Self-Adaptive Disk Arrays

Jehan-François Pâris[1]
Dept. of Computer Science
University of Houston
Houston, TX 77204-3010

Thomas J. E. Schwarz
Dept. of Computer Engineering
Santa Clara University
Santa Clara, CA 95053

Darrell D. E. Long
Dept. of Computer Science
University of California
Santa Cruz, CA 95064

ABSTRACT

We present a disk array organization that adapts itself to successive disk failures. When all disks are operational, all data are replicated on two disks. Whenever a disk fails, the array reorganizes itself, by selecting a disk containing redundant data and replacing these data by their exclusive or (XOR) with the other copy of the data contained on the disk that failed. This will protect the array against any single disk failure until the failed disk gets replaced and the array can revert to its original condition. Hence data will remain protected against the successive failures of up to one half of the original number of disks, provided that no critical disk failure happens while the array is reorganizing itself. As a result, our scheme achieves the same access times as a replicated organization under normal operational conditions while having a much lower likelihood of loosing data under abnormal conditions. In addition it tolerates much longer repair times than static disk arrays/

Keywords: fault-tolerant systems, storage systems, repairable systems, *k*-out-of-*n* systems.

1 INTRODUCTION

Today's disks have mean time to failures of more than ten years, which means that a given disk has a less than ten percent probability of failing during any given year of its useful lifetime. While this reliability level is acceptable for all the applications that only require the storage of a few hundreds of gigabytes of non-critical information over relatively short time intervals, it does not satisfy the needs of applications having to store terabytes of data over many years.

Backups have been the traditional way of protecting data against equipment failures. Unfortunately, they suffer from several important limitations. First, they do not scale well; indeed the amount of time required to make a copy of a large data set can exceed the interval between daily backups. Second, the process is not as trustworthy as it should be due to both human error and the frailty of most recording media. Finally, backup technologies are subject to technical obsolescence, which means that saved data risk becoming unreadable after only ten to twenty years. A much better solution is to introduce redundancy into our storage systems

The two primary ways of introducing that redundancy are replication and *m*-out-of-*n* codes. Both techniques have their advantages and disadvantages. Replication—also known as mirroring—offers the

[1] Contact author (Email: paris@cs.uh.edu; office phone: 713-668-4348; fax: 713-743-3335

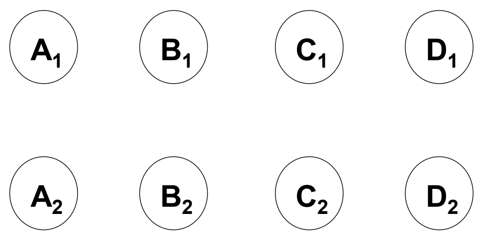

Fig. 1. A small disk array consisting of four pairs of disks with data replicated on each pair of disks.

two main advantages of reducing read access times and having a reasonable update overhead. Identifying failed disks can always be done by detecting which replicas have become unavailable. On the other hand, *m*-out-of-*n* codes provide much higher data survivability. Consider, for instance, the case of a small disk array consisting of eight disks. A replicated organization that maintains two copies of each file on separate disks would protect data against all single disk failures and most double disk failures. A simultaneous failure of three disks would have a bigger impact as it would result in data loss in 43 percent of the cases. This is much worse than an optimal 4-out-of-8 code as that code would protect data in the presence of up to four arbitrary disk failures. In fact, this is such an improbable event that erasure codes tolerating more than two simultaneous failures are never used in actual storage systems.

We propose a self-adaptive disk array organization that combines most advantages of both replication and erasure coding. As long as most disks are operational, it will provide the same read and write access times as a replicated organization. Whenever a disk fails, it will reorganize itself and quickly return to a state where data are again protected against a single failure. As a result, data will remain protected against the consecutive failures of up to one half of the original number of disks, provided that no critical disk failure happens while the array is reorganizing itself. This is a rather unlikely event as the reorganization process will normally take less than a couple of hours.

The remainder of this paper is organized as follows. Section 2 will introduce our self-adaptive disk array organizations. Section 3 will compare the mean times to data loss (MTTDL) achieved by self-adaptive arrays with those achieved by array organizations using replication. Section 4 will review previous work and Section 5 will have our conclusions.

