Parity Redundancy Strategies in a Large Scale Distributed Storage System

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut
Storrs, CT 06269-1157
jchandy@uconn.edu
tel +1-860-486-5047

Abstract

With the deployment of larger and larger distributed storage systems, data reliability becomes more and more of a concern. In particular, redundancy techniques that may have been appropriate in small-scale storage systems and disk arrays may not be sufficient when applied to larger scale systems. We propose a new mechanism called delayed parity generation with active data replication (DPGADR) to maintain high reliability in a large scale distributed storage system without sacrificing fault-free performance.

1 Introduction

Data creation and consumption has increased significantly in recent years and studies have suggested that the amount of information stored digitally will continue to double every year for the foreseeable future. This increasing need for information storage leads to a corresponding need for high-performance and reliable storage systems. Single storage nodes can not provide the required storage capacity or scalability. Thus, in an effort to satisfy this need, there has been significant work in the area of distributed storage systems where storage nodes are aggregated together into a larger cohesive storage system. These include distributing data amongst shared disks [1, 6, 12], dedicated storage nodes [8], clustered servers [4], or the clients themselves [2, 7]. It is not unreasonable to expect systems with petabytes of data distributed across thousands of nodes in these distributed storage systems.

However, as we increase the number of nodes, the reliability of the entire system decreases correspondingly unless steps are taken to introduce some form of redundancy into the system. In this paper, we discuss redundancy mechanisms to provide high reliability without sacrificing fault free performance. For the purposes of this paper, we refer to individual storage devices in the distributed storage system as nodes - whether they be disks in a shared disk SAN, servers in a clustered server, or OBSDs in a network attached disk system. The techniques and strategies apply with slight variations to all implementations.

Data redundancy in most disk array subsystems is typically provided by using RAID. These same techniques used at the disk level can also be used at the node level. Mirroring,

or RAID1, entails replication of the data on multiple nodes. Parity striping, or RAID5, involves spreading data along with parity across multiple nodes. Choosing which RAID level to use is typically determined by cost and application requirements. At the disk array level, the redundancy choice is usually RAID5 as it provides excellent availability, little storage overhead, and adequate performance.

However, with a large scale distributed system, the choice is not so clear. RAID5 no longer provides sufficient reliability since a thousand node system could exhibit MTTFs of a few years. A mirrored distributed storage system, however, can have an MTTF of several decades. In addition, RAID5 suffers from the well-known write penalty whereby parity updates require two extra reads to generate the parity. Techniques to overcome this problem in a disk subsystem such as the use of non-volatile caches can not be used in a distributed system. Moreover, the cost of the write penalty is more significant because of the high latency costs inherent in network communications.

Because of the limitations of parity striping, in many distributed storage systems, replication or mirroring is the preferred strategy for redundancy [3, 15]. In addition, replication allows widely distributed clients and nodes to take advantage of locality and retrieve data from the closest storage node. However, the cost of mirroring is the 100% storage overhead.

In this paper, we present methods to achieve the low storage overhead of parity striping but retaining the performance and reliability characteristics of mirroring. In particular, we discuss the use of delayed parity generation to improve parity striping performance.

2 Delayed Parity Generation with Active Data Replication

The concept of delayed parity generation with active data replication (DPGADR) is based on reducing the number of accesses required to generate parity in a RAID5 system. In a standard RAID5 disk array, the array controller must read old values from both the data and parity disks and then write the new data back to the data disk and the XOR'ed parity result back to the parity disk. This results in a total of 4 disk accesses (potentially 2 if the parity and data reads had been cached). In a DPGADR system, we delay the generation of the parity, and thus do not require the reading of old data and parity or the writing the parity result. However, without parity generation, the system is potentially compromised in the event of failure. To address this, we replicate the new data to a *replication node* that is not part of the RAIDed redundancy group. We have reduced the number of accesses to just 2 writes - both of which can be proceed in parallel. Figure 1 shows the data distribution in a DPGADR system. In order to distribute the load, the replication node can be rotated across the redundancy group.

Each replication node keeps a map relating the actual block location to the active data locations kept on the node. Thus, the client is not responsible for identifying the block location for the replicated data on the replication node. The client can use a simple hash algorithm to map from block ID to replication node, and it is then the replication node's responsibility to allocate storage space locally.

