Increasing the capacity of RAID5 by online gradual assimilation

Jose Luis Gonzalez,Toni Cortes

joseluig,toni@ac.upc.es

Departament d Arquiectura de Computadors, Universitat Politecnica de Catalunya, Campus Nord, C6-002, C/Jordi Girona,1-3,ES-08034, Barcelona, Spain.

Abstract

Disk arrays level 5 (RAID5) are very commonly used in many environments. This kind of arrays has the advantage of parallel access, fault tolerance and little waste of space for redundancy issues. Nevertheless, this kind of storage architecture has a problem when more disks have to be added to the array. Currently, there is no simple, efficient and on-line mechanism to add any number of new disks (not replacing them), and this is an important drawback in systems that cannot be stopped when the storage capacity needs to be increased. We propose an algorithm to add N disks to an array while it continues running. The proposed algorithm for a gradual assimilation of disks has three major advantages: it has an easily controlled overhead, it allows the user to benefit from the higher parallelism achieved by the part of the array that has already been converted, and finally, it can be used in 7/24 systems.

1 introduction

One of the most important problems in current systems is the increasing performance and storage-capacity demand observed by applications. RAID5 [1] can solve both problems in a very simple way: just by adding more disks to the array. On the one hand, more disks means more capacity as the redundancy overhead is not increased by the addition of new disks. On the other hand, adding disks increases the bandwidth because more parallelism can be obtained. Although the problem of adding news disks seems to be a good solution, the process of disk addition is a big problem in itself, specially in highly loaded systems that have to run 24 hours a day 7 days a week. Currently, adding a new disk is done by one of the following procedures:
Stop the service.- is the most used solution but has a long downtime. To add disks, the system has to be stopped, backed up, and reloaded, and this may not be affordable in many system.
Concatenation.- implies that the same number of disks is added to the array making each original disks ”twice” larger. This has the problem that only capacity is increased, but not bandwidth. In addition, it does not allow the addition of any number of disks.
Add a new RAID5 to the cluster.- In this case the size problem is solved but the original array

is the same. There are tools like MylexMORE[2] designed in order to make on-line capacity expansion, that have limitations as the number of disk that can be added or that the user can't to stop the expand process, ACU by HP [3] is other expand capacity tool but both algorithms aren't public in order to make any comparison.
On the other hand, many computer systems in e-business, scientific, e-commerce, or web environments depend on the availability of information stored in their storage systems; so, the assimilation of new disks must be done online. Hence, this paper presents an algorithm that allows an online gradual assimilation of new disks to RAID5 avoiding downtimes. This algorithm will reconstruct the information (and parity) in the new array concurrently with application requests trying to take into account idle period, if any, but is not limited to them. As an interesting side effect, applications will take advantage of the gradual increase of bandwidth (due to the new added disks) attenuating the effects of the reconstruction overhead. In addition, we will also evaluate the overhead this procedure has on a running system.
The paper is divided into 7 sections. Section 2 presents the most relevant works on the reorganization techniques for storage systems; Section 3 shows in full detail the algorithm in section 3; Section 4 presents the methodology used to obtain the results presented in the section 5, finally our conclusions are presented in the section 6 and the future work in the section 7.

2 Related Work

The reorganization studies on storage systems are targeted mainly to not restrictive placement techniques. The random placement is used by these techniques in order to move so few objects as possible from the old disks to the new disks [4][5]; trying to achieve a balanced load over the added disks. The efforts in order to increase the bandwidth of the storage systems that use striping (restrictive) technique is an important research aim, but addressed mainly to multimedia environments [6], [7],[8] techniques of placements as the Staggered striping [9] are a popular solution to increase bandwidth by a decluster multimedia objects in cluster environments, of course, this striping technique made a placement of blocks at level 0; in these studies do not grows the capacity of the storage system. AutoRaid is a firmware of hierarchical storage proposed by Wilkes in [10] here are used two storage levels: a mirrored as the top level and a RAID5 as the bottom level, this approach allows an online storage capacity expansion, taking advantage of the additional space by allocating more mirrored storage but only takes advantage of more bandwidth when the assimilation is over and, on the other hand,

3 The Intuitive Idea

This algorithm is based on a simple idea: an online gradual assimilation of new disks by a sequential reorganization of full stripes. Our algorithm makes this reorganization according to a predefined schedule that tries to guarantee the assimilation of the new disks at a given time; this schedule performs reorganizations during idle periods or, when there are no idle periods, little by little adding a controlled overhead. The way the schedule is build will be described later.