2 OUR APPROACH

Consider for instance the small disk array displayed on Fig. 1. It consists of four pairs of disks with data replicated on each pair of disks. For instance, disks A_1 and A_2 contain the same data set A. Assume now that disk B_1 fails. As a result, only one remaining copy of data set B will remain and the array will become vulnerable to a failure of disk B_2. Waiting for the replacement of disk B_1 is not an attractive option as the process make take several days. To adapt itself to the failure, the array will immediately locate a disk containing data that are replicated elsewhere, say, disk A_1, and replace its contents by the exclusive or (XOR) of data sets A and B thus making the array immune to a single disk failure. Fig. 2 displays the outcome of that reconfiguration. The array can now tolerate any single disk failure. The sole drawback of the process is that accesses to data sets A and B will be now somewhat slower. In particular, updates to these two data sets will be significantly slower as each update will now require one additional read operation. This condition is only temporary as the array will revert to its original condition as soon as the failed disk is replaced.

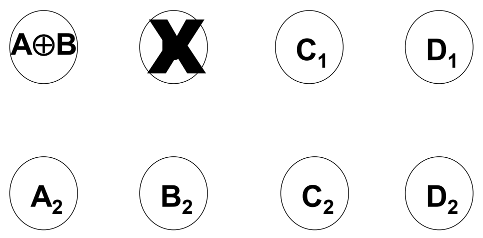

Fig. 2. The same disk array after disk B_1 has failed.

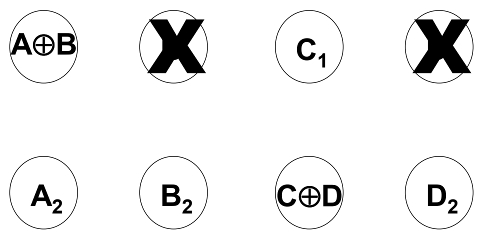

Fig.3. The same disk array after disk D_1 has failed.

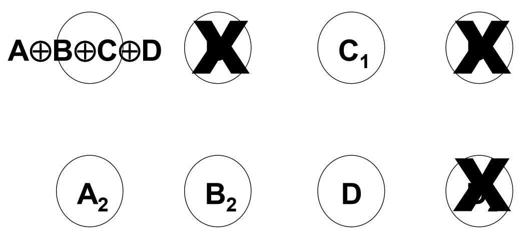

Fig. 4. The same disk array after disk D_2 has failed.

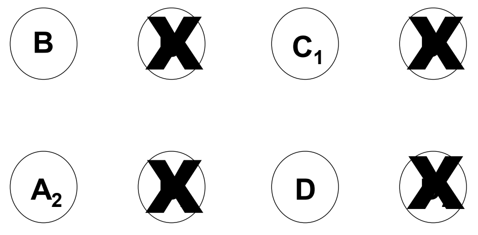

Fig. 5. The same disk array after disk B_2 has failed.

Consider now what would happen if a second disk failed, say, a failure of disk D_1, before disk B_1 was repaired. This second failure would remove one of the two copies of data set *D* and make the array vulnerable to a failure of disk D_2. To return to a safer state, the array will locate a disk containing data that are replicated elsewhere, say, disk C_2, and replace its contents by the exclusive or (XOR) of data sets *C* and *D*. Fig.3 displays the outcome of this reorganization.

Under most circumstances, the two failing disks will be replaced before a third failure could occur. Delays in the repair process and accelerated disk failures resulting from environmental conditions

Assumptions:

disk D is failed data disk

Algorithm:

```
begin
    find disk $E$ having same contents as disk $D$
    if found then
        find a disk $F$ whose contents are replicated on another disk $G$
        if found then
            replace contents ($F$) by contents($E$) XOR contents($F$)
        else
            find parity disk $Z$ whose contents are XORed contents of fewest data disks
            if found then
                replace contents ($Z$) by contents($E$) XOR contents($Z$)
            else
                do nothing
            endif
        endif
    else
        find sufficient set $S$ of disks to reconstitute content($D$)
        if found then
            reconstitute content($D$) on a parity disk $X$ in S
            replace parity disk $X$
        else
            declare failure
        endif
    endif
end
```

Fig. 6. Replacing a failed data disk.

could however result in the occurrence of a third disk failure before disks B_1 and D_1 are replaced. Let us assume that disk D_2 fails this time. Observe that this failure destroys the last copy of data set *D*. The fastest way to reconstitute this data set is to send the contents of disk C_1 to the disk that now contains $C \oplus D$ and to XOR the contents of these two disks *in situ*. While doing that, the array will also send the old contents of the disk to the disk that contains $A \oplus B$ in order to obtain there $A \oplus B \oplus C \oplus D$. As seen on Fig. 4, the disk array now consists of four disks holding data and one parity disk.

