1997 Cluster Computing Conference, GA, March 9-11, 1997.

A Disk Architecture for Large Clusters of Workstations

Jehan-François Pâris

Department of Computer Science
University of Houston
Houston, TX 77204-3475
paris@cs.uh.edu

ABSTRACT

Today and tomorrow's computing clusters are likely to have one hard drive at every node. Hence large clusters need to be protected against disk failures. Extant solutions either require more than one drive per node or destroy the natural locality of disk accesses resulting from the decomposition of the problem space. We have presented a method avoiding these two pitfalls and showed how it can be tuned to provide either inexpensive reads or inexpensive writes.

A series of discrete event simulations has shown that the reliability of very large clusters managed by our method is quite satisfactory and better indeed than that of reliable arrays of distributed disks.

Keywords: computer clusters, file systems, fault-tolerant systems, RAID systems

1. Introduction

One of the best ways to alleviate the I/O bottleneck in computing clusters is to allocate to each computing node sufficient disk resources to let it perform locally most of its disk accesses. This solution is particularly attractive because it takes advantage of the natural locality of disk accesses resulting from the decomposition of the physical problem space among the nodes. It presents nevertheless the drawback of making the whole cluster much more susceptible to disk failures.

Due to their mechanical nature, disk drives are one of the least reliable components of a computer system with mean time to fail varying between 8,000 and 800,000 hours [3, 4]. Hence, unless some redundancy is introduced, attaching one disk drive to each computing node of a large cluster will considerably lower the overall reliability of the system. One possible solution is to attach a Reliable Array of Independent Disks (RAID)[1, 7] on each computing node. This would unfortunately require having at least two, three or four disks per computing node. Another solution would be to organize the cluster nodes themselves into one of more *disk striping* units implementing the same

architecture as a RAID but without a shared controller [2, 5]. The two major advantages of the solution are its low space overhead and its very good reliability. Organizing the cluster nodes into striping units would however destroy the locality of disk accesses and consequently increase the network traffic among the nodes.

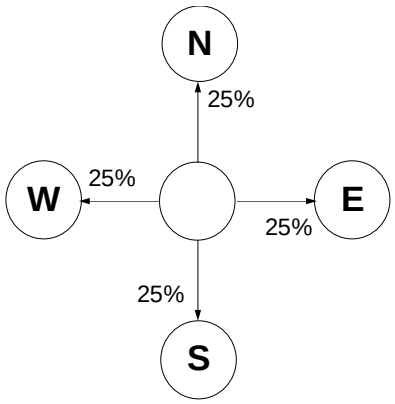

Figure 1: A computing node exporting copies of its critical data to its four neighbors

We propose a method for increasing the reliability of large clusters of workstations that preserves the locality of disk accesses. In its simplest form, it requires more disk space than RAID-like organizations. We present however two possible methods to eliminate this problem, one providing fast reads and the other fast writes.

2. Our Solution

The solution we propose is based on the observation that some of the data stored on disk are not critical to the pursuit of the computation because they could be easily reconstituted. Hence they do not need to be backed up. We will replicate only the data that are critical to the computation. We also recognize that each computing node within a cluster has one or more *neighboring nodes*. These neighboring nodes can sometimes be defined by the topology of the network; one could then call them *physical neighbors*. More generally they can be defined by the decomposition of the problem space among the computing nodes; we will call these neighboring nodes *logical neighbors*. What characterizes both physical and logical neighbors is the fact that the nodes are much more likely to exchange data with their neighbors than with the other nodes within the cluster. We propose to use these neighbors to store redundant copies of the critical data stored by each node.

For instance, a computing node with four neighbors such as the node on Figure 1 would export copies of 25 percent of its critical data to each of its neighbors.

