CSAR-2: a Case Study of Parallel File System Dependability Analysis*

D. Cotroneo, G. Paolillo† and S. Russo
Dipartimento di Informatica e Sistemistica, Università di Napoli "Federico II"
Via Claudio 21, 80125 - Napoli, Italy
{*cotroneo, gepaolil, sterusso*}*@unina.it*

M. Lauria
Dept. of Computer Science and Engineering, The Ohio State University,
2015 Neil Ave., Columbus, OH 43210, USA,
lauria@cse.ohio-state.edu

Abstract

Modern cluster file systems such as PVFS that stripe files across multiple nodes have shown to provide high aggregate I/O bandwidth but are prone to data loss since the failure of a single disk or server affects the whole file system. To address this problem a number of distributed data redundancy schemes have been proposed that represent different trade-offs between performance, storage efficiency and level of fault tolerance. However the actual level of dependability of an enhanced striped file system is determined by more than just the redundancy scheme adopted, depending in general on other factors such as the type of fault detection mechanism, the nature and the speed of the recovery. In this paper we address the question of how to assess the dependability of CSAR, a version of PVFS augmented with a RAID5 distributed redundancy scheme we described in a previous work. First, we address the issues encountered in adding fault detection and recovery mechanisms to CSAR in order to produce CSAR-2. Second, we build a reliability model of the new system with parameters obtained from a CSAR-2 prototype and from the literature. Finally, we assess the system and discuss some interesting observations that can be made with the help of the model. According to our analysis, a representative configuration shows a four nine reliability; the sensitivity analysis shows that a reduction of 15% of the system outage time can be obtained by increasing the speed of the reconstruction by a faster network.

1 Introduction

Parallel scientific applications need a fast I/O subsystem to satisfy their demand of aggregate bandwidth. In particular in clusters environment applications will benefit from a parallel file system (PFS) that can exploit the high-bandwidth and low latency of high performance interconnect such as Myrinet and Gbps Ethernet. PFS such as PVFS [1], can improve significantly the performance of I/O operations in clusters by using striping across different cluster's node. The main objective in the construction of such architecture for data intensive applications continues to be the performance, but the current direction is also toward systems that provide high availability and reliability level. Parallel file systems, and more generally distributed file systems, are complex systems. As the number of entities participating in the system grows, so does the opportunity for failures. One of the major problems with the striping is the reliability because the striping of data across multiple server increases the likelihood of the data loss. Classical RAID approaches are usable locally to the server to provide tolerance

*This work has been partially supported by the Consorzio Interuniversitario Nazionale per l'Informatica (CINI), by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the FIRB Project "Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS)", by the National Partnership for Advanced Computational Infrastructure, by the Ohio Supercomputer Center through grants PAS0036 and PAS0121, and by NSF grant CNS-0403342. M.L. is partially supported by NSF DBI-0317335. Support from Hewlett-Packard is also gratefully acknowledged.

†G. Paolillo performed this work while visiting dr. M. Lauria's group at the Ohio State University.

to the disk failure, but if a server crashes, all the data on that server will be inaccessible until the server is recovered. To solve this problem many redundancy techniques across the servers have been proposed since 1990 [2]. In CEFT-PVFS [4] and RAID-x [25] architectures the RAID1 strategy has been used, in xFS [18], SWARM [7] and Zebra [6] RAID5 has been employed. RAID-like schemes specialized for PFS are being proposed [9] that explore the trade-off among performance, storage efficiency and reliability. There are other schemes extensively used in large distributed storage systems which guarantee more resilience but they have the other drawback to inevitably entail an considerable computational complexity cost both in the read and write operations due to the encode and decode phases respectively [3]. Otherwise, the RAID-like approaches, with the exception of RAID6, are based on parity information that are simply computed in the write operations, while the read operations do not require any decoding phase. Although the disk failures on storage node or more generally storage node failures are the most studied in such systems, simple data redundancy is not sufficient to protect the parallel file system. Due to the concurrency of the client accesses and to the dependency among the storage nodes introduced with the redundant schemes that was not present in the original striped file systems, end failure might damage the system by violating the original semantic of file system. It is worth noting that one of the general guidelines for achieving high I/O performance in parallel application consist in distributing the I/O accesses evenly among several processes. The failure of a process running on a client node might happen during a write operation leaving the system in a inconsistent state (i.e., system error). An error resulting from a client node failure might becomes a system failure (i.e., semantic violation) subsequently to a recovery reconstruction based on corrupted data. So far, to the best of our knowledge, there are no works in the literature that address the semantic issues of parallel file systems dealing with both client and server failures. This paper analyzes dependability issues related to the striped file systems. We adopted the CSAR [9] parallel file system such as a case of study and improve its dependability characteristics in spite of a new type of fault. Quantitative assessment shows the reliability and availability improvements achieved from the enhanced CSAR.

