Protecting Data Against Consecutive Disk Failures in RAID-5

Miguel S. Suárez-Castañón
ESCOM – IPN
Av. J. de Dios Bátiz s/n
México, D.F., 07738, México,
+(52)5555796000 xt. 52028
sasuarez@prodigy.net.mx

Jehan-François Pâris
Dept. of Computer Science
University of Houston
Houston, TX 77204-3010
JehanParis@aol.com

Carlos Aguilar-Ibañez
CIC – IPN
Av. J. de Dios Bátiz s/n
México, D.F., 07738, México,
+(52)5555796000 xt. 56568
caguilar@cic.ipn.mx

Abstract

In this letter we present a reorganization method to protect against data loss when one or two disks fail in a RAID level 5. The main advantage of the proposed method is that it is robust against a second failure if a first failed disk has not been replaced yet. Our proposal is motivated by the fact that new disks have a high possibility to fail during their first year of operation and during this period there is enough free space to rebuild the lost data in the failed disk and store it by a reorganization in the remaining disks.

Keywords: Data Survability, Disk Array, Storage System

1. Introduction

Current organizations of any kind base their operation and competitiveness on one of their most valuable resources: information. As such, information must be available at any moment and the risk of losing it due to flaws in the devices where it is stored should be eliminated.

At the present time an ever increasing number of organizations satisfy these two requirements using disk arrays as their online storage devices. This trend results from the convergence of several factors. First, advances in magnetic storage technology have considerably reduced the cost of storing data online. Second, regulatory requirements now obligate public corporations to retain their audit data over longer periods of time than in the past and to keep them immediately accessible. Finally, the rate at which digital data are produced keeps increasing in nearly all organizations [Lyman & Varian, 2000].

Given the above, we can affirm that the design of a storage system should be focused to ensure the survival of its data over periods that can span decades. Techniques like mirroring and erasure codes are two of the most used. Mirroring maintains multiple redundant copies of the stored data. Erasure codes, such as the well-known *m*-out-of-*n* codes, store data on *n* distinct disks along with enough redundant information to allow access to the data in the event *n*–*m* of these disks fail. The best-known organizations using these codes are RAID level 5, which uses an (*n*–1)-out-of-*n* code, and RAID level 6, which uses an (*n*–2)-out-of-*n* code.

In the case of RAID level 5 when one disk fails the lost data can be recovered and the storage system will be able to keep working, provided none of the remaining disks fail before the failed disk has been replaced and the system is completely restored to its normal configuration. Otherwise, some data will get lost. Then, when using RAID level 5 we must consider two issues that greatly compromise its data when functioning after one disk failure: a) disk infant mortality and b) the bad batch problem. Disks have much higher failure rates—between two and three times higher than those indicated by their mean time to failure—during their first year of operation. In addition, most failures resulting from a bad batch of disks also show up sometimes during that year. The traditional solution of burning in devices before actually using them would not help much since a prudent burn-in period would take one year and use up one fifth to one sixth of the disk lifespan [Elerath, 2000, Xin *et al*. 2005]. We are thus forced to use disks drives while they are still in their period of high infant mortality, and are still subject to bad batch failures.

We must take into consideration the high possibility that some of the previous problems can happen during the first year of operation of a RAID level 5, and could lead to a permanent data loss.

A solution to avoid the risk of data loss due to a second disk failure before the replacement of a first failed disk exists. This solution must allow us to restore the information contained in the second failed disk only using data previously stored in the free space in the remaining disks. Also, the number of XOR should be maintained to the minimum to increase the disks' overhead as little as possible. The main idea consists of storing the data of the first failed disk, after

its restoration, in the area where the redundant information was stored and the free space will be used to store the new redundant data, corresponding to the new configuration and a few copies of some of the data blocks. The data stored in the free space will be used if another disk fails and the first failed disk has not been replaced.

To evaluate the benefits of this new disk organization, we have analyzed the behavior of a small RAID level 5 system consisting of five disks. Our results indicate that our technique can reduce the probability of a data loss during the first year of operation of the system by at least 75 percent.[1]

The rest of the paper is organized as follows: Section 2 surveys previous relevant work. Section 3 introduces our technique and Section 4 evaluates its performance. Finally, Section 5 has our conclusions.