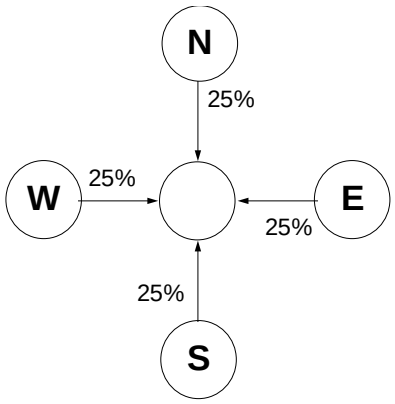

Figure 2: The same node receiving copies of the critical data of its four neighbors

Our solution will introduce two types of overheads. First each computing node will have to process the backup data blocks of all its neighbors. Second it will have to allocate disk space for them. We do not believe that the processing overhead is a major problem because the likelihood of finding spare processor cycles is bound to increase thanks to the ever increasing speeds of processors. This observation was recently confirmed by Kotz and Cai [5]. The space overhead issue could be more critical but there are at least two potential approaches to reduce it.

Consider the case of a computing node of figure 2, which has four neighbors, respectively named *N*, *E*, *S* and *W*. Let us assume for the sake of simplicity that its backup areas for the four neighbors are all equal in size. Instead of keeping these four areas separate, the node could:

(a) combine all four areas into a single area *A* where the *exclusive or* (XOR) of the contents of all four areas would be stored; hence we have $A = N \oplus E \oplus S \oplus W$, or

(b) maintain one area A' with the XOR of the backup blocks of *N* and *S* and another area A'' with the XOR of the backup blocks of *E* and *W*; thus we have $A' = N \oplus S$ and $A'' = E \oplus W$.

As a result, the storage overhead can be reduced to 25 percent of the backed up data blocks when we store the XOR of the four backup areas or to 50 percent when an XOR of *N* and *S* and an XOR of *E* and *W* are maintained.

After a drive failure, each neighbor of the node must first reconstruct its share of the backup data. Then each neighbor must find backup nodes for (a) the fraction of its own data their were backed up by disk that failed, and (b) its share of the data that were on the disk that failed. Once recovery is completed, the processor with the failed disk can use its neighbors as file servers and all data on disk have again back ups.

One major disadvantage of the approach is that writes have now become more expensive. In order to compute the new value of the backup area after a data block d has been modified, we need both the old value of the data block d_{old} and the old value of the corresponding block of the backup area p_{old} as p_{new} is given by:

$$p_{new} = p_{old} \oplus d_{old} \oplus d_{new}.$$

Hence one write would require two reads to obtain the current values of d_{old} and p_{old} in addition to the two writes necessary to store d_{new} and p_{new}.

In many applications the critical data that need to be saved essentially consist of snapshots that record the state of the computation at a given time and can be used to restart the computation after a malfunction. This is to say that each snapshot is written once but is very unlikely to be read before being overwritten by another snapshot. An alternative approach is then possible. It requires each node to distribute the copies of its critical data among its neighbors in a circular fashion so that if a node has m neighbors numbered from 0 to m-1, the backup copy of block d_k will always be stored on the neighbor number k mod m. As a result any set of m consecutive data blocks will have their backup copies dispersed among all the m neighboring node. If we are willing to accept temporarily inconsistencies of the checkpoint data, while the checkpoint is constructed, we can:

(a) delay the writes until a full set of data blocks numbered from $k.m$ to $km+m$-1 for some positive integer k are ready to written,

(b) distribute the backup copies of these m blocks among the m neighboring nodes, and

(c) store on the node itself the XOR of the blocks.