1.1 Parallel and Distributed File Systems

Parallel and distributed file systems can be divided in three main groups:

- Commercial parallel file systems;
- Distributed file systems;
- Research parallel file systems.

The first group comprise: PFS for the Intel Paragon [11]; PIOFS and GPFS for the IBM SP [12]; HFS for the HP Exemplar [13]; XFS for the SGI Origin2000 [14]. These file systems provide the high performance and functionality desired for I/O-intensive applications but are strictly tied to the specific platforms on which the vendor has implemented them. Although they are very effective in providing high level of reliability, they are very expensive to develop and deploy and usually cannot keep pace with the computing industry technology curve. Furthermore they often lack generality allowing only access techniques specifically supported by the proprietary hardware. The second group is characterized by systems designed to provide distributed access to files from multiple client machines, and their consistency semantics and caching behaviour are designed accordingly. The types of workloads resulting from large parallel scientific applications usually do not mesh well with file systems designed for distributed access; particularly, distributed file systems are not designed for high-bandwidth concurrent writes that parallel applications typically require. Some of the best known distributed file systems are NFS [15], AFS/Coda [16], InterMezzo [17], xFS [18], GFS [19]. The last group comprises the research project in the areas of parallel I/O and parallel file systems. PIOUS [8] is one of them and it focuses on viewing I/O from the viewpoint of transactions. PPFS [20] research focuses on adaptive caching and prefetching. In the last few years, among the non-proprietary systems, Galley [21] and PVFS [1] received attention in the context of cluster architectures. Galley looks at disk-access optimization and alternative file organizations. The goal of PVFS is to provide a high-performance file system for the Beowulf class of parallel machines taking advantage of commodity hardware. PVFS is able to deliver very good performance and provides different interfaces for applications, including VFS, MPI-IO and a native one. Though these file systems are freely available, they are mostly research prototype. We have chosen PVFS as a platform to evaluate our approach for fault tolerant parallel file systems because it was designed for performance but in its current form it does not provide any guarantee from the reliability point of view.

Performance Indicators		RAID-10	RAID-5	RAID-x
Parallel Read/Write Time	Large Read	mR/n	mR/n	mR/n
	Small Read	R	R	R
	Large Write	2mW/n	mW/(n-1)	mW/n + mW/n(n-1)
	Small Write	2W	R+W	W
Max. Fault Coverage		n/2 disk failures	Single Disk failure	Single Disk Failure