2. Previous Work

To reduce the possibilities of losing data stored in magnetic devices or disks, making copies has been a comon practice and may be the oldest data loss protection mechanism. Recently, with the appearance of redundant arrays of inexpensive disks, known as RAID [Patterson *et. al*, 1988] and the constant price reduction and increment of capacity and reliability of disk, the survability data levels have been considerably increased. Maybe the most used RAID comfigurations are RAID level 5 and RAID Level 6. These disk configurations are based on erasure coding, where RAID 5 uses (n – 1)-out-of-n codes [Chen *et al.*, 1994, Patterson *et al.*, 1988, Schwars and Burkhard, 1992, Schulze, *et al.*, 1989] and RAID 6 uses (n – 2)-out-of-n codes to protect data against double disk failures [Burkhard and Menon, 1993].

Self-organizing fault-tolerant disk arrays have been considered as a mechanism to protect data against disk failures. As an example, the HP AutoRAID [Wilkes *et al.*, 1996] automatically and transparently manages migration of data blocks between a replicated storage class and a RAID level 5 storage class as access patterns change. Its main objective is to save disk space without compromising system performance by storing data that are frequently accessed in a replicated organization while relegating inactive data to a RAID level 5 organization. As a result, it reacts to changes in data access patterns rather than to disk failures.

[1] Even better results could be achieved by taking advantage of the failure prediction capabilities of the new S.M.A.R.T. disks [Hughes *et al.*, 2002].

Much less work has been dedicated to self-organizing fault-tolerant disk arrays. The HP AutoRAID [Wilkes et al., 1996] automatically and transparently manages migration of data blocks between a replicated storage class and a RAID level 5 storage class as access patterns change. Its main objective is to save disk space without compromising system performance by storing data that are frequently accessed in a replicated organization while relegating inactive data to a RAID level 5 organization. As a result, it reacts to changes in data access patterns rather than to disk failures.

More related to our proposal, as it provides reorganization of remaining disk in the presence of a disk failure, is sparing. Having an extra disk in an array of disks allows us to use it as a failed disk replacement. Distributed sparing [Thomasian and Menon, 1997] gains performance benefits in the initial state and degrades to normal performance after the first disk failure.

In [Pâris *et al.*, 2006] a disk array organization that adapts itself to successive disk failures is presented. When all disks are operational, all data are mirrored on two disks. Whenever a disk fails, the array reorganizes itself, by selecting data by their exclusive or (XOR) with the other copy of the data contained on the disk that failed. Once the failed disk is replaced, the array returns to its original configuration. Since this scheme operates by replacing existing data by their XOR, with other data, it does not require any spare space. Its main drawback is a more complex recovery as the data that were overwritten need then to be restored.

Creating copies of data to be used to recover lost data due to disk failures was introduced in [Pu *et al.*, 1988], where the regeneration algorithm was presented.

3. Our Technique

Our goal is to increase the reliability of a RAID level 5 storage system during its first year of operation, a period during which it experiences higher disk failure rates than during the remainder of its useful lifetime. In addition, we wanted a solution that would not require any additional hardware. The solution we propose satisfies these two requirements since:

1. It uses the free space that normally exists on recently deployed drives to increase the redundancy of the stored data.
2. It brings no changes to the storage system as long as all disks are operational: new parity blocks and some copies of the stored data are only created in response to a disk failure and

are deleted as soon as the failed drive has been replaced.

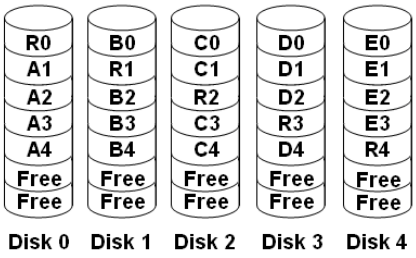

Fig. 1. A RAID level 5 consisting of five disks with free space.

Consider the RAID level 5 system in Fig. 1. It consists of five disks. We will assume that each disk has at least 2/7 of free space, a reasonable assumption for a disk array that has been recently deployed.

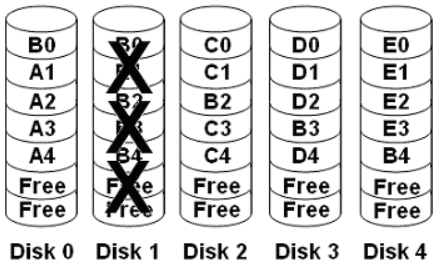

Fig. 2. The same RAID5 level 5 after the failure of disk 2.