Let us now consider for the sake of completeness the rather improbable case of a fourth disk failure occurring before any of the three failed disks can be replaced. Let us assume that disk B_2 fails this time. As Fig. 5 indicates, the sole option left is to reconstitute the contents of the failed disk by XORing the contents of the parity disk ($A \oplus B \oplus C \oplus D$) with those of disks A_2, C_1 and D and store these contents on the former parity disk. This would keep all four data sets available but would leave all of them vulnerable to a single disk failure.

In its essence, our proposal is to let the array adapt itself to the temporary loss of disks by switching to more compact data representations and selecting whenever possible a configuration that

Assumptions:

disk *X* is failed parity disk

Algorithm:

```
begin
    find sufficient set S of disks to reconstitute contents(X)
    if found then
            find parity disk Z whose contents are XORed contents of fewest data disks
            if found then
                replace contents (Z) by contents(Z) XOR contents(X)
            else
                do nothing
            endif
    else
        declare failure
    endif
end
```

Fig. 7. Replacing a failed parity disk.

protects the data against a single disk failure. That process will involve introducing parity disks, merging them and sometimes using them to reconstitute lost data.

Fig. 6 and 7 give a more formal description of our scheme. The first algorithm describes how the array reacts to the loss of a data disk. Two main cases have to be considered depending on whether the contents of the failed disk D can be found on another disk *E*. When this is the case, the array will protect the contents of disk *E* against of a failure of that disk by storing on some disk *F* the XOR of the contents of *E* and the contents of one or more disks. To select this disk *F*, the array will first search for disks whose contents are replicated on some other disk. If it cannot find one, it will then select the parity disk *Z* whose contents are the XORed contents of the fewest data disks. The second case is somewhat more complex. When the contents of the failed disk *D* cannot be found on another disk, the array will attempt to find a sufficient set *S* of disks to reconstitute the contents of the lost disk. If this set exists, it will reconstitute the contents of the lost data disk *D* on a parity disk *X* in S. Once this is done, the array will try to remedy the loss of the parity data on disk *X* by calling the second algorithm.

Our second algorithm describes how the array reacts to the loss of a parity disk *X*. This loss can either be the direct result of a disk failure or a side-effect of the recovery of the contents of a data disk *D*. In either case, the array checks first if it can reconstitute the contents of the failed parity disk *X*. This will be normally possible unless the array has experienced two simultaneous disk failures. If the contents of *X* can be reconstituted, the array will try to XOR them with the contents of a data disk that was replicated somewhere else. If no such data disk exists, the array will XOR the reconstituted contents of *X* with the contents of the parity disk *Z* whose contents are the XORed contents of the fewest data disks.

Space considerations prevent us from discussing how the array will handle disk repairs. In essence, it will attempt to return to its original configuration, first by splitting the parity disks whose contents are the XORed contents of the largest number of parity disks then by replacing the remaining parity disks by pairs of data disks. A more interesting issue is how the self-adapting array would react to the failure of a disk involved in a reconfiguration step. Let us return to our previous example and consider what would happen if disk B_2 failed after disk B_1 failed but before the contents of disk A_1 could be completely replaced by the XOR of the contents of disks A_1 and B_2. Assuming that we do this replacement track by track, disk A_1 would be left in a state where it would contain some of its original tracks and some tracks containing the XOR of the corresponding tracks of disks A_1 and B_2. This means that some but not all the contents of disk B_2 would be recoverable and that some but not all contents of disk A_1 would have become vulnerable to a single disk failure.

3 PERFORMANCE ANALYSIS

Self-adaptive disk arrays occupy a place between replicated disk organizations and organizations using erasure coding. As long as most disks are operational, they provide the same read and write access times as static replicated organizations. In addition, they are more resilient to disk failures. We propose to evaluate this resilience and to compare it with that of static replicated organizations.

Estimating the reliability of a storage system means estimating the probability $R(t)$ that the system will operate correctly over the time interval [0, t] given that it operated correctly at time $t = 0$. Computing that function requires solving a system of linear differential equations, a task that becomes quickly unmanageable as the complexity of the system grows. A simpler option is to focus on the mean time to data loss (MTTDL) of the storage system. This is the approach we will take here.