On the surface, this DPGADR scheme appears to be simply mirroring of data. However, we do not maintain the mirrored data on the replication node in perpetuity. In order to avoid replicating all data writes, the replication node will periodically generate the parity for any

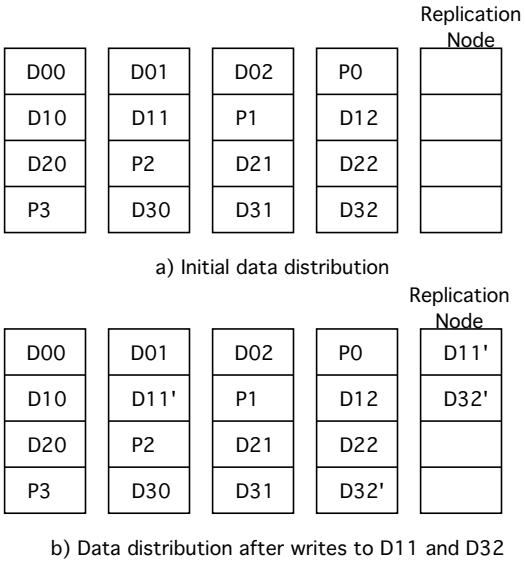

Figure 1: DPGADR data distribution.

blocks that it contains and then flush these blocks from its data store. Because of this periodic data flushing, the replication node is in effect a cache of actively used data blocks.

Parity generation from the replication node is not a trivial problem as the replication node may not have the required data to generate the parity. In such a case, the node must retrieve the stripe data from the relevant nodes before generating the parity. However, in general, it is likely that the active data set on the replication node will contain all or most of the data blocks in a particular stripe because of locality and small working set sizes [11]. We would like to keep the stripe length small so as to make sure that the entire stripe is in the active data set on the replication node. This is also desirable for reliability reasons since it reduces the size of the redundancy group. A reasonable stripe length is 5 nodes, thus requiring 200 stripes to span a 1000 node system.

3 Reliability

As mentioned above, as we increase the number of nodes in a large scale storage system, RAID5 parity striping no longer provides sufficient redundancy to give adequate system reliability. The use of DPGADR can improve system reliability significantly. Since recently used data is copied to a replication node, the system exhibits mean time to data loss (MTTDL) characteristics near to that of a mirrored system. Figure 2 illustrates how the system can tolerate more than one failure in a redundancy set and still recover the most recent data. Even though two nodes have failed, the active data blocks, *D11'* and *D32'*, are still available from the replication node. In fact if all the nodes except for the replication node fail, the DPGADR method allows for the recovery of all active data. While the replication node can prevent loss of active data in the presence of multiple failures, inactive data can still be lost if there is more than one fault. To prevent data loss in such a scenario, we require that the system have judicious backup procedures so that all inactive data is present on backup media. The replication node must be large enough to accommodate all active

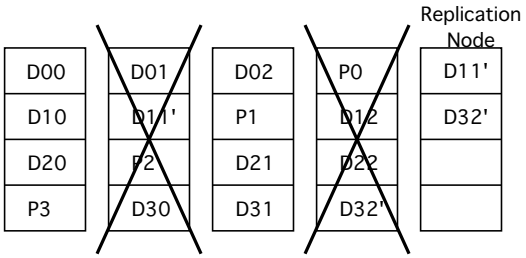

Figure 2: DPGADR failure scenario.

data between backups. This need not be that large as the working set size of a storage system over a 24 hour period is typically around 5% of the entire storage space [11]. Thus, it is sufficient to have one replication node for every 20 data nodes and using daily backups to prevent data loss of inactive data in the presence of dual faults. We also suspend access to the DPGADR group once a second fault has been recorded. This prevents invalid data being read from inactive data blocks. Thus, the DPGADR method prevents data loss in most dual fault cases but system availability is the same as a RAID system because of the access blocking due to a second fault.