Assume now that one disk, let us say disk 1, fails. As shown in Fig. 2, the new configuration holds the data blocks of disk 1 in the blocks where the parity blocks were originally stored. However, the array will become vulnerable to a second disk failure until the first failed disk is replaced. Although the arrangement maintains all the data that were stored before the first failure occurred, waiting for the replacement of disk 1 is not an attractive option as the process may take several days. Then, the internal organization of the array must be modified to be tolerant to another disk failure. To do this the parity blocks will be immediately computed as follows:

$$\begin{aligned} X0 &= C0 \oplus D0 \oplus E0, \\ X2 &= D2 \oplus E2 \oplus A2, \\ X3 &= E3 \oplus A3 \oplus C3, \\ X4 &= A4 \oplus C4 \oplus D4. \end{aligned} \quad (1)$$

And each of them will be stored in the first free block of each disk, as shown in Fig. 3. As can be seen from Fig. 3 and the expressions in (1), blocks **A1**, **B0**, **B2**, **B3**, **B4**, **C1**, **D1** and **E1** are not protected yet. Thus, we need to perform some extra **XOR**'s using some of these blocks and make copies of the remaining blocks, as follows:

$$\begin{aligned} Y0 &= C1 \oplus D1 \oplus E1, \\ Y2 &= B3 \oplus B4 \oplus B0, \\ Y3 &= B2, \\ Y4 &= A1. \end{aligned} \quad (2)$$

Finally, we need to store the values of *Y0*, *Y2*, *Y3* and *Y4* in the remaining free space in the array of disks, as shown in Fig. 4. Then, after these operations have been performed every data block is protected against the failure of any of the four remaining disks. In order to show that our proposed RAID system reorganization after one of its disks fail is still tolerant to a second disk failure, we present the tables in Fig. 5. Each table is related to each possible failed disk in our RAID system. These tables are divided in three parts: a) the new organization after the failure, b) which *X's* or *Y's* are needed to restore the blocks contained in a second failed disk and c) how the *X's* and the *Y's* were estimated. Here is an example to see how to interpret these tables. Let us suppose that disk 0 fails and our RAID system is reorganized as shown in Fig. 5A. While we wait for the replacement of disk 0, disk 1 also fails (this disk is represented by the shaded column in Fig. 5A.). Then, blocks B0, A1, B2, B3 and B4, contained in disk 1, will be lost. However from Fig. 5A we can see that using blocks *Y4*, *Y2*, *X2*, *X3* and *X4* respectively, they can be reconstructed. Finally, in Fig. 5A we can find how *Y4*, *Y2*, *X2*, *X3* and *X4* were computed. In a similar fashion we can verify, using the tables in Fig.5, that no matter what pair of disks fail in our RAID system, we will always be able to reconstruct the lost data.

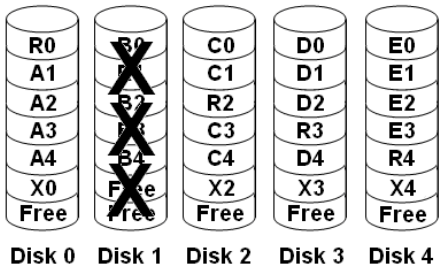

Fig. 3. The same RAID level 5 array after the parity blocks X_1, X_3, X_4 and X_5 have been computed and stored .

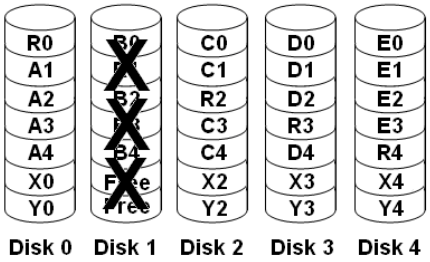

Fig. 4. The same RAID level 5 array after the parity blocks Y_1, Y_3, Y_4 and Y_5 have been computed and stored .