Figure 1. Characteristics of distributed RAID architectures

1.2 Data consistency and semantic in parallel file systems

The semantics of file system are an important part of the service it provides. The semantics define what can be expected from the different system calls provided for the interface to the file system, in presence of concurrent accesses and failures. The consistency semantics that should be provided by PFSs is still an open issue because the classical strong consistency [10] seems too penalizing in the context of parallel I/O. Most distributed file system provide some form of POSIX semantics which is very restrictive for performance and new relaxed consistency models are starting to be used. Among the non-proprietary parallel file systems, we focused on PVFS [1] which is able to deliver very good performance and provides different interfaces for applications, including VFS, MPI-IO and a native one. PVFS was developed thinking of a parallel file system as a layer of the I/O software stack that is able to match the performance and scalability requirements of the HPC applications. From this point of view, the role of parallel file systems is to: i) manage storage hardware (presenting a single logical view and providing data redundancy); ii) scale to large number of clients (handling concurrent and independent accesses and considering client failures to be a common case); iii) provide a API and semantics (able to preserve it in face of the most common system component failure). Differently from the distributed file systems and most of the PFSs, PVFS does not provide a POSIX-like consistency semantic because it is not clear it can be implemented preserving high performance. In particular, PVFS does not provide guarantee about the atomicity of read and write operation performed concurrently. The responsibility of the consistency is split between the PFS and the I/O middleware running on top of it (i.e., MPI-IO over PVFS). The I/O middleware layer is in charge of the management of concurrent accesses by group of processes while the PFS layer provides only the simple atomic non overlapping write (i.e, it guarantees that all write operations involving non overlapping regions are performed in an atomic way).

1.3 Data Replication for parallel file systems

In order to deal with the system failures in the context of parallel file systems different strategies can be adopted but they all have to take in consideration the added overhead in terms of performance and architectural cost. The objective of guaranteeing an high system availability is achieved through fault tolerance. The increasing number of storage nodes involved exposes the system to failure resulting from a disk or node failure. The idea to extend the well known RAID techniques to the distributed case was explored for the first time in the 1990 by Stonebraker and Schloss [2]. Based on this idea many solutions have been proposed. All the mirroring strategies like RAID-10, chained-declustering RAID [26] and orthogonal striping and mirroring [25] in spite of an low space-efficiency, storage cost equal to 2, provides a good level of reliability since the maximum number of disk/node failures tolerable is n/2. The RAID-5 technique provides a better space-efficiency, variable storage cost of (n+1)/n, it can tolerate only a single disk/node failure. Differently from mirroring, RAID-5 uses a redundant disk block per each stripe as the parity check for that stripe. Another possible approach is to use erasure coding, such as LDPC [28] which has the properties necessary to add arbitrary levels of fault-tolerance by increasing the storage cost. Unfortunately the computational cost of these approaches make impossible their use in parallel file systems since the read and write operations require complex matrix multiplications respectively to decode and encode the data that are prohibitively expensive in parallel file systems. Furthermore, the complexity of those operations, even for the last improved version of erasure codes [29], grows linearly with the size of data to process. For this reason they are extensively used only in wide distributed systems [23] in which the communication times are long enough to justify the encoding and decoding times and the reliability is more important than the performance. In Figure 1.3 is shown the performance of three common distributed RAID architectures suitable for parallel file systems. The data in Figure 1.3 is relative to the parallel I/O of a file of m blocks on RAID architectures with n nodes. The read and write latencies per block are denoted as R and W. For the read, all the RAID architectures theoretically have the same performance even though practical experiments have shown that RAID-5 can exhibit slightly higher performance [25]. Instead, for parallel writes, RAID-10 requires a double number of disk accesses

respect to the simple striping. In RAID-x, the large write is reduced respect to the mirroring to mW/n + mW/n(n-1), for more details see the reference [25]. In RAID-5, the large write takes only mW/(n-1) time to be completed although for the small write there is the well known problem [24] to pre-read the parity and the old data to compute the new parity and execute finally the write. Furthermore, in a distributed striped file system, the lack of a centralized controller introduces a new problem for RAID-5 not present in disk array: each stripe write need to be executed atomically to avoid simultaneous read or update of shared parity blocks. Each one of the above techniques represents a viable solution to tolerate the disk failure that brings the system to a definitive loss of data (system failure). The optimal choice among the above techniques depends on parallel read/write desired performance, the level of required fault tolerance, and the cost-effectiveness in specific I/O processing applications. We concentrate on the RAID-5 technique rather than mirroring because it achieves single fault tolerance with a much lower storage cost by using a fewer redundant disk space. Indeed, even though the storage cost is decreasing rapidly, having the double number of storage nodes also doubles the system exposure to these type of failures. Furthermore there is some previous work [30] that has proposed a solution to alleviate the shortcoming of the RAID-5 write performance.

