Evaluating the Effects of Upgrading Heterogeneous Disk Arrays

J. L. Gonzalez {DAC},Toni Cortes{DAC,BSC}
joseluig@ac.upc.es, toni@ac.upc.es
Departament d Arquiectura de Computadors(DAC),Universitat Politecnica de Catalunya
Barcelona Supercomputing Center(BSC)

Keywords:RAID, Data Block Placement, Storage upgrade, Low-cost clusters, AdaptRaid.

***Abstract*— When a set of disks with different capabilities must be configured as a RAID system in low-cost clusters, applying a block placement technique that takes advantage of disks with higher capabilities (AdaptRaid) is the key to maximize the array potential (capacity and performance). However, the AdaptRaid practical application requires adaptability mechanisms because the current applications need expandable and reliable storage systems working in a 24/7 manner. This paper presents an algorithm that offers a reliability mechanism when recovering a damaged disk, upgrade mechanisms when adding or replacing disks, and a transition mechanim from an installed RAID system to an AdaptRaid one. Focusing on the increase of parallelism and without interrupting the normal operation, the mechanisms change a RAID or AdaptRaid block placement to an AdaptRaid one to greatly improve the arrays potential and achieve a balanced load. We examine the algorithm effects on the array normal operation. The evaluation reveals that the mechanisms improve the performance and reliability of arrays producing a low overhead.**

I. Introduction

New data is growing annually at the rate of 30% [1]. In order to satisfy such increasing demand on data-capacity with high I/O performance, the storage systems are periodically upgraded by adding disks, replacing old disks, replacing damaged disks, etc. This practice, as well as a constant improvement observed year after year in the disks capabilities [2], have both caused the development of heterogeneous storage environments.

The heterogeneous storage is especially common in low-cost clusters as a consequence of the upgrades performed periodically by adding and replacing nodes to meet the constantly growing computing requirements of scientific or general-purpose applications.

Currently, the heterogeneous approaches such as Logical Volume/Disks Managers [3], [4] and heterogeneous arrays for multimedia environments [5], [6] have fast upgrade processes, but produce an unbalanced load over the disk arrays.

The low-cost clusters are vulnerable to load imbalance, which results in performance degradation. Therefore, disks with different capabilities are configured as homogeneous RAID5 [7] because the data placement based on striping used by this configuration offers a load balancing, fault-tolerance and efficiency. However, RAID5 configuration assumes that all disks have the lowest common speed and the slowest common capacity resulting in a wasted potential (capacity and/or performance).

Upgrading a disk array at a reasonable cost, preserving the RAID5 benefits (a balanced load, fault-tolerance and efficiency) and taking advantage of disk heterogeneity is, for low-cost clusters environments, an optimal way to upgrade the array potential.

The solution to achieve this kind of upgrade is quite simple: upgrade from an installed block placement to a heterogeneous one that maximizes the array potential.

However, this solution is only available for homogeneous arrays, and in an offline manner in which a backup, restore and reload of the array are required. In addition, the online application of this solution produces a high overhead because the migration process delays the incoming I/O requests and this effect is increased when all data blocks have to be moved. Therefore, a mechanism able to minimize the overhead by gradually using the new disks bandwidth is the key in the online applicability of this solution.

We present an algorithm that, at reasonable cost and without interrupting the array normal operation, upgrades from an installed array block placement to a heterogeneous one. The new block placement preserves the RAID5 benefits, takes advantage of disk heterogeneity, maximizes the parallelism offered by the array and avoids any a posteriori capacity management process.

The algorithm is able to handle the disk heterogeneity and preserve the RAID5 benefits because it uses the AdaptRaid5 technique [8] to determine the new block placement of arrays.

The new block placement gives parallelism and heterogeneity benefits and the algorithm offers a mechanism to gradually profit from these benefits and the bandwidth of new disks to reduce the upgrade overhead until it is eliminated.

Currently, there are not available results about the effects of online upgrading the data placement for the scientific and general-purpose applications. We examine the algorithm effects on the disk arrays normal operation for this kind of applications. The evaluation reveals, and it is the most significant part of this work, that the mechanisms improve the performance and reliability of the arrays producing a low overhead during a short time.

This paper is divided into 7 Sections. Section 2 presents the most relevant works on the upgrade techniques for storage systems; Section 3 presents a brief overview on AdaptRaid5 thechnique; Section 4 shows in full detail the upgrade algorithm; Section 5 presents the methodology used to obtain the results presented in the section 6, finally our conclusions are presented in the Section 7.

II. Related Work

The mechanisms to replace failed disks in RAID systems have been deeply studied. They mainly consist on finding new placement algorithms that allow a better performance while the damaged disk is being reconstructed [9], [10] and [11]. In all these cases the replacement of a failed disk does not imply a reorganization of data but only a data recovery of the failed disk.

With the addition of new disks in homogeneous arrays, a full reorganization is needed [12] unless the same number of new disks (or virtual disks) is added to the end of the existing array [4]. This mechanism represents a limitation in the potential parallelism that can be achieved because its aim is optimizing the array capacity. In addition, it requires a data redistribution to balance the array load and the user intervention to manage the added capacity.

On the other hand, there has been some effort in multimedia environments in defining algorithms based on random [13], [14] or pseudorandom [15] placement that allow the addition of new disks to the array and only move the data that goes to the new disks leaving the rest of the data unmoved. Nevertheless, these solutions are only usable in environments that use large blocks, such as multimedia environments.Since the complexity in block management increases with small size blocks, this solution is not feasible for scientific and general-purpose applications, which commonly are suitable in low-cost clusters environments.