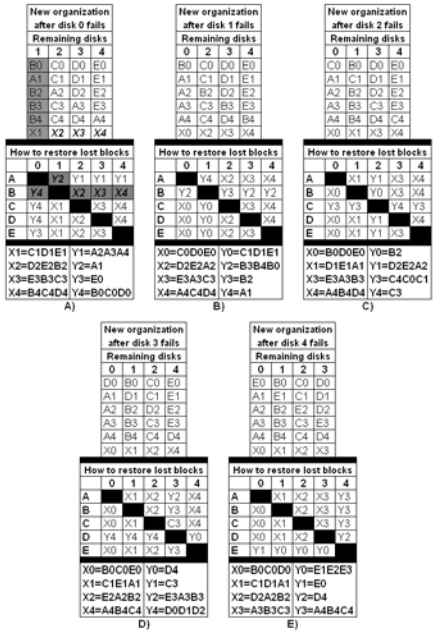

Fig. 5. Tables A, B, C, D and E show how to reconstruct the lost blocks if a second disk failure occurs before the first failed disk is replaced.

The previous discussion on how to rearrange a RAID level 5 system after one of its disk fails and keep it protected against a second failure while we wait for the replacement of the first failed disk, can be formally generalized with *Algorithm 1* to any number *n* of disks, provided *n>=5* and the blocks of the first failed disk are already reconstructed and stored, as shown in Fig. 2.

Algorithm 1 is divided in three sections. The first section determines which blocks will be used to estimate the *X's* values, that is, the new parity blocks. The second section determines which blocks will be used to estimate the *Y´s* values that they involve OR-Exclusive operations. The last section determines where the remaining unprotected blocks will be copied. The three sections determine where, inside the array, the *X's* and *Y's* values will be stored.

Algorithm 1: Let *k* be the first failed disk, *n* the number of disks in the array, matrix *r[0..n-1,0..n-1]* the representation of a RAID level 5 system and operator % denote the modulus operation.

```
nD ← n
fD ← k

/* Section 1: X's  values estimation*/

row ← fD+1%nD;
while(row<>fD)
   col ← row+1 nD;
   while(col<>row)
     if(col<>fD)
        r(nD,row) ← r(nD,row) ⊕ r(row,col);
     col ← col + 1%nD;
   row ← row+1%nD;

/* Section 2: Ys values estimation */

col ← fD+1;
while(col+1<> fD)
  r(nD+1,fD+nD-1%nD) ← r(nD+1, fD+nD-1%nD)⊕
r(fD,col);

r(nD+1,fD+1%nD)←r(nD+1,fD+1%nD)⊕r(col+1%nD,col+1%n
D);
   col ← col+1%nD;

/* Section 3: Protecting the last two unprotected blocks
*/
r(nD+1,fD+nD-2%nD) ← r(fD,col);

r(nD+1,fD+2%nD) ← r(fD+1,fD+1);
```

Roughly speaking, Algorithm 1 works as follows. Since we know the number of disks (nD) and which one failed (fD), we use these values as a starting point to perform, in section 1, a shift rotate for every row to compute the new parity blocks. For every shift rotate, we read all blocks in the corresponding row, except the block whose column coincides with that of the parity block we are computing. Once we finish with one row we move to the next along the main diagonal. Section 2 works analogous to section 1, but here we move simultaneously through the main diagonal and through the row that does not have parity block; in both cases we skip one block, for the same reason as in section 1.

Finally, in section 3 we made copies of the skipped blocks in the previous section to protect them.

Figures 6 and 7 show graphically how Algorithm 1 works.

It is worth mentioning that as the number of disks become greater than five, we will have for six disks one block free in the *Y's* section, two blocks for seven disks and so on. Then we have a tradeoff on the number of extra XOR's and the remaining free space. That is, we can slightly reduce the number of performed XOR's increasing the number of unprotected blocks to be copied to the free space, or we can keep the free space unused and in two the number of unprotected blocks to be copied. Fig. 8 shows the difference between these two options for the case of six disks and disk cero failed. In this figure we can see that leaving free the unused block we need to perform three XOR's, but if we copy there one block, the one not included in *Y1*, we reduce one XOR

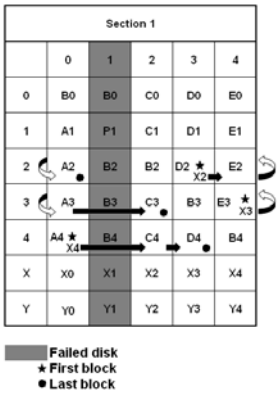

Fig. 6. After disk 1 fails, the new parity blocks *X2, X3* and *X4* are estimated. In this figure we illustrate the order in which the blocks are read by the section 1 of the proposed algorithm.