2 Related Work

PVFS is a RAID-0 style high performance file system providing parallel data access. While it addresses I/O issues for the low-cost Linux clusters by aggregating the bandwidth of the existing disks on cluster nodes, PVFS does not provide, in its current form, any fault tolerance and thus the failure of any single server node renders the entire file system service unavailable. Several approaches have been proposed in the literature to provide some form of tolerance to the disk failure. In [4] is proposed an extension to PVFS from a RAID-0 to a RAID-10 style parallel file system to meet the critical demands on reliability and to minimize the performance degradation due to the management of redundancy. To alleviate the degradation of write performance due to the double data flow common for all the schemes that make use of mirroring, the authors proposed four protocols to optimize the tradeoff between write performance and reliability. Furthermore, they employ an centralized byte-range two-phased locking mechanism to support the multi-reader single-writer semantics with a grant that expires after a short lease period. This centralized mechanism limits the parallelism of I/O operations and adds a context-switching overhead that increases with the number of client nodes. xFS [18], the Berkeley ”serverless” file system makes use of striping for the parallel I/O and RAID-5 to tolerate the disk failure. xFS addressed the problem of small write by buffering writes in memory and then committing them to disk in large, contiguous, fixed-sized groups called log segments. As a result of use of log-structured writes, xFS suffers from the garbage collection overhead. Furthermore, this technique is not suitable in the context of concurrent accesses on the same file from different processes since in the general case a process could be interested only in the write of a partial stripe and thus it could not commit it to disk as a full stripe. Pillai and Lauria address this issue in [30] by implementing in PVFS an hybrid scheme that exploits both the advantage of mirroring on the small write and RAID-5 for the large write. Although all these works address the disk failure problem on parallel I/O architectures, none of them deal with client failure that also represents a potential source of system failure. This paper presents a strategy to achieve tolerance with respect to both disk and client failure by using RAID-5 together with a distributed reservation mechanism to make the system recoverable from those kind of faults.

3 Design of CSAR-2

3.1 Motivations

There are many results in the literature that deal with disk failure for distributed file system but only some of them focus on high performance parallel file systems. The combination of the requirements of high performance and fault tolerance in the context of PFS make the use of different optimized solution possible to accomplish a specific tradeoff. The aim of this study is to preserve the good PVFS performance and its semantic in a cost-effective way while making it robust to the following types of fault: *i)* disk failure; *ii)* storage node failure; *iii)* client failure. As stated in the Section 1.2 the semantic provided by PVFS does not give guarantees on the overlapping I/O operations but assures only that I/O operations that not access the same region will be sequentially consistent. The choice of not supporting POSIX-like semantics was taken to avoid excessive communication needed to coordinate access among all the clients. The necessity to explore new semantic in this context has already been expressed in [25]. One way to deal with disk failure is to replicate the data across different nodes of the architecture by extending the well known RAID techniques to the distributed case. However, assuming that one of