Making the assimilation of new disks on-line has a very beneficial side effect that is an increase in the RAID access bandwidth. The algorithm increases the bandwidth because the new disks are gradually available in order to serve requests during the assimilation process; the gradual utilization of the new disks grows with the assimilation process. For instance, let's suppose that we are adding 2 disks to a 5 disk array. When half of the reorganization has been done, hopefully, half of the request will fall in the reorganized portion and will observe an increase of 50% in the bandwidth (two more disks are used to handle requests, the load is distributed over more disks and there are less seeks). The process of assimilating new disks to a RAID5 has three important issues that have to be discussed in detail:
1. The conversion of stripes from the small array to the new array. This is normally done by reading some blocks from the old array in order to write a full stripe in the new array.
2. The assimilation schedule. In order to control the overhead, the new array has to be built step by step concurrently with regular file-system operations. The schedule will decide when a new stripe is converted to the new array trying to reduce the overhead as much a possible.
3. Serving regular file-system requests. This action has to take into account that some part of the blocks already belong to the new and larger array while other blocks are still in the original array.

3.1 Stripe conversion

converting stripe from the small array to the new array (with added disks) implies two requests; these requests are sent one after another in strict sequence and atomically with respect to file-system 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 multiple of a full stripe in the new array (in order to avoid unnecessary small writes). Note, that this operation does not necessarily read full stripes in the small disks, but this is not important because the performance for reading part of a stripe is not penalized as happen in a small write.
The second request is a write operation, is sent exactly after the first request finishes and uses the same blocks read by the first request in order to write full in the large array. The process of gradual assimilation of disks implies that, during the process, there are stripes that use all disks and stripes that only use the original set of disks. These two sets of stripes divide the array in the three following areas (also seen in Figure 1):
The original area: It does not include any block in the new disks and the strips are "small". It contains all blocks that have not been reorganized yet and not necessarily starts at the beginning of a stripe.
The reorganized area: It has blocks in all disks originals and new and the stripes are "large". It contains all blocks that have been reorganized and only has full stripes.
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. These blocks are ignored and will soon be reused.

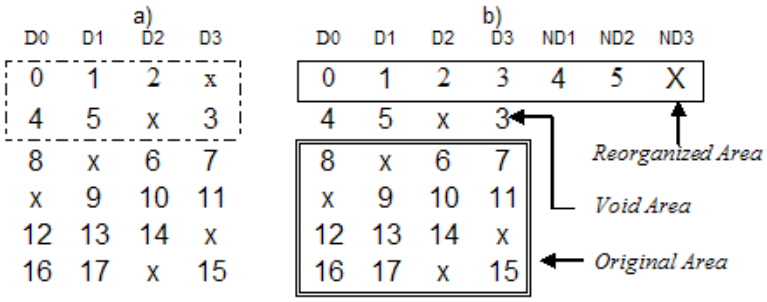

Figure 1: *This figure shows one conversion of stripe. The array a is the original array (in which 6 blocks will be reorganized) and the array b is the same array but after the first conversion of stripe (new disks included). In the array b can be observed the 3 areas.*