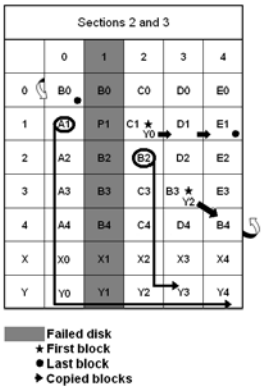

Fig. 7 This figure illustrates the order in which the blocks are read in section 2 of our algorithm to estimate *Y0 and Y1,* and which blocks are copied to *Y3* and *Y4* in section 3.

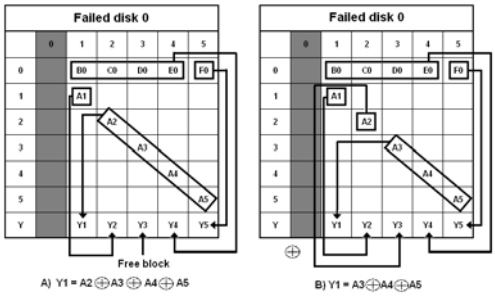

Fig. 8 This figure illustrates how we can reduce the number of XOR´s needed to protect all blocks after a disk failure, when the number of disks in the array is greater than five..

4. Reliability Analysis

To estimate the reliability of a storage system we must compute the probability *R(t)* that the system will operate during an interval *[0, t]* provided that it operated correctly when *t = 0*. To do it we must solve a system of linear differential equations, which eventually becomes an unmanageable task as the complexity of the system grows. An easy way to obtain *R(t)* is to estimate the mean time to data loss (MTTDL), which is the approach we will take here.

The system consists of an array of disks with independent failure modes. If a disk fails, we immediately start a repair process. When more than one disk fails the repair process starts in parallel on the failed disks' drives.

The disk failures are independent events exponentially distributed with rate λ, and the repairs are exponentially distributed with rate μ. Almost always the repair time is consumed by ordering and scheduling delays and the actual replacement of a failed disk only takes a few hours. Reorganization transitions consisting of the creation of additional copies of the stored data are equally assumed to be exponentially distributed with rate $\kappa > \mu$.

We consentrate our analysis on the first year of operation of the small disk array of Fig. 1. We will consider the case where none of its five disks is more than five sevenths full and we do not receive any early warning of future disk failures. Fig. 9 displays the state probability transition diagram of that array. State <0> is the normal state of the array when its five disks are operational.

When one of its five disks fails the system goes from state <0> to state <1>, that is the state of the array depicted in Fig. 2. This state is a less than desirable state as the array is not reorganized yet. Hence a failure of one of the remaining disks containing redundant

data would result in a data loss. To avoid that possibility, the array will restore the data corresponding to the failed disk using the parity blocks of the remaining disks, using *Algorithm 1*. This will move the array to state <1'>. State <1'> is the state of the array depicted in Fig. 3 and 4. A failure of any of the four remaining drives will bring the array into state <2'> where each of the remaining disks has enough redundant data blocks to restore all data after replacing the two failed disks. A failure of any of these three disks will therefore result in a data loss.

Repair transitions go from states <1> and <1'> to state <0> and, from states <2'> to state <1'>.

The Kolmogorov system of differential equations describing the behavior of the array is

$$\begin{aligned}
\frac{dp_0(t)}{dt} &= -n\lambda p_0(t) + \mu(p_1(t) + p_{1'}(t)) \\
\frac{dp_1(t)}{dt} &= -((n-1)\lambda + \kappa + \mu)p_1(t) + n\lambda p_0(t) \\
\frac{dp_{1'}(t)}{dt} &= -((n-1)\lambda + \mu)p_{1'}(t) + 2\mu p_{2'}(t) + \kappa p_1(t) \\
\frac{dp_{2'}(t)}{dt} &= -((n-2)\lambda + 2\mu)p_{2'}(t) + (n-1)\lambda p_{1'}(t)
\end{aligned}$$

where $p_i(t)$ is the probability that the system is in state <ij> with the initial conditions $p_0(0) = 1$ and $p_i(0) = 0$ for all other states.