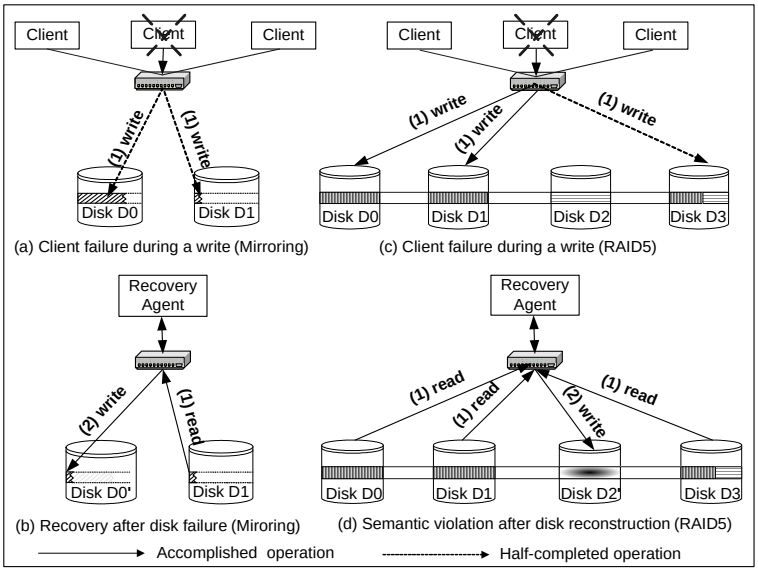

Figure 2. Violation of File System Consistency

redundant techniques is applied to the parallel file system, a client failure could represent a failure for the whole system since the file system semantic could be violated. Figure 2 shows an example of semantic violation due to a client failure in the case of mirroring and RAID5. This violation consist of a not deterministic reconstruction of corrupted file during the recovery procedure due to a client failure occurred in the middle of a write operation. Those situations become possible because of the distributed nature of the file system components and because of the new dependencies introduced by the redundant techniques. The I/O traces of scientific applications show that it is frequent to observe concurrent access from multiple clients on the same files [22]. Actually, due to the large number of components involved the parallel file system should provide data redundancy but also consider client failure to be a common case. To address this issue, the proposed solution aims to tolerate the fault by recovering from them quickly when possible and guaranteeing the file system semantic at every instant.

3.2 Assumptions

One of the basic assumption made by PVFS is the splitting of the responsibility about the semantic between the overlaying middleware (MPI-IO) and PVFS. This assumption allows to speed up the file system I/O operations by avoiding locking mechanisms or complex communications among the clients. By means of the management of the conflicting accesses performed by the MPI-IO, each client will have access to non-overlapping regions of the same file and thus the file system has to care only about the correct execution of non-conflicting accesses as prescribed by its semantic.

3.3 Fault Model

Generally, distributed file systems should be prepared to handle several types of failures: a server's disk can fail, a server can fail, communication links can be broken, and clients can fail. Each type of failure can affect the system differently. Usually server failures are assumed to be fail-stop in that the server does not show any degradation before the failure, and when it fails, it fails by stopping all processing completely. Failure of a machine (server or client) cannot be distinguished from the failure of a communication link, or from slow responses due to extreme overloading. Therefore, when a site does not respond one cannot determine if the site has failed and stopped processing, or if a communication link has failed and the site is still operational. For each one of those faults the file system should handle it in such a way that the consistency and semantic guarantees of the system will not be violated and when possible activate automatic recovery without human intervention. We are primarily concerned with three types of faults: *i)* storage node failure caused by hardware, operating system (OS) or application software faults in the node or by fault on the communication link; *ii)* disk failure on storage node; *iii)* client failure caused by hardware and software but the contribution of hardware errors to the client failure rate is fairly small. Most of the client failures are likely to be due to software cause [32].