Our system model consists of a disk array with independent failure modes for each disk. When a disk fails, a repair process is immediately initiated for that disk. Should several disk fail, the repair process will be performed in parallel on those disks. We assume that disk failures are independent events exponentially distributed with mean λ, and that repairs are exponentially distributed with mean μ.

The MTTDL for data replicated on two disks is

$$MTTDL = \frac{3\lambda + \mu}{2\lambda^2}$$

and the corresponding failure rate L is

$$L = \frac{2\lambda^2}{3\lambda + \mu}.$$

Consider an array consisting of n disks with all data replicated on exactly two disks. Since each pair of disk fails in an independent fashion, the global failure rate $L(n)$ of the array will be

$$L(n) = nL = \frac{n\lambda^2}{3\lambda + \mu}$$

and the global mean time to data loss $MTTDL(n)$ will be

$$MTTDL(n) = \frac{3\lambda + \mu}{n\lambda^2}$$

Fig. 8 displays the state transition diagram for a very small self-adaptive array consisting of two pairs of disks with each pair storing two identical replicas of the same data set. Let us assume that disks A_1 and A_2 contain identical copies of data set A while disks B_1 and B_2 store identical copies of data set B. State <2, 2> represents the normal state of the array when its four disks are all operational. A failure of any of these disks, say disk A_1 would bring the array in state <2, 1>. This state is a less than desirable state because the array has now a single copy of data set A on disk A_2. Hence a failure of that disk would result in a data loss.

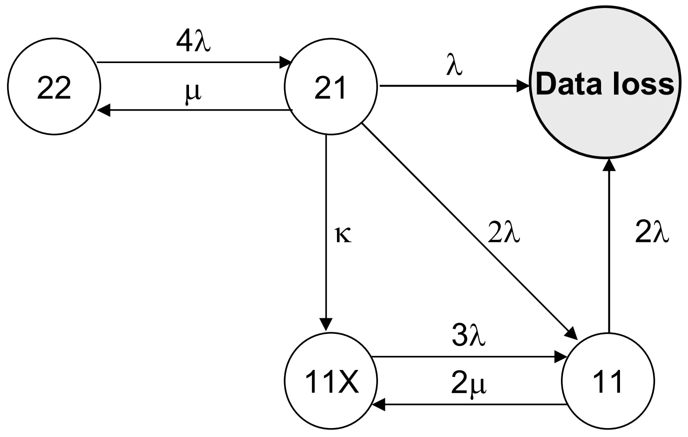

Fig. 8. State transition diagram for a self-adaptive disk array consisting of two pairs of mirrored disks.

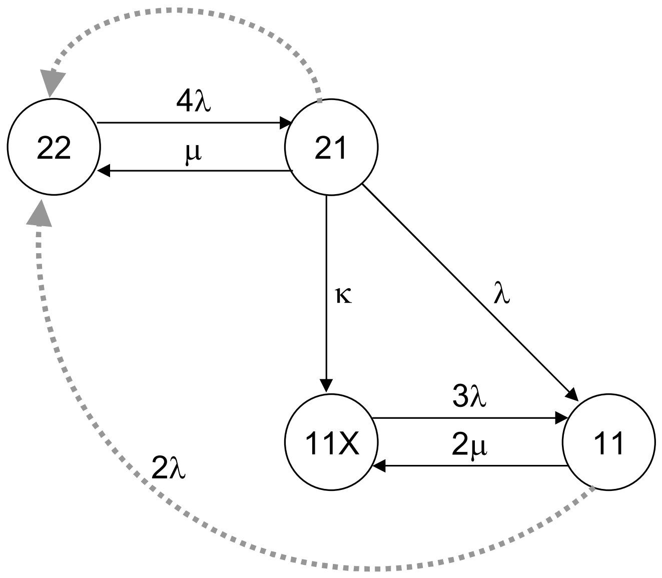

Fig. 9. Modified state transition diagram for the same disk array.

To return to a more resilient state, the array will immediately start replacing the contents of either disk B_1 or disk B_2 with the XOR of data sets A and B, thus bringing the system from state <2, 1> to state <1, 1, X>. We assume that the duration of this self-adaptive process will be exponentially distributed with mean κ. A failure of either of the two redundant disks present in state <2, 1> or a failure of any of the three disks in state <1, 1, X> would leave the array in state <1, 1>, incapable of tolerating any additional disk failure.