The Laplace transforms of these equations are

$$\begin{aligned}
s\overset{*}{p}_0(s) &= -n\lambda \overset{*}{p}_0(s) + \mu(\overset{*}{p}_1(s) + \overset{*}{p}_{1'}(s)) \\
s\overset{*}{p}_1(s) &= -((n-1)\lambda + \kappa + \mu)\overset{*}{p}_1(s) + n\lambda \overset{*}{p}_0(s) \\
s\overset{*}{p}_{1'}(s) &= -((n-1)\lambda + \mu)\overset{*}{p}_{1'}(s) + 2\mu \overset{*}{p}_{2'}(s) + \kappa \overset{*}{p}_1(s) \\
s\overset{*}{p}_{2'}(s) &= -((n-2)\lambda + 2\mu)\overset{*}{p}_{2'}(s) + (n-1)\lambda \overset{*}{p}_{1'}(s)
\end{aligned}$$

Observing that the mean time to (MTTDL) of the array is given by

$$MTTDL = \sum_i \overset{*}{p}_i(0)$$

we solve the system of Laplace transforms for $s = 0$ and use this result to compute the MTTDL and the mean failure rate (1/MTTDL). The expressions we obtain are quotients of two polynomials that are too large to be displayed.

We assumed that the disk failure rate λ during the first year was one failure every one hundred thousand hours, that is, slightly less than one failure every eleven years. We let the average disk repair times vary between one half-day and one week and considered the two cases where the reorganization process could either take one or four hours.

From Fig. 10 we can see that the data loss probabilities achieved by our self-adaptive technique are significantly lower than those achieved by conventional RAID. The best results are obtained for a combination of a fast reorganization process (high κ) and a long repair time (low μ) as the reorganization process keeps the data protected during most of the repair process. Conversely, the reorganization process has much less impact on the array data loss probability when we have both a relatively slow reorganization process and a relatively fast repair process. Even then, the benefits of the reorganization process remain clear: our technique will always reduce the probability of a data loss during the first year of operation provided the the reorganization process takes at most 12 hours.

A last issue to consider is the applicability of our technique to larger disk arrays. We have only considered so far a very small array consisting of five disks. However, we can reasonably expect our proposed reorganization algorithm will work as well, if not better; in larger disk arrays as these arrays will be more likely to have spare spaces not involved in some previous reconfiguration process.

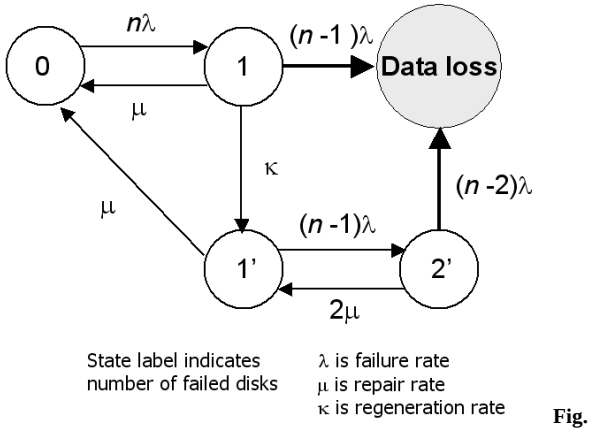

Fig. 9 State transition diagram for a self-adaptive array of five drives when none of its drives is more than five sevenths full.

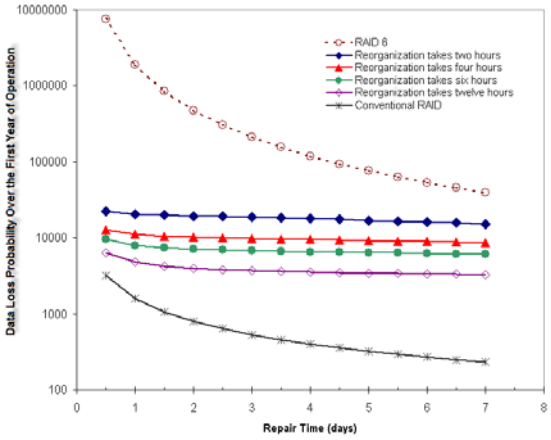

Fig. 10 Array failure rates during its first year of operation assuming that each disk drive is at most five seventh full.