3.2 Assimilation schedule

The idea of the assimilation schedule is to finish the assimilation work within a specified time and distribute the stripe conversion actions among this time. For instance if the new array will have 6 stripes and we want is fully converted in 1 hour, we will start a stripe conversion every 10 minutes (of course this is an unrealistic example). Using this kind of schedule allows the administrator to set a limit to the overhead added. It is easy to have an idea of the maximum time need to do a stripe conversion (maximum seek time, read a stripe, compute parity, write a stripe, and another maximum seek that can affect the next user request). If we multiply this by the number of stripes in the new disk, we have the maximum time needed to perform the assimilation of the new disk, and thus we can decide the time we want for the assimilation (remember that the assimilation overhead will be equally distributed among the time). Implementing the algorithm just as mentioned before would not take into account idle periods to avoid interferences. The study performed in [11] shows that there are substantial idle times in order to perform background tasks. Furthermore, in AFRAID [12] and AutoRaid [10] the most operations are done during idle periods. In a similar way, we will also use these idle periods to advance in the assimilation work. The idea is to start an stripe conversion whenever an idle period is detected. Once this conversion is finished, if the disks continue idle, then a new stripe conversion is started. This procedure continues until a request from the file system arrives. As some work is advanced during idle periods, when the time for a scheduled stripe conversion arrives, we first check if the conversion due was already done during a previous idle period. If it has already been done, we do not disturb the disk with another conversion. On the other hand, if the stripe conversion due has not been done because the disk was busy, it is then started.

3.3 Managing regular file-system requests

As we have already mentioned, when a regular request reaches the array, it has to take into account that part of the array has already been converted while the rest has not. Note that in a disk fail case the assimilation process will be stopped and when the system come back to the

normal operation then the assimilation process stars in the same point previous to the fail disk.

4 Methodology

Simulation Issues: In order to perform this work, we use the HRaid [13], which is storage-system simulator that allows us to simulate an array made by the disks of the nodes in a cluster. The tests presented in this paper were performed simulating an array of 9 disks Seagate cheetah st136403LC, adding 1, 4 and 8 disks; The characteristics of the disks used are shown in the table 1. These disks and the hosts were connected trough a Gigabit network (10 ms latency and 1

Seagate Cheetah ST136403LC 36GB 80-pin SCA Hard Drive

Formatted capacity (gb)	36.4
Block Size (bytes)	512
Average Sectors Per Track	302
Tracks	235,224
Cylinders	9801
Sync Spindle Speed(RPM)	10,016
Average Latency (mSEC)	2.99
Buffer	2MB

Table 1: *Specifications of used disks*

Gbits/s bandwidth). We simulated the contention of the network, but no protocol overhead was simulated. We have to keep in mind that in the simulations we only took the network and disks into account. The possible overhead of the requesting hosts was not simulated from because it greatly depends on the implementation of the file system. The only issue we simulated from the file system was that it could only handle 10 requests at a time. The rest of requests wait in a queue until one of the previous requests has been served.
Workload issues: We have studied the behavior of our algorithm on a set of synthetic workloads based on the followings parameters: Kind of request.- Whether requests were reads or writes. This parameter was determined using a uniform distribution using the characteristics percentages in HP-suit traces of 99. Requests arrival time.- Arrival time to the array. According to the actual tends, this parameter was calculated with an ON/OFF function in order to simulate the burst behavior appreciated on [11] [14]. Requests size.- The size of all requests in the load. We chose a mean of 20kb, with a Poisson distribution. Requests location.- the position of the requests is always determined by uniform distribution over the disks as is considered in many studies.

5 Experimental results

The first workload analyzed had, on average, 51 requests per second and 19ms of inter arrival time. It is important to notice that these parameters are higher than the highest server's workload analyzed in [11], and thus are significant in order to simulate a real server's load. Figure 2 shows

3 graphs; graph 1 shows the service time observer in three different arrays: one with 9 disks, one with 10 disks, and an array that starts with 9 disks and it is gradually converted to a 10 disk array (with a time limit of 24 hours).

