio multimedia storage system with heterogeneous disk configurations. *ACM98*, pages 303–308, 1998.
13. A. Brinkmann, S. Effert, M. Heidebuer, and M. Vodisek. Influence of adaptive data layouts on performance in dynamically changing storage environments. In *PDP '06*, pages 155–162, 2006.
14. Heinz Maulschagen. Logical volume management for linux.
15. David Nagle, Denis Serenyi, and Abbie Matthews. The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In *SC '04*, page 53, 2004.
16. Lionel M. Ni Y. Chen and M. Yang. Costore: A storage cluster architecture using network attached storage devices. *in: Proceedings of the Ninth.ICPADS'02*, 2002.
17. J.L. Gonzalez and T. Cortes. Adaptive data placement based on deterministic zones (adaptivez). *GADA07*, page pending, 2007.
18. Weikuan Yu, Shuang Liang, and Dhabaleswar K. Panda. Adaptive data availability for chained declustering. *ACM ICS '05*, pages 323–331, 2005.
19. A. Brinkmann, S. Effert, and F. Meyer auf der Heide. Dynamic and redundant data placement. *ICDCS '07*, page 29, 2007.
20. J. L. Gonzalez and T. Cortes. Increasing the capacity of raid5 by online gradual assimilation. *SNAPI 04*, pages 17–24, 2004.
21. J. L. Gonzalez and T. Cortes. Evaluating the effects of upgrading heterogeneous disk arrays. In *SPECTS06*, 2006.
22. J. Labarta T. Cortes. Hraid: A flexible storage-system simulator. *In: Proceedings of the International Conference on parallel and Distributed Processing Techniques and Applications, CSREA Pres*, 163:772, 1999.
23. H. Franke N. Gautam Y. Zhang J. Zhang, A. Sivasubramaniam and S. Nagar. Synthesizing representative i/o workloads for tpc-h. *HPCA*, pages 142–151, 2004.
24. T. M. Madhyastha B Hong and B. Zhang. Cluster based input/output trace synthesis. *ipccc05*, 2005.
25. OLTP Application I/O. http://traces.cs.umass.edu/index.php/storage/storage.