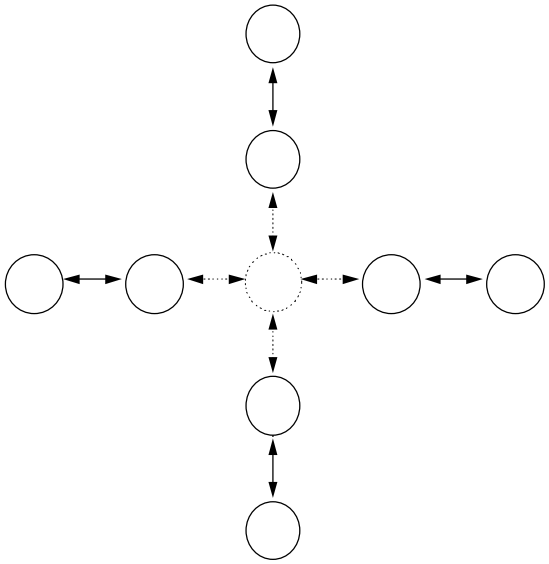

Figure 3: Nodes involved in the recovery of a disk drive that was backed up using partial XORs

With this method, the total cost of a write is one message sent to a neighbor and 1+1/*m* disk accesses. Read operations have in turn become more expensive because they now require the exchange of two messages (a request and a reply) with a neighbor.

3. Reliability Analysis

In addition to making writes more costly, our first approach also presents a potential reliability risk because recovering the data lost due to a drive failure will now require the cooperation of some or all of the neighbors of the nodes holding the back-up data. Consider for instance, the case of a cluster where each node has four neighbors and the back-up data were compressed using partial XORing. As figure 3 shows, the failure of the disk drive of any given node would require each of the four neighbors to cooperate with four of their own neighbors to reconstruct the lost data. If any of the eight disk drives corresponding to these eight nodes failed before this task is completed, some data would be permanently lost and the whole cluster would fail. We will refer to this type of global failures as *protocol failures*. The situation is even worse when the back-up data rare compressed using full XORing. As figure 4 indicates, a total of twelve nodes need now to cooperate to reconstruct the missing data. Moreover, the four nodes on the two diagonals are becoming bottlenecks since they need to send their data blocks to two of the neighbors of the site.

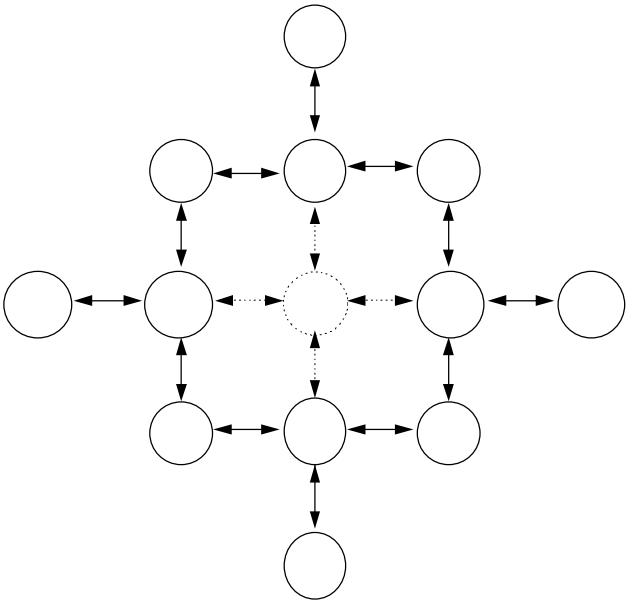

Figure 4: Nodes involved in the recovery of a disk drive that was backed up using full XORs

To evaluate the importance of these protocol failures, we conducted a study of the level of reliability afforded by our first approach and compared it with that achieved by organizing the cluster into a set of redundant arrays of distributed disks (RADD) [8].

Definition. *The reliability R(t) of a system is the probability that it will operate correctly over [0, t) given that it was operating correctly at t = 0.*

Reliabilities can be predicted either through Markov or through discrete event simulation. We selected the latter method because we wanted to evaluate the reliability of very large clusters of workstations and the systems of differential linear equations we would have had to solve would have been untractable.

We assumed that disk failures would be independent events exponentially distributed with a mean time to fail (MTTF) of 800,000 hours. We also assumed that the time to recover from a disk failure would be essentially equal to the time required to read and transfer the information necessary to reconstruct the lost data. For RADD architectures, we assumed this time to be equal to one hour.