With heterogeneous disk arrays such as AdaptRaid, we identified the need to find a solution to change/replace/add disks and evaluate it.

Stripe	Dsk1	Dsk2	Dsk3	Dsk4
0	0,0	0,1	0,2	0,x
1	1,4	1,5	1,x	1,3
2	2,8	2,x	2,6	2,7
3			3,9	3,x
4			4,x	4,10

Fig. 1. *Distribution of data and parity blocks according to the intuitive idea.*

We want to emphasize that the upgrade proposal of this algorithm does not require human intervention once the upgrade process has finished, which is a very important property because the human errors have been identified as a significant source of unmasked system failures [16].

III. ADAPTRAID5 OVERVIEW

A. *AdaptRaid5 Block Placement*

The intuitive idea: The intuitive idea is to place more data blocks on disks with higher capabilities because these disks can serve more blocks for each unit of time. Following this idea, and as for regular RAID5, all *D* disks hold as many stripes as the number of blocks that fit on the disks with low capabilities. When these disks are full, the remaining disks can be used as if they were a disk array for *D-1* disks. The distribution continues until all the disks are full of data.

A side effect of this distribution is that the system may result in stripes with varied lengths such as an array with *D* disks where F of them with higher capabilities will have stripes with *D-1* blocks (plus the parity block). It will also have stripes with *F-1* blocks plus the parity block. For each stripe, the parity block is placed in the same position it occupied in the regular array and with as many disks as there are blocks in the stripe.

Figure 1 shows the distribution of blocks in a four-disk array, where disks 3 and 4 have higher capabilities than the others. In this figure, every block is labeled with the stripe number followed by a block number in the array (e.g. 0,2 represents data block 2, located on stripe 0). Parity blocks are simply labeled with an x and their respective stripe.

Patterns: If the algorithm is applied as we have described it so far, we observe that longer stripes are placed in the lower portion of the address space of the array while the shorter ones appear in the higher portion of the address space. The requests that fall in the lower part of the address space can use more disks (longer stripes) while the requests that fall in the higher part of it only use a small subset of the disks (shorter stripes). This can be a problem if the file system tries to place together all the blocks of a file, which is the usual practice [17].

Therefore, a given file may have most of its blocks in the lower part of the address space (long stripes) while another file may have all its blocks in the higher part of the address space (short stripes). Although the global access in the system will correspond to an average time, the first file will have a faster access time (more parallelism), while the second one will have a slower access time (less parallelism). For this reason, an even distribution of long and short stripes all over the array will reduce the variance between the access times in the different portions of the disk array.

To achieve this stripes distribution, a pattern was introduced to the algorithm. At the beginning, the algorithm assumes that the disks are smaller than they actually are (but with the same proportions in size) and distributes the blocks in this smaller array. This distribution becomes the pattern that is repeated until all disks are full.

The resulting distribution has the same number of stripes as the previous version of the algorithm. Each disk also has the same number of blocks as in the previous version. The only difference is that short and long stripes are distributed all over the array. An example of this pattern repetition can be seen in Figure 2. The pattern established can be repeated in disks thousands of times larger than the one presented in Figure 1. This concept of the patterns also serves to find the physical location of a given block.

Stripe	Dsk1	Dsk2	Dsk3	Dsk4
0	0,0	0,1	0,2	0,x
1	1,4	1,5	1,x	1,3
2	2,8	2,x	2,6	2,7
3	5,11	5,12	3,9	3,x
4	6,15	6,16	4,x	4,10
5	7,19	7,x	5,13	5,x
6	10,22	10,23	6,x	6,14
7	.	.	7,17	7,18
8	.	.	8,20	8,x
9			9,x	9,21
10			10,24	10,x

Fig. 2. *Example of pattern repetition.*

IV. AN ALGORITHM TO UPGRADE THE BLOCK PLACEMENT OF ARRAYS

In this section, we describe the algorithm that upgrades the block placement of the arrays, as well as the online-and-background mechanisms used to maximize the array potential and reduce the upgrade overhead. To this end, we describe the above mentioned upgrade scenarios, as well as the mechanisms used by the algorithm in each scenario.

Finally, we describe the mechanisms that the algorithm uses when a disk failure occurs during the upgrade process.

A. *Adding new disks to the array*

First of all, we will present a mechanism that allows the addition of N disks to an array (RAID5 or AdaptRaid5).

As we have mentioned above, the algorithm upgrades from an installed array block placement to an AdapRaid5 one. Therefore, in order to add new disks to the array, the algorithm defines two patterns: the first one based on the original array configuration and the second one based on a configuration that optimizes the new-disks capabilities.

Figure 3 shows how the algorithm defines both patterns to add 3 new disks to the original array (Figure 2). The new pattern maximizes the new-disks usage because they have higher capabilities than the original array ones.

Once the patterns have been defined, the algorithm can start the upgrade process by converting stripes from the old pattern to stripes in the new one.

This process implies two kind of requests:

The first request is a read operation and only uses the old disks. The size of this request is equal to the number of blocks that will be reorganized and has to be a multiple of a full stripe in the new array (to avoid unnecessary small writes).

The second request is a write operation that is sent exactly after the first request finishes. It uses the same blocks read by the first request to write a full stripe in the large array.

Stripes are converted in a sequential order and atomically with respect to regular file-system requests.