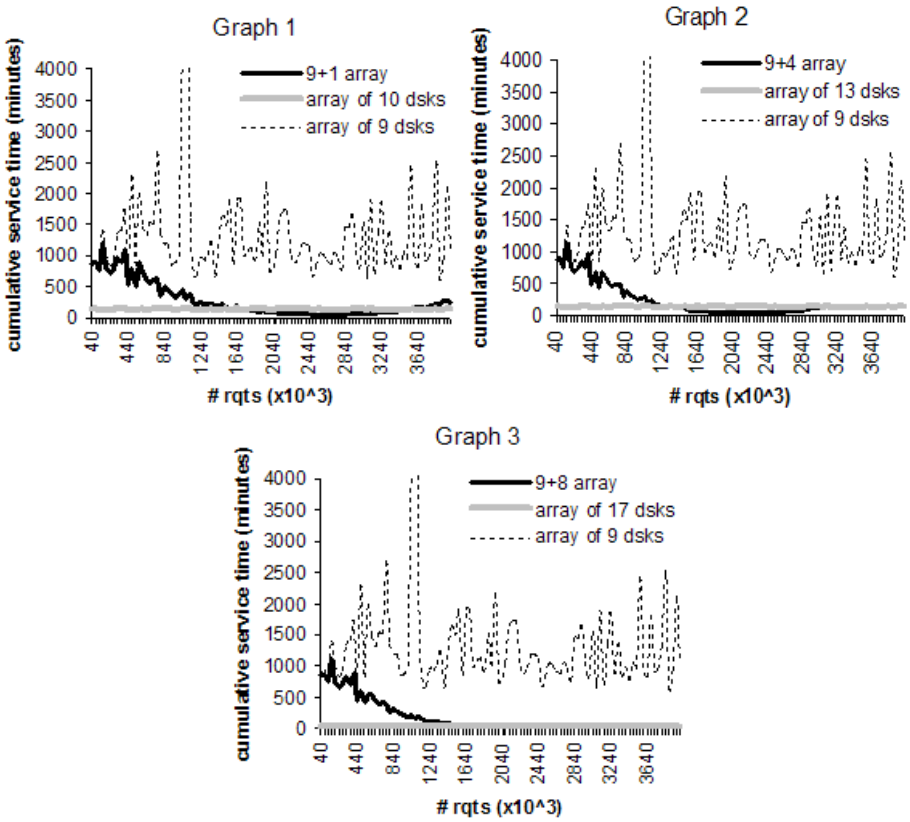

Figure 2: *This figure shows 3 graphs. In the graph 1 shows the differences of service times among an array of 9 disks, an array of 10 disks and an array of 9 disks adding 1 disk during a 22 hours period.Graphs 2 and 3 represent the same experiment as graph 1 but instead of adding one disk, 4 disks are added in graph 2 and 8 in graph 3*

In the x axes, we represent the request number and in the clustered in groups of 40x10e3 and the y axis represents the cumulative service time for these requests, thus each point in the three lines is easily comparable. The first thing we can observe is that the array with 10 disks is faster than the array with 9 disks. In addition, and the most important thing, is that the service time of the array that changes from 9 to 10 disks starts obtaining the performance of a 9 disks array, but gradually approaches the performance of the 10 disks array. We can see that there is no evident overhead and it is because most stripe conversions can be done during idle periods and thus they add few overhead to the system. The increase in performance occurs because as more stripes use 10 disks, more requests take advantage of this extra disk. Graphs 2 and 3 represent the same experiment as graph 1 but instead of adding one disk, 4 disks are added in graph 2 and

8 in graph 3. We can observe a very similar behavior and the explanation is also the same. The main difference is that the assimilation of the disks is faster when more disks are added. Adding more disks is faster because less stripes are written (same amount of data distributed among more disks) and more sequential blocks are read in order to build the new stripes. If the system has enough idle periods (as in the experiment) this is an advantage and thus the assimilation is done faster. As we will see latter this is not the case in a very highly loaded system. Figure 3 tries to show the results of these experiments. On the x axis, we have, as in the previous graph, the request number. In the y axis we have the percentage of gain/loss observed when comparing the service time obtained by the original array to the array that is incorporating 1, 4 or 8 disks. The