Since reliabilities are difficult to evaluate through discrete event simulation, we measured instead the MTTFs. For each run, we observed 10,000 independent failures and used them to compute 95

percent confidence intervals of the MTTF. In many systems with spares, the MTTF is linked with the reliability by the approximate relation $R(t) \cong \exp(-\frac{t}{MTTF})$ [9].

RADD Organization

We assumed in our simulation that the cluster would be subdivided into independent groups of *n* workstations that would each implement a RAID-like architecture. In addition to the MTTFs, we also measured the percentage of failures that would correspond to protocol failures, that is the percentage of failures occurring while the RADD was recovering from a previous failure and had not yet reconstructed all the lost data.

Table 1: MTTFs for RADDs assuming we need 80 percent of the disk drives

RADD Size	Mean Time To Fail	Protocol Failures
5	362,055 h ± 5,103	0%
10	267,190 h ± 3,063	0.01%

Table 1 displays the MTTFs and the percentage of protocol failures for RADDs of either five of ten nodes assuming that a RADD would fail as soon as more than 20 percent of the drives fail. Being quite intrigued by the very low percentage of protocol failures, we repeated the simulations assuming this time that a RADD would continue to remain operational as long as one of its drives was operational. While being totally unrealistic, this hypothesis was presenting an interesting limit case for the percentage of protocol failures.

Table 2: MTTFs for RADDs assuming we need only one disk drive

RADD Size	Mean Time To Fail	Protocol Failures
5	1,824,482 h ± 19,168	0%
10	2,350,115 h ± 19,670	0.02%

As table 2 shows, these percentages remain extremely low despite the fact we are in presence of a limit case. The very high MTTFs were included as a curiosity but are not meaningful.

It should be pointed out that these values only apply to the individual RADDs into which the cluster was subdivided. Since a failure of any of these clusters would result in a failure of the whole

cluster, the MTTF of the cluster can be obtained by dividing the MTTF of a RADD by the number *n* of RADDs in the cluster.

XORed backups:

We focused our simulation study on very large clusters using fully XORed because these configurations were the most susceptible to protocol failures. We assumed that all nodes would have four neighbors and that the busiest nodes would have to read and forward 50 percent of the data. Hence the total recovery time is only divided by two.

To further assess the impact of the recovery time on the MTTF of the cluster, we ran all our simulations for read times equal to zero hour, one hour and one day. While the first and the last of these three values are totally unrealistic, they respectively provide good upper and lower bounds for the true read times.

Table 3: MTTFs for a cluster of 512 nodes assuming we need 80 percent of the disk drives

Time to Read Data	Mean Time To Fail	Protocol Failures
0 h	181,506 h ± 350	0%
1 h	181,383 h ± 353	0.11%
1 day	179,883 h ± 448	1.72%

Table 3: MTTFs for a cluster of 1024 nodes assuming we need 80 percent of the disk drives

Time to Read Data	Mean Time To Fail	Protocol Failures
0 h	179,628 h ± 248	0%
1 h	179,417 h ± 267	0.19%
1 day	176, 465 h ± 459	3.32%

Table 3 and 4 contain the MTTFs and the percentage of protocol failures for clusters with respectively 512 and 1,024 nodes. As one can see, the percentage of protocol failures increases with the duration of the recovery period while the MTTF does not seem to be very significantly affected by either the number of nodes in the cluster or the duration of the recovery period. As for the case of RADDs, we wanted to know how the percentage of protocol failures would evolve if the cluster could remain operational with less that 80 percent of its drives operational. We repeated thus the

simulations assuming that the cluster would continue to remain operational as long as one of its drives was operational.