During the gradual assimilation process of disks some stripes use the new disks and some others only use the original set of disks. These two sets of stripes divide the array in the three following areas, as illustrated in Figure 4.

The original area: It does not include any block in the new disks and the stripes are ”small”. It contains all the blocks that have not been reorganized yet.

The reorganized area: It contains the blocks allocated according to the new AdaptRaid5 pattern and the stripes are ”large”. It contains all the reorganized blocks.

The void area: It includes a copy of the blocks that have already been reorganized in the new array but have not been rewritten yet.

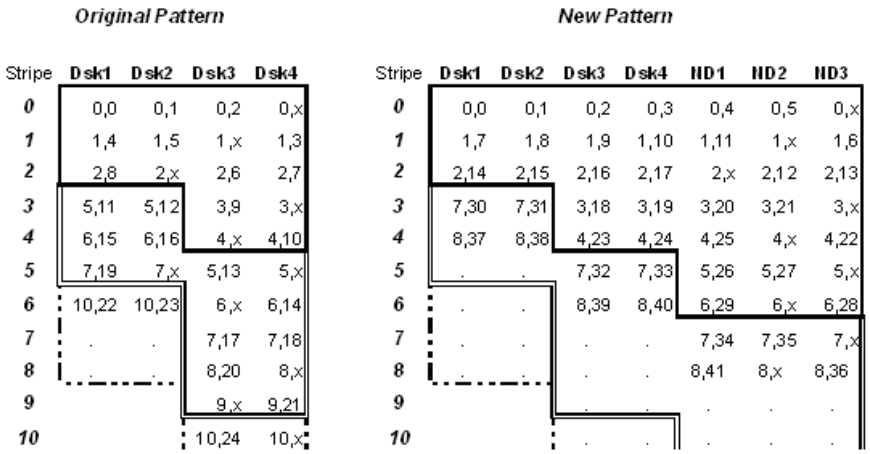

Fig. 3. *Original configuration (to the left) and the new pattern (to the right) defined to add 3 ND disks. The new pattern has stripes of three different lengths.*

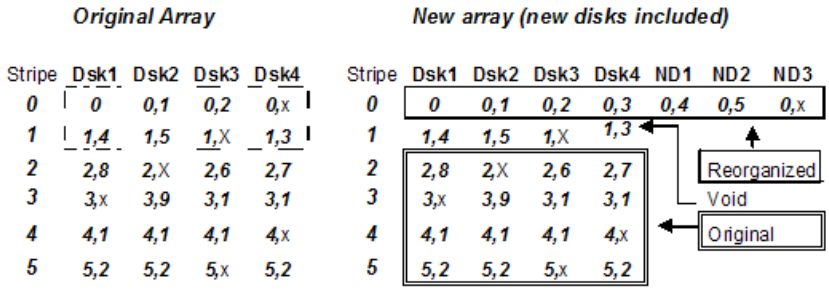

Fig. 4. *Array before (to the left) and after (to the right) that 6 blocks have been reconvert to the new array with 3 new disks.*

The management of these areas is crucial to serve user requests during the upgrade.

The algorithm uses a redirection mechanism that allows gradually serving user requests with the reorganized area (which includes the new disks and an AdaptRaid5 placement). **The reorganized area delivers better service times than original one. Therefore, it improves the array performance allowing a gradual reduction in the upgrade overhead until it is eliminated**.

On the other hand, the replicated blocks in void area are overwritten to avoid extra data migration and, strictly, are only overwritten when they have already been reorganized.

The stripe conversions are only initiated during idle periods to avoid interferences and minimize the upgrade overhead as much as possible. Studies [18],[19] have shown that the idle times are long enough to perform background tasks.

The algorithm maintains both patterns during the assimilation process of new disks until all the data have been properly placed. **Then the old pattern can be eliminated, leaving only the new one**. This process avoids, a posteriori, the application of complicated approaches to manage the added capacity.

B. *Replacing old disks with newer ones*

This scenario appears when old disks become a bottleneck to the array because they are too small or too slow compared to the rest of the disks on the array. The old disks can be replaced with faster and/or larger new disks. Obviously, this scenario appears only in heterogeneous arrays because a homogeneous array treats all disks equally. The mechanism used in this scenario is able to add as many new disks as the ones to be replaced. The difference with the first mechanism is that once the array has been upgraded, the old disks do not contain any block because they do not appear in the new pattern. They can be removed afterward without any problem.

C. *Transition from RAID5 to AdaptRaid5*

This scenario appears when we want to change the block placement of a heterogeneous set of disks. For instance, we could have an installed RAID5 configuration that does not take the disk heterogeneity into account or we could already have an AdaptRaid5 one, but we want to change the parameters of the configuration. In both cases, we have to move blocks around to match the desired new configuration. In this scenario, although no new disks are added, the mechanism to perform this reorganization is very similar to the previous ones: the algorithm converts an original array configuration into a new one. The new configuration is improved and the disks in the array respond better to the system needs.

D. *Workload issues*

E. *Replacing a damaged disk*

When a disk failure is detected in RAID5, the array can be restored to fault-free state by reconstructing in background each block of damaged disk on a new one. During this reconstruction, the array is vulnerable to data loss due to a second failure. So, the duration of the reconstruction process is commonly considered as a vulnerability window [10]. It is important to keep this concept in mind because any algorithm proposed to replace damaged disks must have a reasonable vulnerability window (as short as possible).