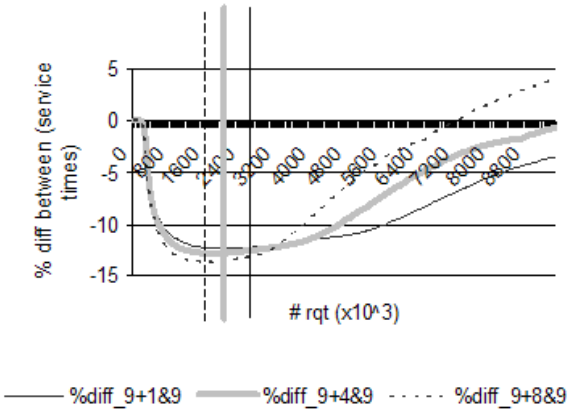

Figure 3: *This graph shows the percentage differences in service time between an array of 9 disks adding 1 (9+1), 4(9+4) and 8 (9+8) disks, assimilating these disks with our algorithm during a 42hours period; The are vertical lines for 9+1, 9+4 and 9+8 each indicates the moment in where requests begin to fall over the reorganized area. A plan of 2 days was employed in this case.*

vertical lines show the moment when requests start falling in the reorganized area and thus start taking advantage of the higher parallelism. We can see that in this case, as no idle periods can be used, requests are around a 14if the array were not being assimilating disks.This overhead seems quite reasonable, but remember that we can lower it by increasing the time needed to fully assimilate a new disk (2 days in this experiment).

6 conclusions

This algorithm is able of adding the new disks in a reasonable period of time. The main characteristics of this algorithm are: first, that disks can be added while the system is running normally (not downtime). Second that the overhead can be easily controlled. And, finally, that user requests can take advantage of the faster array while it is being reorganized.

7 Future Work

This kind of assimilation bemuse of greater interest in heterogeneous arrays and thus we plan to adapt this algorithm to one heterogeneous array like AdaptRaid [15] in the future.

References

[1] G. Gibson D.A.Patterson and R.H. Katz. A case for redundant arrays of inexpensive disks (raid). *In Proceedings of the 1988 ACM Conference on Management of data (SIGMOD)*, page 109 116, 1988.

[2] MyLEX MORE http://www.alfa.com.au/supports/mylex/raid/gam/m44205d1.pdf.

[3] ACU http://h18000.www1.hp.com/products/servers/proliantstorage/software management/acumatrix/advantage.html.

[4] C. Scheideler. A. Brinkmann, K. Salzwedel. Efficient, distributed data placement strategies for storage area networks. *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures*.

[5] Ethan L. Miller R. J. Honicky. A fast algorithm for online placement and reorganization of replicated data. *IPDPS 2003 17th. International Parallel and Distributed Symposium*, 2003.

[6] S. Didi Yao R. Zimmermann A. Goel, C. Shahabi. Scaddar: An efficient randomized technique to reorganize continuous media blocks. *IEEE 18th. International Conference on Data Engineering (ICDE'02)*, pages 473 – 482, 2002.

[7] Ghandeharizadeh and C. Shahabi. Management of physical replicas in parallel multimedia information systems. *In Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference*, pages 51 – 58, 1993.

[8] R. Baird F. Tobagi, J. Pang and M. Gang. Streaming raid: A disk array management system for video files. *In First ACM Conference on Multimedia*, pages 393 – 400, August 1993.

[9] R. Muntz Berson, S. Ghandeharizadeh and X. Ju. Staggered striping in multimedia information systems. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79 – 90, May 1994.

[10] C. Staelin T. Sullivan J. Wilkes, R. Golding. The hp autoraid hierarchical storage system. *in: Proceedings of the 15th. Operating System Review, ACM Press, New York*, pages 108 – 136, Feb 1996.

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

[12] J. Wilkes S. Savage. Afraid: A frequently redundant array of independent disks. *in : the Proceedings of the USENIX 1996 Annual Technical Conference*, Jan 1996.

[13] 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*, page 772 778, Jun 1999.

[14] C. Reummler and J. Wilkes. Unix disk access patterns. *In Winter USENIX Conference (San Diego, CA)*, pages 405 – 420, 1993.

[15] T. Cortes and J. Labarta. Taking advantage of heterogeneity in disks arrays. *In Journal of Parallel and distributing Computing Volume 63*, pages 448 – 464, April 2003.