We can develop a model for the MTTDL and availability of a DPGADR system using a similar analysis methodology to that outlined in [10]. The nodes are assumed to have independent and exponential failure rates. We assume d nodes per parity group, one redundancy node per n_G parity groups, and the mean time to failure of each node is $MTTF_{Node}$ The MTTDL for a DPGADR group is:

$$MTTDL_{DPGADR} = \frac{\frac{MTTF_{Node}}{n_G(d+1)+1}}{Pr[\text{data loss failure during repair time}]} \tag{1}$$

Data loss during the repair time of the failed node can happen in three cases: 1) the first failed node was the replication node and any other node fails, 2) the first failed node was not the replication node and the replication node fails, and 3) the failed node was not the replication node and two non-replication nodes in the same parity group fail. Thus, the probability of data loss causing failure during the repair time is as follows:

$$\begin{aligned} Pr[\text{data loss failure during repair time}] = \\ Pr[\text{first failed node was a replication node}]p_f \\ + \quad (1 - Pr[\text{first failed node was a replication node}])(p_{rf} + p_{2f}) \end{aligned} \tag{2}$$

where p_f is the probability that any one of the remaining $n_G(d+1)$ nodes fails during the repair time, p_{rf} is the probability that the replication node fails during the repair time, and p_{2f} is the probability that 2 non-replication nodes from the same parity group fail during the repair time. The derivation of p_f is straightforward. If we define the mean time to repair the node as $MTTR_{Node}$, then assuming exponential failure rates,

$$p_f \approx \frac{\frac{MTTR_{Node}}{n_G(d+1)}}{MTTF_{Node}} \tag{3}$$

Configuration	MTTDL (years)	Redundancy overhead
RAID5 (d=5)	7.9	200 nodes
Mirror	23.8	1000 nodes
DPGADR (d=5,n_G=4)	39.6	250 nodes

Table 1: MTTDL and Overhead for a 1000 data node system. ($MTTF_{Node}$ = 100000 hours and $MTTR_{Node}$ = 24 hours

when $MTTF_{Node} \gg MTTR_{Node}$. Similarly, p_{rf} is equal to $\frac{MTTR_{Node}}{MTTF_{Node}}$. p_{2f} can be expressed as follows:

$$p_{2f} = \binom{d}{2}(\frac{MTTR_{Node}}{MTTF_{Node}})^2(1-\frac{MTTR_{Node}}{MTTF_{Node}})^{d-1} \approx \frac{d(d-1)}{2}(\frac{MTTF_{Node}}{MTTR_{Node}})^2 \quad (4)$$

Substituting into Eqs. 1 and 2, we arrive at:

$$MTTDL_{DPGADR} = \frac{\frac{MTTF_{Node}}{n_G(d+1)+1}}{\frac{1}{n_G(d+1)+1}[\frac{MTTR_{Node}}{n_G(d+1)MTTF_{Node}}] + \frac{n_G(d+1)}{n_G(d+1)+1}[\frac{MTTR_{Node}}{MTTF_{Node}} + \frac{d(d-1)}{2}(\frac{MTTR_{Node}}{MTTF_{Node}})^2]}$$
$$\approx \frac{MTTF^2_{Node}}{n_G(d+1)MTTR_{Node}} \quad (5)$$

In a large system with n_S DPGADR groups, the $MTTDL$ is $\frac{MTTF^2_{Node}}{n_S n_G(d+1)MTTR_{Node}}$. If we define D as the total number of data nodes in the system, i.e. $n_S n_G d$, we can rewrite $MTTDL$ as $\frac{MTTF^2_{Node}}{D(1+\frac{1}{d})MTTR_{Node}}$. The redundancy overhead to support parity and replication is $\frac{D}{d} + \frac{D}{n_G\ d}$. By comparison, a mirrored system with D data nodes has a MTTDL of $\frac{MTTF^2_{Node}}{2D\ MTTR_{Node}}$ with an overhead of D nodes, and a RAID5 system has an MTTDL of $\frac{MTTF^2_{Node}}{D(d+1)\ MTTR_{Node}}$ and an overhead of $\frac{D}{d}$ nodes. The DPGADR system actually has better MTTDL times than a mirrored system with significantly less redundancy overhead. Table 1 shows MTTDL and overhead numbers for a 1000 data node system. Note that while the MTTDL of a DPGADR system is better than a mirrored system, the availability, i.e. the probability that the system is available for use, is more like a RAID5 system. This is because after a second failure, the system is suspended until the repair is complete.