This reconstruction process has been deeply analyzed for RAID5 but not for AdaptRaid5. Accordingly, we present a mechanism that replaces a damaged disk by a new one and optimizes its capabilities in a RAID5 as well as AdaptRaid5. The mechanism offers two alternatives. The first alternative is to recover the failed disk and reorganize the data blocks to adapt the heterogeneity of the new disk at the same time.

	Fast and large (F) Seagate Cheetah ST136403LC	*Slow and small (S)* Caviar Western Digital WD204BB	*Slowest and smaller (Swt)* Seagate Hawk 4 ST-15230W
Formatted capacity (GB)	36.4	20.4	4.2
Block Size (bytes)	512	512	512
Average Sectors Per Track	302	63	110
Tracks	235224	688086	75848
Cylinders	9801	16383	3992
Sync Spindle Speed(RPM)	10016	7200	5411
Average Latency (ms)	2.99	4.16	5.54
Buffer	2MB	2MB	512Kb

TABLE I
Specifications of used disks

	SW	**HSW**	**DBHW**
% of read requests (Uniform distribution)	55%	58%	78%
Request size (Poison distribution)	12Kb	12Kb	8Kb
Request location	Uniform distribution	Uniform distribution	Uniform distribution 30% sequential
Arrival (ON/OFF model)	19 ms arrival time 450 ms OFF periods	13 ms arrival time 300 ms OFF periods	13 ms arrival time 300 ms OFF periods
Based on	[18]	[20]	[21]

TABLE II
Parameters used for the synthetic workloads

The idea here is to perform the same tasks as replacing an old disk, but with the small extra work of computing the blocks in the failing disk by reading the rest of the stripe and computing the XOR. This implies some overhead (to be measured) because the lost blocks have to be recomputed, but especially because new full stripes have to be written. This extra overhead (compared to the cost of just rewriting the lost blocks) may excessively increase the vulnerability window.

The second alternative tries to avoid the increase in the vulnerability window that may occur in the first alternative. The idea is to make the recovery in two steps: first, recover only the failed disk as it is normally done in RAID5 systems, and then perform the reorganization to adapt the new disk heterogeneity. This approach may reduce the vulnerability window but increases the time in which the overhead is observed.

F. Vulnerability of the algorithms

Since the algorithm are moving data, is very important to know what happens if one of the disks fails during the upgrade process. For this reason, we examine in this section the impact of a disk failure during the mechanisms application.

If a disk fails while **adding new disks**, **replacing old disks**, or **converting RAID5 to AdaptRaid5**, then we just need to stop the upgrade process. Afterward, we reconstruct the current state as traditional RAID5 does. This reconstruction will end up with a disk array that is in the middle of a reconstruction process, an ”unstable” one. From this point, we can follow with the upgrade as if nothing had happened.

It is important to treat the new disk as if it was the failed one (same characteristics). This means that if it is larger or faster, we will not take advantage of it yet. Once the new disk has been fully incorporated, if it has more potential than the failed one, then we can start another reorganization of the array. The reorganization should always be done after the disk has been recovered and the previous operation of reorganization has been finished. Summarizing, the disk failure in this case will not have a critical effect, only some performance penalty.

On the other hand, if another disk fails when **replacing a damaged disk**, then we have the same problem as with the traditional RAID5: the array is vulnerable to data loss due to a second failure.

V. METHODOLOGY

A. Simulation and environments issues

The upgade mechanisms presented in this paper are important, but more important is to know the impact they have in the both performance and reliability of the regular storage system usage. Users cannot be penalized too much for these background tasks and the array reliability can not be reduced.

In this section we present how the experiments are performed and in the next section we will present the results obtained.

In order to perform this evaluation, we have used HRaid [22], which is a storage-system simulator that allows us to simulate a disk array.

As we are going to study the effect of heterogeneity in the arrays, we have chosen three kind of disks: New disks (large and fast), already installed disks (small and slow) and old disks (smallest and slowest), which will be represented in the rest of this paper as F, S and Swt respectively. (See Table I). These disks and the hosts were connected through a Gigabit network (10 microseconds latency and 1 Gbits/s bandwidth). We simulated the contention of the network, but no protocol overhead was simulated. We also have to keep in mind that in the simulations we only took the network and disks (controller and drive) into account. The possible overhead of the requesting hosts was not simulated because it greatly depends on the implementation of the file system. The only issue we simulated from the file system was that it can only handle 10 requests (that can be of any size requested by the application) at a time. The rest of requests wait in a queue until one of the previous requests has been served.

We have studied the behaviour of our algorithm on a set of synthetic workloads because they allowed us to examine the behaviour related to each the system load. The loads have been extracted from different works that characterized I/O in different environments.

SW (Server Workload): This workload is based on the characteristics of the highest server’s workload analyzed in [18] and represents the load expected in a standard server for engineering work.

HSW (Heavy Server Workload): This workload extracts the information from the busiest period of the HP99 traces [20] and extends it for a larger period using the mechanism presented in [23]. The idea of this trace also represents an engineering environment but much more loaded.

DBHW (Data Base Heavy workload): We made this workload to simulate a data base system workload and extracted the information from [21].

A list of the parameters used to define the different workloads is presented in Table II.

B. *Configurations studied*

All the experiments presented in this paper have been done using disk arrays with 9 disks. This number of disks is large enough to see the possible benefits and limitations of the proposal. It is also small enough to make things easier to understand.

To evaluate the behaviour of our proposals, we perform a set of tests for each mechanism.

Adding new disks to the array: These tests were performed to evaluate the algorithm behaviour adding 2, 4 and 8 new disks (F) to one homogeneous and one heterogeneous disk array.