5. Conclusions

We have presented a new technique for protecting data against the increased risk of disk failures during their first year of operation. When all disks are operational, all data are protected on five disks because all the parity blocks were constructed and stored in the corresponding places. Whenever a disk fails, the system reorganizes itself, by reconstructing the lost data contained in the failed disk and computes the new parity blocks. This new reorganization guaranties that in the event that a second disk fails before the fist one is replaced, no data will be lost. To evaluate the benefits of this new disk organization, we have analyzed the behavior of a small system consisting of five disks under standard Markovian assumptions. Our results indicate that our technique can reduce the probability of a data loss during the first year of operation of the system.

More work is still needed to evaluate the performance of our technique on larger disk arrays, investigate more realistic repair time distributions and measure the impact of our technique on the data survival rate over the whole lifetime of a disk array.

6. Acknowledges

This work was supported by CIC-IPN, and by the Secretaría de Investigación y Posgrado (SIP-IPN) under research grants 20071088 and 20071109.

Miguel S. Suárez-Castañón wants to thank to the IPN and to the FIDERH of the Banco de México for making possible his postdoctoral stay at the University of Houston. This work was done while M. Suárez stayed at the University of Houston

6. References

[1] Burkhard W. A. And Menon J., "Disk Array Storage System Reliability", *Proc. 23rd. International Symposium on Fault-Tolerant Computing (FTCS-23)*, pp. 423-441, 1993.

[2] Chen P. M., Lee E. K., Gibson G. A., Katz H. And Patterson D. "RAID, High-Performance, Reliable Secondary Storage", *ACM Computing Surveys*, Vol 26, No. 2, pp. 145-185, 1994.

[3]Elerath J. G., "Specifying Reliability in the Disk Drive Industry: No More MTBF's", *Proc. 2000 Annual Reliability and Maintainability Symposium*, pp. 194-199, Jan. 2000.

[4]Hughes G. F., Murray K., Kreutz-Delgado and Elkan C., "Improved Disk-Drive Failures Warnings", *IEEE Trans. On Reliability*, Vol. 5, No. 3, pp. 50-57, Sep. 2002.

[5] Lyman P. and Varian H. P., "How Much Information?", *The Journal of Electronic Publishing*, http://www.press. umich.edu/jep, Vol. 6, No. 2, Dec. 2000.

[6] Pâris J.-F., Schwarz T. J. E. And Long D. D. E., "Self-Adaptive Disk Arrays", *Proc. 8th. International Symposium on Stabilization, Safety and Security of Distributed Systems (SSS 2006)*, Dallas, TX., pp. 469-483, Nov. 2006.

[7] Patterson D. A., Gibson G. A. And Katz R. H., "A case for Redundant Arrays of Inexpensive Disks (RAID)", *Proc. SIGMOD 1988 International Conference on Data Management*, pp. 109-116, June 1988.

[8] Pu C., Noe J. D. And Proudfoot A., "Regeneration of Replicated Objects: A Technique and Its Eden Implementation", *IEEE Trans. On Software Engineering*, Vol. 14, No. 7, pp. 936-945, July 1988.

[9] Thomasian A. and Menon J., "RAID 5 Performance with Distribuited Sparing", *IEEE Trans. On Parallel and Distribuited Systems*, Vol. 8, No. 6, pp. 640-657, June 1997.

[10] Schulze M., Gibson G. A., Katz R. H. And Patterson D. A., "How Reliable Is a RAID?", *Proc. Spring COMPCON '89 Conference*, pp. 118-123, March 1989.

[11] Schwars T. J. E. And Burkhard W. A., "RAID Organization and Performance", *Proc. 12th International Conference on Distributed Computing Systems*, pp. 318-123, June 1992.

[12] Wilkes J., Golding R., Stealing C. and Sullivan T., "The HP AutoRaid Hierarchical Storage System", *ACM Trans. On Computer Systems*, Vol. 14, No. 1, pp. 1-29, Feb. 1996.

[13] Xin Q., Schawrs T. J. E. And Miller E. L., "Disk Infant Mortality in Large Storage Systems", Proc. 13th IEEE International Symposium on Modeling, analysis and Simulation of Computer and Telecommunications Systems (MASCOTS '05), pp. 125-134, Aug. 2005.