4 Related Work

Xin et al [15], have proposed three different large storage system redundancy architectures with two fast recovery mechanisms: fast mirroring copy (FMC) and lazy parity backup (LPB). The LPB method is similar to the DPGADR scheme in that parity calculation is delayed. However, it relies on a RAID5 redundancy set to be completely mirrored at a

greater than 100% storage overhead. The DPGADR scheme requires significantly less overhead since only actively used data is replicated.

Other related work is in the area of RAID5 disk arrays and of particular interest are parity logging [13], data logging [5], and hot mirroring [9, 14].

The parity logging technique eliminates the need for parity disk accesses by caching the partial parity formed from the old and new data in non-volatile memory at the controller. The partial parity can then be periodically flushed to a log disk, which can then be cleaned out at a later time to generate the actual parity disk data. This process reduces the number of disk accesses from 4 to 2 and clearly, this reduction in accesses will greatly speed up the performance of writes in a RAID system. Parity logging, however, is not practical in a distributed system because of the need to cache data in non-volatile memory. It is not reasonable to expect distributed system clients to have non-volatile memory available. Moreover, the management of the cache across multiple clients can be problematic. The DPGADR system does not require non-volatility at the client since all data is pushed out to the replication node immediately. Data logging is similar to DPGADR except that it performs an old data read and stores that to the data log as well. This requires an extra disk access and the maintenance of log maps requires non-volatile memory at the clients as well.

Hot mirroring [9] and AutoRAID [14] are similar techniques that attempt to move actively used data to mirrored regions of the array and less frequently used data to parity logged regions (hot mirroring) or parity striped regions (AutoRAID). These system require a background process that evaluates the "hotness" of data and then moves them to or from mirrored regions as required. If a data block is in the parity striped region, it will remain there until a background process has tagged it as hot even if it is experiencing high activity. DPGADR systems, however, dynamically adjust to the activity of the data since the latest data is always pushed to the mirrored region, i.e. the replication node.

5 Conclusions

In this paper, we have described a delayed parity construction mechanism called DPGADR that allows parity striping to be used on large scale distributed storage systems without suffering from the small write performance penalty. Compared to mirroring it can reduce the storage overhead from 100% to less than 20% and compared to parity striping it can reduce small write accesses to just two parallel accesses. Because of redundancy in the active data replication node, the overall system reliability is better than mirroring for significantly less overhead.

References

[1] D. Anderson and J. Chase. Failure-atomic file access in the Slice interposed network storage system. *Cluster Computing*, 5(4):411–419, Oct. 2002.

[2] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proceedings of the Symposium on Operating System Principles*, pages 109–126, Dec. 1995.

[3] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 34–43, June 2000.

[4] J. D. Bright and J. A. Chandy. A scalable architecture for clustered network attached storage. In *Proceedings of the IEEE/NASA Goddard Symposium on Mass Storage Systems and Technologies*, pages 196–206, Apr. 2003.

[5] E. Gabber and H. F. Korth. Data logging: A method for efficient data updates in constantly active RAIDs. In *Proceedings of the International Conference on Data Engineering*, pages 144–153, 1998.

[6] G. A. Gibson and R. Van Meter. Network attached storage architecture. *Commun. ACM*, 43(11):37–45, Nov. 2000.

[7] J. H. Hartman and J. K. Ousterhout. Zebra: A striped network file system. In *Proceedings of the USENIX 1992 Workshop on File Systems*, May 1992.

[8] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, Oct. 1996.

[9] K. Mogi and M. Kitsuregawa. Hot mirroring: A method of hiding parity update penalty and degradation during rebuilds for RAID5. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 183–194, June 1996.

[10] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 109–116, June 1988.

[11] C. Ruemmler and J. Wilkes. A trace-driven analysis of working set sizes. Technical Report HPL-OSR-93-23, Hewlett-Packard, Palo Alto, CA, Apr. 1993.

[12] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of USENIX Conference on File and Storage Technologies*, pages 231–244, Jan. 2002.

[13] D. Stodolsky, G. Gibson, and M. Holland. Parity logging: Overcoming the small write problem in redundant disk arrays. In *Proceedings of the International Symposium on Computer Architecture*, 1993.

[14] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, Feb. 1996.

[15] Q. Xin, E. L. Miller, T. Schwarz, D. D. E. Long, S. A. Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *Proceedings of the IEEE/NASA Goddard Symposium on Mass Storage Systems and Technologies*, pages 146–156, Apr. 2003.