Replacing old disks by new ones: These tests were performed to evaluate the algorithm when replacing 2 old disks (Swt) with 2 new disks (F).

Replacing a damaged disk: In these tests we perform a reliability comparison between Raid5 and AdaptRaid5 systems and evaluate when replacing a damaged disk (S) with a new disk (F).

Transition from RAID5 to AdaptRaid5: The last tests were performed to evaluate the algorithm when changing the block-placement of array to recover the unused potential of the array.

Note that all tests are performed using a track as a rebuilt unit [9],[10] In addition, the size of the stripping units used is 32 Kbytes.

VI. Experimental results

For the purposes of our work, we define ***the overhead*** produced by the assimilation work as the increment observed in service time of the array.

On the other hand, ***the assimilation time*** is the required time to complete the upgrade process.This time is represented by *t* in the rest of the paper. The time *t* is divided in two parts: *t1* represents the overhead and *t2* the improvement observed in service times during the upgrade process. Therefore t = t1 + t2.

A. *Adding new disks to the array*

Here we show the results obtained when adding 2, 4, and 8 *(F)* new disks to an array with 9 *(S)* disks. Then we repeat the experiment adding the disks to a heterogeneous array with 5 *(S)* and 4 *(F)* disks. Figure 5 shows the service times observed in three different arrays: 9S disk array (the not upgraded configuration), 9S and 2F disk array (the upgraded configuration), and an 9S + 2F disk array, which starts as 9S disk array and is gradually converted to a 9S and 2F disk array.

The horizontal axis represents the assimilation time *(t)*, which was measured every 40x10e3 requests. The vertial axis represents the cumulative service times for these requests, to compare easily each point in the three lines.

In each experiment for this scenario, we observed the repetition of a behaviour pattern (See 9S + 2F array in Figure 5): there is a controlled overhead at the beginning of the assimilation *(t1)*, then the overhead is gradually eliminated and the algorithm also

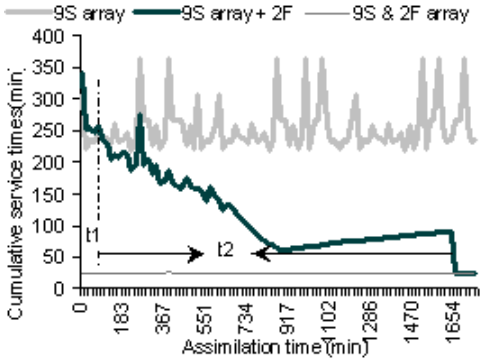

Fig. 5. *The service times differences adding 2F disks (using SW workload).*

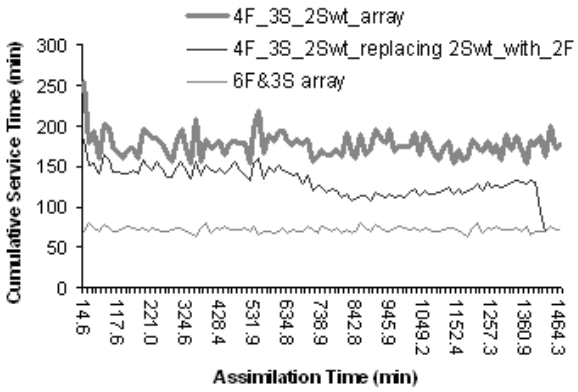

Fig. 6. *The service times replacing 2Swt with 2F disks (using SW workload).*

gradually improves the array performance during the greatest part of the assimilation process *(t2)*.

The algorithm application yields this behaviour because it is able to gradually use the new disks to serve file system requests.

By distributing the load over more disks, the algorithm achieves to reduce the overhead until it is eliminated. Moreover, the reorganization process reduces the amount of blocks per disk reducing disks seeks (of course this benefit will disappear when the array is fully upgraded and the file system is able to use the new blocks).

All these benefits are increased gradually and yield the effect shown in Figure 5. We observed it in all the experiments performed for this scenario. There is a sudden drop in the graphs all the way at the end for the disk arrays that are being modified in Figures 5 and 6 because at this point the upgrade process has been done and there are no delays induced to the incoming I/O requests Table III shows the results from the experiments using SW workload. This table shows *t, t1*, and *t2* for all experiments. It also shows the maximum overhead during *t1*, the mean overhead during *t1* and finally, the mean improvement of the array performance during *t2*. The same results are shown for HSW workload in Table IV and for DBHW workload in Table V. Adding more disks reduces assimilation time (See *t* times in Tables III, IV, and V) because less stripes are written (same amount of data distributed among more disks), more sequential blocks are read to build the new stripes and more parallelism can be used.

B. *Replacing old disks by new ones*

The following results were obtained by replacing 2Swt disks with 2F disks in a 4F-3S-2Swt array.