Table5: MTTFs for a cluster of 512 nodes assuming we need one disk drive

Time to Read Data	Mean Time To Fail	Protocol Failures
0 h	5,462,641 h ± 20,255	0%
1 h	5,438,248 h ± 21,353	0.55%
1 day	5,024,328 h ± 32,640	8.97%

Table 6: MTTFs for a cluster of1,024 nodes assuming we need one disk drive

Time to Read Data	Mean Time To Fail	Protocol Failures
0 h	6,002,005 h ± 20,013	0%
1 h	5,960.546 h ± 22,017	0.82%
1 day	5,121,407 h ± 42,750	16.65%

These results are summarized in tables 5 and 6. It should first be noted that the MTTFs are unrealistically high and can only be compared against themselves. We observe however that the percentage of protocol failures remain very low for all reasonable values of the time needed to reconstruct the data lost due to a site failure.

We can definitively conclude from these simulation that the number of nodes involved in the reconstruction of the lost data has very little impact on the MTTF of the system for all reasonable values of the reconstruction time. We need however to handle more cautiously these MTTFs. It is clear that the main factor affecting the MTTF of a cluster of workstation is the percentage of drive failures it can tolerate. We can therefore suggest that it will always be a good idea to equip the workstations with large disk drives having a sufficient number of free disk space to allow for the fast reconstruction of the critical data lost due to a drive failures. We have however several reasons to remain skeptical about our estimates of the MTTFs themselves. First, we have not taken into account the fact that drive failures are not independent events because the environmental factors that have caused one disk failure (power spike, vibrations and so forth) are likely to affect the reliabilities of the other drives in the cluster. Second, we assumed that nothing could hamper the progress of the reconstruction process but the failure of one of the involved drives.

4. Conclusion

Today and tomorrow's computing clusters are likely to have one hard drive at every node. Hence large clusters need to be protected against disk failures. Extant solutions either require more than one drive per node or destroy the natural locality of disk accesses resulting from the decomposition of the problem space. We have presented a method avoiding these two pitfalls and showed how it can be tuned to provide either inexpensive reads or inexpensive writes.

A series of discrete event simulations has shown that the reliability of very large clusters managed by our method is quite satisfactory and better indeed than that of reliable arrays of distributed disks. More work is still needed to obtain more realistic estimates of the mean time to fail of the clusters.

Acknowledgments

We wish to thank Mr. Edward Robinson for his numerous comments and suggestions. This work was supported in part by the CENJU-3 Project at the University of Houston sponsored by NECSYL.

References

[1] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz and D. A. Patterson, "RAID: high-performance, reliable secondary storage," *ACM Computing Surveys*, Vol. 26. No. 2 (1994), pp. 145-185.

[2] A. L. Drapeau, K. W. Shiriff, J. H. Hartman, E. L. Miller, S. Seshan, R. H. Katz, K. Lutz, D. A. Patterson, E. K. Lee, P. M. Chen and G. A. Gibson, "RAID-II: a high bandwidth network file server," *Proc. 21st Annual International Symposium on Computer Architecture,* (1994), pp. 234–244.

[3] T. J. Glover and M. M. Young, *Pocket PCRef*. (4th edition), Sequoia Publishing Inc. (1994)

[4] IBM Corp., "MTBF–A measure of OEM disk drive reliability," WWW posting (URL `http://www.almaden.ibm.com/storage/oem/tech/mtbf.htm`) (1995).

[5] D. Kotz and T. Cai, "Exploring the uses of I/O nodes for computation in a MIMD multiprocessor," Technical Report TR 95-253, Dartmouth College, 1995.

[6] D. D. E. Long, B. R. Montague and L.-F. Cabrera, "SWIFT/RAID: a distributed raid system," *Computer Systems*, Vol. 7., No. 3 (1994), pp. 333–359.

[7] D. Patterson, G. Gibson and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks," *Proc. 1988 SIGMOD Conference*. (1988), pp. 109–116

[8] M. Stonebraker and G. A. Schloss, ``Distributed RAID—A new multiple copy algorithm," *Proc. 6th International Conference on Data Engineering* (1990), pp. 91–95

[9] M. A. M^{c}Gregor, ``Approximation formulas for reliability with repair," *IEEE Transactions. on Reliability*, Vol. R-12 (1963), pp. 64–92.