3.4 Fault Detection

Preliminary considerations. In RAID-5 the non-overlapping write operations that share one stripe need to be executed in a mutually exclusive way because they could update simultaneously the parity block of the shared stripe. For this reason a mechanism is needed to assure the sequential execution of the write operations that involve regions of the same file on the same stripe. In PVFS different files cannot share the same stripe thus this event could occur only when different clients want to access different portion of the same file which share a stripe. In this case only the first one of the two write operations can be executed and the other one should wait for the completion of the previous one. The mechanism adopted in CSAR [9] makes use of queues on the servers to store the pre-read requests of the parity block that precede every small write (i.e., write of partial stripe) on the same stripe. The pre-read of the parity block is an operation preceding every small write and thus it can be used as a synchronization point: only after the update of the parity block from the current write is completed, the first pre-read request in queue will be served and so on. We modify the CSAR file system so that it can tolerate the client, server, and disk failure. For this purpose we need to put in place a mechanism to detect these failures, trigger the appropriate recovery procedure as soon as possible to remedy the fault, and make the system available again. The client failure represents a potential failure for the file system only during a write operation because it could modify partially the data leaving the system in an inconsistent state. Thus, the client failure detection process need to be performed only during the write operations. The only system component that is informed about each write operation is the storage node involved in the write. For this reason it is natural to think of a client failure detection performed by the storage nodes. Differently, the storage node and the disk failures colud happen at any time and it should be detected as soon as possible in order to recover the node or the disk. Failure of a server's disk differs from failure of a server in that the same server can inform a specific node, on which runs a pvfs client, that we called recovery agent, which performs the appropriate recovery procedure.

Server failure detection. The basic idea is to use a mutual fault detection between clients and servers by exploiting as much as possible the already existent interactions between them. This choice allows to contain the detection overhead during the normal operation of the system preserving the performance but on the other hand could increase the detection time of server node failure. In fact, if the server crashes when there is no client that is accessing the system, the detection will be postponed to the first access. Instead, if the server crashes during an access the involved client will detect suddenly the failure and will inform the recovery agent about the failed server. In the case that the access is a write operation, the client will send to the recovery agent also the information about the stripe under write, but it will also complete the write operation on the remaining servers. In that way the recovery agent will be able to fix the corrupted stripe after the server recovery. It is worth noting that the delay in the detection of node failure does not change the resulting file system consistency in that no operations will be performed in the while. Furthermore, assuming that those kind of systems are used for most of their life time, we could state that there is always some process in progress which is accessing the system and thus the delay in the detection could be not considered at all.

Client failure detection. We detect the client failure only when it is necessary, that is during a write operation. To enable the server nodes to detect the client failure a timeout mechanisms is used in the write phases. The only timeout mechanism is not sufficient in that each server can check out the only correct execution of the write operation in progress on itself. In fact, a single client write involves at the same time more writes on different servers and the correct completion of a part of them represents an undetectable system error from the single server point of view. For instance, when a client starts a write operation that involves three servers but it crashes just after the completion of the writings on two of them but before it was initiating the writing on third one, each server node will be unable to detect the client failure because each one has completed its operation correctly or does not perform it at all. For this reason we modified the protocol concerning the write operation adding a reservation phase before each write. When a client wants to perform a write, it sends a short reservation message with the information about the data to write (i.e., file, region in the file and write size) to each server involved in the operation and starts a timer. Each server replies to the client by sending an acknowledgment message and starting a local timer. Only if the client receives all the acknowledgments from the servers by the timeout, it concludes that no server is failed before the write starts and it stops the timer and starts the write. During the write phase each client message exchanged with the servers is acknowledged and thus eventually server failures can be detected by timeout. The server can also detect client failure during the write phase in that the server knows the amount of data that it should receive before the timeout expires. With this new phase each server knows that a write operation is in progress and only if every servers receive the complete data from the client by a timeout then the write operation can be considered performed properly. Each server, on which the timeout expires, will notify the client failure to the recovery agent. The reservation phase performed before each write operation should take

approximately the time of a round trip message but in the case of small write it can be piggybacked along with the request of read necessary to update properly the parity block in RAID-5. Furthermore, the reservation phase could solve also the problem of guaranteeing the atomicity of the write operations on the shared stripe by blocking successive clients up to the completion of the one in progress. Only the first client that receive all the reply messages by the servers can proceed with the write while the successive clients will have to wait for the completion of the first one in that they will receive a negative reply from the server already busy on the same stripe.

3.5 Recovery Procedures

For each different type of system fault, there is an appropriate recovery procedure that should be activated to recover the system. In the next subsections we describe the three recovery procedures that are thought to lead the system in a consistent state and then recover the system automatically or wait for the human intervention to eventually replace the faulty component.