Configuration	t1 (hrs.)	t2 (hrs.)	t (hrs)	Max. Overhead during t1 (%)	Mean Overhead during t1 (%)	Mean Improvement during t2(%)
Adding 2F to 9S	1,13	26,44	27,57	10,18	6,26	27,31
Adding 4F to 9S	0,36	22,06	22,45	8,15	6,38	57,85
Adding 8F to 9S	0,36	15,42	16,18	8,48	6,44	73,43
Adding 2F to 5S-4F	1,13	20,54	22,07	13,76	9,96	36,3
Adding 4F to 5S-4F	0,55	17,33	18,28	10,31	10,01	52,5
Adding 8F to 5S-4F	0,36	13,47	14,23	8,73	6,8	62,2

TABLE III

Adding 2F, 4F and 8F to both the 9S array and the 5S and 4F Array. Tests with SW Workload

Configuration	t1 (hrs.)	t2 (hrs.)	t (hrs)	Max. Overhead during t1 (%)	Mean Overhead during t1 (%)	Mean Improvement during t2(%)
Adding 2F to 9S	0,58	83,32	84,3	3,97	2,93	28,82
Adding 4F to 9S	0,39	62,50	63,29	3,27	2,7	49,46
Adding 8F to 9S	0,39	44,50	45,29	1,83	1,82	62,79
Adding 2F to 5S-4F	1,50	55,31	57,21	7,66	3,8	27,46
Adding 4F to 5S-4F	0,27	45,57	46,24	6,7	4,1	55,4
Adding 8F to 5S-4F	0,14	36,36	36,5	8,01	8,01	64,4

TABLE IV

Adding 2F, 4F and 8F to both the 9S array and the 5S and 4F Array. Tests with HSW Workload

Configuration	t1 (hrs.)	t2 (hrs.)	t (hrs)	Max. Overhead during t1 (%)	Mean Overhead during t1 (%)	Mean Improvement during t2(%)
Adding 2F to 9S	0,44	71.32	72,16	4,3	3,16	30,81
Adding 4F to 9S	0,44	45.14	45,58	4,3	3,16	53,27
Adding 8F to 9S	0,14	30.41	30,56	2,75	2,75	66,14
Adding 2F to 5S-4F	2,27	77.57	80,24	6,81	3,1	23,82
Adding 4F to 5S-4F	1,14	53.33	54,47	5,76	2,29	39,55
Adding 8F to 5S-4F	0,21	36.17	36,38	0,67	0,67	57,24

TABLE V

Adding 2F, 4F and 8F to both the 9S array and the 5S and 4F Array. tests with DBHW Workload

Configuration	t1 (hrs.)	t2 (hrs.)	t (hrs)	Max. Overhead during t1 (%)	Mean Overhead during t1 (%)	Mean Improvement during t2(%)
SW	0	23,3	23,3	0	0	26,25
HSW	0	31,58	31,58	0	0	43,8
DHSW	0	24,02	24,02	0	0	20,17

TABLE VI

The assimilation times for three workloads replacing 2Swt with 2F in 4F-3S-2Swt array.

In the tests we observe the same behaviour as in the previous scenario.

The figure 6 shows the service times observed in three different arrays: one with 4F, 3S and 2Swt disks (4F-3S-2Swt array), one with 6F and 3S disks (6F-3F array), and an array that starts with 4F, 3S and 2Swt disks and it is gradually converted to a 6F-3F array. The x and y axes are the same as in the previous figure.

Table VI shows results of this experiment using the three studied workloads. These results are similar to the previous ones because this second scenario is very similar to the first one.

However, as we can see in Table VI, t1 is always 0, which means that there is no evident overhead period. This occurs because from the beginning we start reducing the usage of bottleneck disks, so the effect is seen immediately.

C. ***Replacing a damaged disk***

In this third scenario, we first want to verify that a heterogeneous array (AdaptRaid5) has, at least, the same vulnerability window as a RAID5 array.

This comparison contributes to show that the array heterogeneity does not increase the vulnerability window during the process of a disk failure recovery.

To this end, we compare the process of a disk failure recovery on a disk array with RAID5 configuration to the same disk array with an AdaptRaid5 configuration. In this comparison, we evaluate the vulnerability window on both configurations using the three studied workloads.

Figure 7 shows this experiment using SW workload. On the horizontal axis, we have the recovery time. On the vertical axis, as in previous graphs, the cumulative service times obtained by a 5S-4F disk array with RAID5 configuration and the same 5S-4F disk array but with AdaptRaid5 configuration, both in recovery mode.

Consequently, the duration of the disk failure recovery for both configurations is easily comparable. A vertical line shows the beginning and the end of the recovery process and the vulnerability

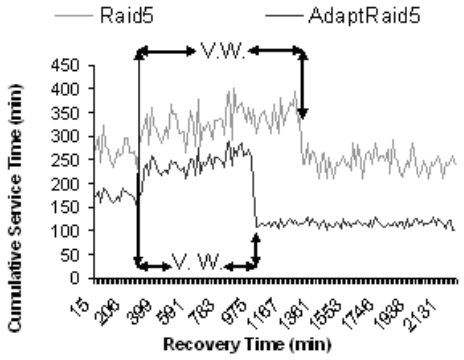

Fig. 7. *The service times for RAID5 and AdaptRaid5 configuration in recovery Mode using SW workload.*