Recovery transitions correspond to the repair of one of the disks that failed. They would bring the array first from state <1, 1> to state <1, 1, X> and then from state <1, 1, X> to state <2, 2>. A third recovery transition would bring the array from state <2, 1> to state <2, 2>. It corresponds to situations where the failed disk can be replaced before the self-adaptive process can be completed.

Since data losses are essentially irrecoverable, the state corresponding to such a loss is an absorbing state. Hence a steady state analysis of the array would provide no insight on its performance.

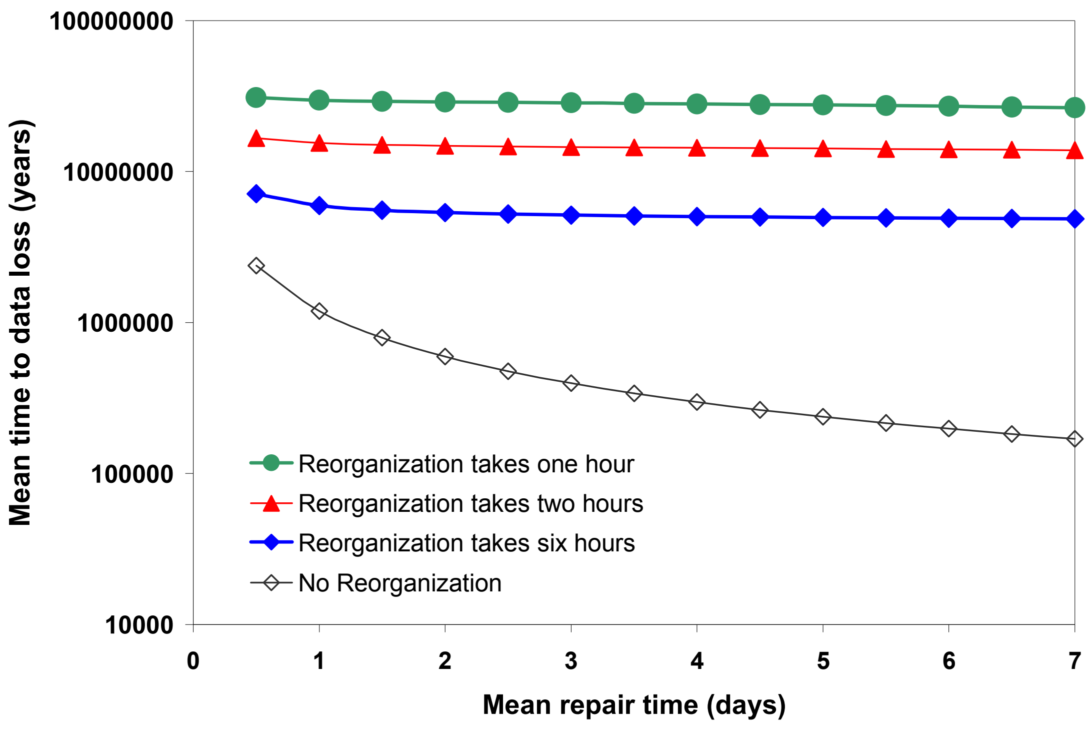

Fig. 9. Mean times to data loss achieved by a self adaptive disk array consisting of two pairs of mirrored disks.

Let us now consider the state transition diagram displayed in Fig, 9. It has the same states and the same transitions as that of Fig.8 but for the two transitions leading to a data loss, which are now redirected to state <2, 2>. This diagram represents what would happen if the array went through continuous cycles during which it would first operate correctly then lose its data and get instantly repaired and reloaded with new data [PS+06]. The corresponding system of equations is

$$\begin{cases} 4\lambda p_{22} = \mu(p_{21} + p_{11X}) + \lambda p_{21} + 2\lambda p_{11}; \\ (3\lambda + \mu + \kappa)p_{21} = 4\lambda p_{22}, \\ (3\lambda + \mu)p_{11X} = \kappa p_{21}, + 2\mu p_{11}, \\ (2\lambda + 2\mu)p_{11} = 3\lambda p_{11X} + 2\lambda p_{21} \end{cases} \tag{1}$$

together with the condition that $p_{22} + p_{21} + p_{11X} + p_{11} = 1$, where p_{ij} represents the steady-state probability of the system being in state $<i, j>$. In addition the rate at which the system will fail before returning to its normal state is

$$L = \lambda p_{21} + 2\lambda p_{11}$$

Solving system (1), we obtain

$$L = \frac{4\lambda^2(9\lambda^2 + 3\kappa\lambda + 3\lambda\mu + \mu^2)}{33\lambda^3 + 13\kappa\lambda^2 + 5\kappa\lambda\mu + 8\lambda\mu^2 + \kappa\mu^2 + \mu^3}.$$