Recovery from Client Failure. When a client failure is detected by one of the server involved in the write, it informs the recovery agent about the region affected by the failed write. The recovery agent undertakes the following steps: *i)* read the data written up to that moment from the failed client; *ii)* compute the parity blocks of the involved stripes; *iii)* write the new parity blocks. This recovery procedure can be performed during the normal system working and thus the system availability does not undergo modifications.

Recovery from Server Failure. Just after the node failure is detected, all the remaining servers are informed that the recovery procedure is in progress and so they will reject all the future requests and complete the ones in progress, already acknowledged in the reservation phase, in order to reach as soon as possible a consistent state in which to block the system. During the time interval between the server failure and the notification of failure to all the other servers no new write operation will be accepted because the faulty server will not reply to any request. After the server is recovered all his stripe units involved in write operations that were in progress at the failure time will be updated according the information of the other servers. Different failures could affect the server, for software failures (e.g., operating system, application software) it is sufficient the node reboot, while for hardware failures it is necessary the substitution of faulty component. To distinguish between the two faults usually it is performed node reboot and only if the problem persists the failure is considered hardware. In both cases, when at least one node does not work properly the system become unavailable for the new requests and it stays in this state until to the end of recovery procedure. The procedure leads the system exactly in the same state in which it should be after the completion of the last request accepted before the failure.

Recovery from Disk Failure. The disk failure recovery procedure differs from the hardware server failure only because after the substitution of the failed disk, requiring a human intervention, it proceeds to the reconstruction of the data on the new disk. The reconstruction operation will take a time proportional to the amount of data that was present at the failure instant. For each stripe the reconstruction consists of parallel block read operations on the *n-1* servers, computation of the XOR function on these data and finally the write of the resulting block on the new disk. It is worth noting that these operations will be performed with the system completely unloaded and thus the time to recover can be estimated deterministically once the system architecture is chosen. As for the time to compute the XOR function, it increases proportionally to the number of servers but it represents a small contribution to the overall reconstruction time because it is a simple binary operation.

4 Dependability Evaluations

In this section, we show the dependability issues related to parallel file system. Quantitative measures have been performed to show the reliability and availability enhancement provided by CSAR-2.

4.1 Experimental Setup

The parameters used to perform the dependability assessment of CSAR-2 have been measured using the prototype implementation deployed on the Ohio Supercomputer Center Itanium 2 cluster. The cluster is composed of compute nodes with four gigabytes of RAM, two 900 MHz Intel Itanium 2 processors, 80 Gigabytes ultra-wide SCSI hard drive and one gigabit Ethernet interface and one Myrinet interface. In particular, the reconstruction speed has been measured twice, using each one of the two networks at a time.

Symbol	Meaning	Value
N	Number of storage nodes	variable
M	Number of clients	variable
$1/\lambda_{disk}$	Mean time to disk failure	500,000 hours
λ_{client}	Client failure rate	0.00094 per hour
$\lambda_{client/write}$	Client failure rate during write operation	0.06278 per year
λ_{Snode}	Software server node failure rate	0.00094 per hour
λ_{Hnode}	Hardware server node failure rate	1 per year
$1/\mu_{client}$	Mean time to client repair	0.5 seconds
$1/\mu_{disk}$	Mean time to disk repair	8 hours
μ_{reboot}	Software server repair rate	20 per hour
$\mu_{replace}$	Hardware server repair rate	6 per day
α	Fraction of total execution time spent performing I/O operations	0.3177
β	Fraction of whole I/O time spent for write operations	0.024

Table 1. Symbols, notations and values of dependability models

4.2 Dependability Model

In this paper we make the following assumptions in the dependability model:

- All component failure events are mutually independent;
- The failure and repair time are exponentially distributed;
- The metadata server is assumed more dependable than the other nodes in the cluster and for this reason we do not consider the failure of this component in our model.
- The faults of network host interface are comprised in the host fault.