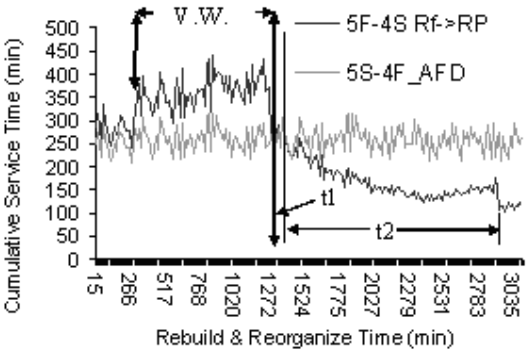

Fig. 9. *services times Recovering Fail disk in two steps.*

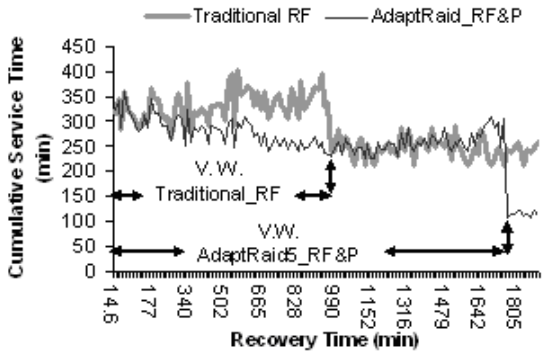

Fig. 8. *The services times recovering only disk Failure (Traditional-RF) and Recovering disk Failure plus the Potential that has not been used*

window for both configurations. It is very important to note that we use the same array for both configurations changing only the block placement. Results: the service times in both configurations after the vulnerability window are better than the times observed before this window because the spare disk is a F disk.

The results of this comparison show a decrease of the vulnerability window in the heterogeneous array of 28.9% for SW, 26% for HSW, and 30.7% for DBHW. As shown in [8] and seen before and after the beginning of vulnerability window in Figure 7, the service times in normal mode for heterogeneous arrays are better than for the homogeneous ones. In other words, the recovery time is shorter for the array with the AdaptRaid5 configuration.

This shorter time is also the result of the use by AdaptRaid5 of various size stripes (a kind of declustering), which has proved to be a good mechanism to reduce the vulnerability window [24].

Once we have showed that the time of the disk failure recovery for AdaptRaid5 configurations is acceptable, then we can start evaluating the effect of not just recovering but also adapting the heterogeneity to the characteristics of the spare disk. As we mentioned in section 4, there are two alternatives to solve this scenario. The first one where the algorithm recovers simultaneously the disk failure and reorganizes the blocks to adapt the spare disk heterogeneity. This solution, intuitively, will increase the size of the vulnerability window due to the data migration caused by this process. This hypothesis was verified by performing the same experiment: recover the failure of a 1S disk replacing it with a 1F disk in a 5S-4F array but adapting the new heterogeneity at the same time.

Figure 8 shows the vulnerability window and the system degradation of this solution and of the disk failure recovery traditionaly used in RAID5 systems. The x and y axes represent the same than in previous figures.

We observe in this figure that this solution solves quickly the degradation of the array comparing to traditional process. However, this process increases the vulnerability window. In fact, there has been an increase of 80% for SW, 100% for HWS, and 67% for DBHW.

It is an aggressive solution acceptable for environments that need to reduce soon the system degradation and can tolerate a larger vulnerability window, but it is not usually the case.

The second alternative is solving this scenario in two steps: first, recover only the disk failure, and afterward recover in normal mode the unused potential. This is a conservative solution acceptable for environments that need a small vulnerability window, which is the most common case.

Figure 9 shows how the algorithm recovers from a disk failure in two steps. The horizontal axis represents the recovery and reorganization time, and the vertical axis represents the cumulative service times for a 5S-4F array where a 1S is replaced with 1F spare in two steps (5S-4F-RF-RP). We observe the overhead produced at each step. This figure also shows the service times for the same array once the disk failure has been solved (5S-4F-AFD).

In this figure, we observe before the first vertical line that the overhead produced at the first step is the same than the one produced during the traditional disk failure recovery process. In addition, we see after this line the same behaviour pattern as when we add or replace disks. In the second step, we observe the time *(t1)* in which there is an overhead due to the reorganization. We also see, after the second vertical line, a time *(t2)* in which the algorithm improves the array service times. Finally, we observe that the time in which the potential of the spare is recovered equals *t1+t2*.

D. Transition from RAID5 to AdaptRaid5

The final experiment consists in improving the configuration (i.e. pattern) of a disk array to respond better to the new systems needs.

In this experiment, we simulated that a 5S and 4F array was configured as if the 9 disks had the same characteristics (performance and capacity). Here, the algorithm recovers the potential that has not been used in the 4F disks.

Table VII shows *t, t1, and t2* for these experiments. It also shows maximum and mean overheads during *t1*, and an improvement of the performance during *t2* for the three studied workloads.

Even if there is no new disks addition to the array, the behaviour of this mechanism is similar to the one previously observed in the other mechanisms. That means the *t, t1 and t2* values change according to the used workload for each experiment. In this scenario,

Configuration	t1 (hrs.)	t2 (hrs.)	t (hrs)	Max. Overhead during t1 (%)	Mean Overhead during t1 (%)	Mean Improvement during t2(%)
SW	1,28	25	26,28	7,44	5,11	33,23
HSW	1,04	53,54	54,59	12,75	7,75	36,10
DBHW	2,10	28,19	30,29	14,8	8,64	36,14

TABLE VII

the workloads recovering wasted potential of 4F disks and the spare.

the algorithm only uses the better block placement to reduce the assimilation work. However, even in this scenario *t1* is shorter than *t2*.

Finally, and as we have already mentioned, we focus our attention on scientific and general purpose applications because multimedia environments, and their special assumptions, have already been addressed[13], [14] and [15] . Onthe other hand, we have to keep in mind that these algorithms have been evaluated for an array built from disks attached to a SAN in a cluster of workstations, but there is no reason to believe that they would not work in other array configurations.

VII. Conclusions

We have presented simple but effective ways to upgrade online heterogeneous disk arrays by adding new disks, or replacing old ones. We also have proposed mechanisms that allow the replacement of failed disk while the characteristics of the new disk are taken into account in the heterogeneous array configuration.