The mean time to data loss of our disk array (MTTDL) is then

$$MTTDL = \frac{1}{L} = \frac{33\lambda^3 + 13\kappa\lambda^2 + 5\kappa\lambda\mu + 8\lambda\mu^2 + \kappa\mu^2 + \mu^3}{4\lambda^2(9\lambda^2 + 3\kappa\lambda + 3\lambda\mu + \mu^2)}.$$

Fig. 10 displays the MTTDLs achieved by the self –adaptive array for selected values of κ and repair times varying between half a day and seven days. We assumed that the disk failure rate l was one failure every one hundred thousand hours, that is, slightly less than one failure every eleven years. MTTDLs are expressed in years.

We can make three main observations. First, our self-adaptive array provides much better MTTDLs than a static array that makes no attempt at reconfiguring itself after disk failures. This is a very significant result as this is a very small array that could only take a single corrective action, namely a transition from state <2, 1> to state <1, 1, X> after a disk failure. Larger self-adaptive disk arrays, such as the array we considered in the previous section, should perform even better as they can take more steps to protect their data after successive disk failures.

Second, these benefits remain evident even when the reconfiguration process takes six hours. Since the reconfiguration process normally consists of reading the whole contents of a disk and XORing these contents with the contents of a second disk, this is clearly an upper bound. In reality we expect most restructuring processes to take between one and two hours depending on the disk bandwidths and capacities.

Finally, the MTTDLs achieved by our self-adaptive organization remain nearly constant over a wide range of disk repair times. This is a significant advantage because fast repair times require maintaining a local pool of spare disks and having maintenance personnel on call 24 hours a day. Since our self-adaptive organization tolerates repair times of up to one week, if not more, it will be cheaper and easier to maintain than a static replicated organization with the same number of disks.

4 PREVIOUS WORK

The idea of creating additional copies of important data in order to increase their chances of survival is likely to be as old as the use of symbolic data representations by mankind and could well have preceded the discovery of writing. Erasure coding appeared first in RAID organizations as $(n - 1)$-out-of-n codes [CL+94, PGK88, SB 92]. RAID level 6 organizations use $(n - 2)$-out-of-n to protect data against double disk failures [BM93].

The HP AutoRAID [WG+96] automatically and transparently manages migration of data blocks between a replicated storage class and a RAID level 5 storage class as access patterns change. This system differs from our proposal in several important aspects. First, its objective is different from ours. AutoRAID attempts 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, data migrations between the two organizations are normally caused by changes in data access patterns rather than by disk failures. Self-adaptive disk arrays only reconfigure themselves in response to disk failures and repairs. Second, AutoRAID actually migrates data between its two storage classes while self-adaptive disk arrays keeps most data sets in place. Finally, the sizes of the transfer units are quite different. A self-adaptive disk array manages its resources at the disk level. The transfer units managed by AutoRAID are *physical extent groups* (PEGs) consisting of at least three *physical extents* (PEXes) whose typical size is a megabyte. Consequently, AutoRAID requires a complex addressing structure to locate these PEXes while a self-adaptive array must only keep track of what happened to the contents of its original disks. Assuming that we have n data sets replicated on $2n$ disks, the actual locations of data sets and their parities can be stored in $2n^2$ bits. In addition, this information is fairly static as it is only updated after a disk failure or a disk repair.

5 CONCLUSIONS

We have presented a disk array organization that adapts itself to successive disk failures. When all disks are operational, all data are replicated on two disks. Whenever a disk fails, the array will immediately reorganize itself and adopt a new configuration that will protect all data against any single disk failure until the failed disk gets replaced and the array can revert to its original condition. Hence data will remain protected against the successive failures of up to one half of the original number of disks, provided that no critical disk failure happens while the array is reorganizing itself. As a result, our scheme achieves the same access times as a replicated organization under normal operational conditions while having a much lower likelihood of loosing data under abnormal conditions. Furthermore it tolerates much longer repair times than static disk arrays.

More work is still needed to investigate larger disk arrays and to devise self-adaptive strategies for disk arrays where some data are more critical than other and thus deserve a higher level of protection. This is the case for archival storage systems implementing chunking to reduce their storage requirements. Chunk-based compression, or chunking, partitions files into variable-size chunks in order to identify identical contents that are shared by several files [MC01]. Chunking can significantly reduce the storage requirements of archival file systems. Unfortunately, it also makes the archive more vulnerable to the loss of chunks that are shared by many files. As a result, these chunks require a higher level or protection than chunks that are only present in a single file [BP+06].

REFERENCES

[AB+02] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer, "Compactly encoding unstructured inputs with differential compression," *Journal of the ACM*, Vol. 49, No. 3, pp. 318–367, May 2002.

[BM93] W. Burkhard and J. Menon, "Disk array storage system reliability." *Proc. 23rd International Symposium on Fault-Tolerant Computing* (FTCS-23), pp. 432-441, 1993.

[BP+06] D. Bhagwat, K. Pollack, D. D. E. Long, E. L. Miller, T. J. Schwarz and J.-F. Pâris. "Providing high reliability in a minimum redundancy archival storage system," *Proc.14th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, *to appear*, Sep. 2006.

[CL+94] P. M. Chen, E. K. Lee, G. A. Gibson, R. Katz, and D. Patterson, "RAID, High-performance, reliable secondary storage." *ACM Computing Surveys*, Vol. 26, No. 2, pp. 145–185, 1994.

[GP93] G. A. Gibson and D. A. Patterson, "Designing disk arrays for high data reliability." *Journal of Parallel and Distributed Computing*, Vol. 17(1/2) p. 4, 1993

[GT90] R. Geist, K. S. Trivedi, "Reliability estimation of fault-tolerant systems: Tools and Techniques," *Computer*, Vol. 23, No. 7, pp. 52–61, July 1990.

[GW+94] G. Ganger, B. Worthington, R. Hou, Y. Patt, "Disk arrays: High-performance, high-reliability storage subsystems." *IEEE Computer* vol. 27(3), p. 30–36. 1994.

[LL90] K. Lu, C. Liew, "Analysis and applications of r-for N protection system. *Global Telecommunications Conference, 1990, and Exhibition.*

[MC+01 A, Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system." *Proc. 18th Symposium on Operating Systems Principles*, pp. 174-187, 2001.

[MT92] M. Malhotra and K. Trivedi, "Reliability modeling of disk array systems." *Proc. 6th*

International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Edinburgh, 1992.

[PGK88] D. A. Patterson, G. A. Gibson, and R. H. Katz. “A case for redundant arrays of inexpensive disks (RAID),” *Proc. SIGMOD 1988 International Conference on Data Management,* pp. 109–116, June 1988.

[PS+06] J.-F. Pâris, T. J. E. Schwarz and D. D. E. Long. *Evaluating the reliability of storage systems*, Technical Report UH-CS-06-08, Department of Computer Science, University of Houston, June 2006.

[SB92] T. J. E. Schwarz and W. A. Burkhard. “RAID organization and performance,” *Proc. 12th International Conference on Distributed Computing Systems*, pp. 318–325, June 1992.

[SB95] T. J. E. Schwarz, and W. A. Burkhard, “Reliability and performance of RAIDs,” *IEEE Transactions on Magnetics*, 1995, Vol 31(2), pp. 1161–1166. March 1995.

[SG+89] M. Schulze, G. Gibson, R. Katz and D. Patterson. “How reliable is a RAID?” *Proc. Spring COMPCON 89 Conference*, pp. 118–123, March 1989.

[XM+03] Q. Xin, E. L. Miller, T. J. E. Schwarz, D. D. E. Long, S. A.Brandt, and W. Litwin, “Reliability mechanisms for very large storage systems,” *Proc. 20th IEEE Conference on Mass Storage Systems and Technologies*, pp/ 146–156, Apr. 2003.

[XSM05] Q. Xin, T. J. E. Schwarz and E. L. Miller, “Disk infant mortality in large storage systems,” *Proc. 13th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, August 2005.