We have proposed an online algorithm to change the blocks placement in the arrays in the case the system administrator decides that a new configuration would suit better the needs of the users.

Finally, and it is the most significant part of our study, a deep evaluation has showed that these mechanisms improve the performance and reliability of the arrays having a small performance degradation compare to the regular usage of the array.

References

[1] P. Charles N. Good L. L. Jordan P. Lyman, H. R. Varian and J. Pal. How much information? *http://www.sims.berkeley.edu/research/projects/howmuch- info-2003/*, 2003.

[2] E. Grochowski and R.F. Hoyt. Future trends in hard disk drives. *IEEE Transactions on Magnetics. Pages 1850-1854*, 1996.

[3] R. Zimmermann. Continuous me dia placement and scheduling in heterogeneous disk storage systems heterogeneous disk storage systems. *ph.d. thesis, University of Southern California*, 1998.

[4] Heinz Maulschagen. Logical volume management for linux.

[5] J. Renato Santos and R. Muntz. Performance analysis of the rio multimedia storage system with heterogeneous disk configurations. pages 303-308. *MULTIMEDIA 98 ACM*, 1998.

[6] A. Dan and D. Sitaram. An online video placement policy based on bandwidth to space ratio (bsr). pages 376-385. *SIGMOD 95*, 1995.

[7] G. Gibson D.A.Patterson and R.H. Katz. A case for redundant arrays of inexpensive disks (raid). pages 109-116. *SIGMOD*, 1988.

[8] T. Cortes and J. Labarta. Taking advantage of heterogeneity in disks arrays. pages 448-464. *In Journal of Parallel and distributing Computing*, 2003.

[9] G. Gibson M. Holland and D. P. Siewiorek. Fast, online failure recovery in redundant disk arrays. pages 422-431. *FTCS*, 1993.

[10] R.Y . Menon J. Patt Y.N. Hou. Balancing i/o response time and disk rebuild time in a raid5 disk array. pages 70-79. *HICSS'93*, 1993.

[11] M. Sivathanu, V. Prabhakaran, Andrea C., and Remzi H. Arpaci-Dusseau. Improving storage system availability with d-graid. *USENIX*, 2004.

[12] T. Cortes J. L. Gonzalez. Increasing the capacity of raid5 by online gradual assimilation. pages 17-24. *SNAPI 04*, 2004.

[13] K. Salzwedel. C Scheideler A. Brinkmann. Compact, adaptive placement schemes for non-uniform distribution requirements. pages 53-62. *SPAA*, 2002.

[14] Ethan L. Miller. R. J. Honicky. A fast algorithm for online placement and reorganization of replicated data. pages 267-268. *IPDPS*, 2003.

[15] S. Didi Yao R. Zimmermann. A. Goel, C. Shahabi. Scaddar: An efficient randomized technique to reorganize continuous media blocks. pages 193-194. *ICDE*, 2002.

[16] A. Ganapathi panchote D. Oppenheimer and D. A. Patterson. Why do internet services fail, and what can be done about it? *USITS*, 2003.

[17] Samuel J. Leffler Marshall K. McKusick, William N. Joy and Robert S. Fabry. A fast file system for unix. pages 181-197. *ACM Trans. Comput. Syst.*, 1984.

[18] W.W. Hsu and A.J. Smith. Characteristics of i/o traffic in personal computer and server workloads. pages 347-371. *IBM Systems Journal, Vol 42 No. 2*, 2003.

[19] J. Wilkes S. Savage. Afraid: A frequently redundant arrayof independent disks. pages 27-39. *USENIX 96*, 1996.

[20] HP-traces. http://tesla.hpl.hp.com/publicsoftware.

[21] H. Franke N. Gautam Y. Zhang J. Zhang, A. Sivasubramaniam and S. Nagar. Synthesizing representative i/o workloads for tpc-h. pages 142-151. *HPCA*, 2004.

[22] J. Labarta. T. Cortes. Hraid: A flexible storage-system simulator. pages 772-778. *ICPPTA, in CSREA Pres*, 1999.

[23] T. M. Madhyastha B Hong and B. Zhang. Clusterbased input/output trace synthesis. *ipccc05*, 2005.

[24] M. Holland and G. Gibson. Parity declustering for continuous operation in redundant disk arrays. pages 142-151. *ASPLOS-V*, 1992.

Dr. Toni Cortes is an associate professor at the Universitat Politecnica de Catalunya (Barcelona?Spain) since 1999. He obtained his M.S. and Ph.D. in computer science at the same university, and is currently the coordinator of the single-system image technical area in the IEEE Task Force on Cluster Computing (TFCC). His main interests are cluster computing and parallel I/O. Dr. Cortes has also been working on several European industrial projects such as Paros, Nanos, and POP. As a result of his research, both in basic research and technology transfer, Dr. Cortes has published more than 25 papers in international journals and conferences, has published 2 book chapters and has edited a book on parallel I/O (High Performance Mass Storage and Parallel I/O: Technologies and Applications, IEEE press). He also acts a program committee for many international conferences and journals.

J.L.Gonzalez got the engineering degree in computer systems in 1995 from Tamaulipas University (Tamaulipas, Mexico) and he is an associate professor at the Intituto Tecnologico cd. Valles (San Luis,Mex) since 1997. He joined Polytechnic University of Catalonia (UPC) in 2002 as PhD student at the Computer Architecture Department. His main interests are High Performance Mass Storage and heterogeneous Parallel I/O.