4.3 Reliability

In PVFS in the current form, with no data redundancy mechanisms, the single disk failure brings the system to the failure because it involves the loss of data. Instead, the client failure does not represent a failure because PVFS does not have data redundancy to keep consistent. Therefore, for PVFS the reliability is:

$$R_{PVFS}(t) = e^{-(N\lambda_{disk})t} \tag{1}$$

where λ_{disk} is the disk failure rate and *N* is the number of server nodes. As for PVFS with distributed RAID-5, two events could lead the system to the failure. The first event is the second disk failure that entails the data loss and the second event is the single disk failure after a client failure during write operation. The client failure during the write represents an error for the system in that the parity information of the involved stripes are not consistent. This error could become a system failure for the system only after a recovery procedure from disk failure. This second event decreases the theoretical reliability of RAID-5 scheme which is expressed in terms of MTTDL (Mean Time To Data Loss) as follow:

$$MTTDL = \frac{(2N-1)\lambda_{disk} + \mu_{disk}}{N(N-1)\lambda_{disk}^2} \tag{2}$$

where μ_{disk} is the disk repair rate. To evaluate the reliability loss due to client failure event we conducted some simple

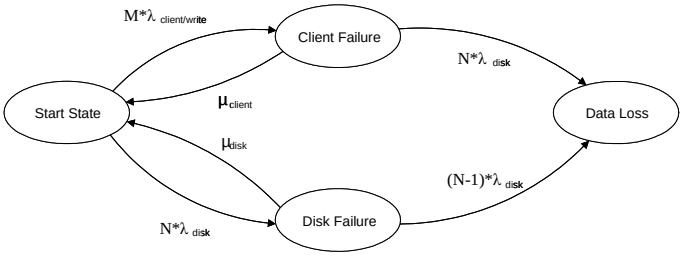

Figure 3. Reliability model for client failure treatment

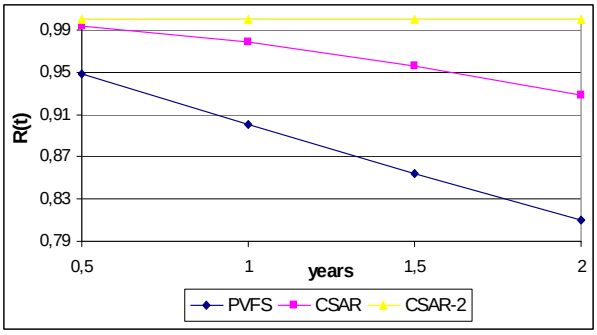

Figure 4. Reliability comparison for 6 server nodes and 8 clients

numerical examples by using the Markov model in Figure 3. The notations, symbols and values in the dependability models are listed as in Table 1. The client failure rate during a write operation, $\lambda_{client/write}$, is predicted as follow:

$$\lambda_{client/write} = \lambda_{client} * \alpha * \beta \tag{3}$$

where λ_{client} is the failure rate of a processing node in a cluster environment, α is the fraction of total execution time spent performing I/O operations and β is the fraction of time spent for write operations with respect to the whole I/O time. It is worth noting that it is hard to extract generic values that can represent scientific application workload. The time spent on I/O operations depends on many factors such as for instance the type of application (CPU intensive vs. I/O intensive) or the level of parallel I/O provided by the architecture. Figure 4 shows the reliability values related to the three different schemes spread over a period of 2 years: *i)* PVFS in the current implementation; *ii)* PVFS with distributed RAID-5 without client failure treatment; *iii)* PVFS with distributed RAID-5 and client failure treatment. The reliability values concerning the schemes with client failure treatment have been obtained solving the model in Figure 3 by using the SHARPE package [31]. As for the scheme without client failure treatment, it has been solved by means the same model without the transition from *Client Failure* to *Start State* that represents the recovery action. The system configuration is composed by 6 server nodes and 8 clients. Each server disks has 80 GB and a mean time to failure (MTTF) of 10^5 hours. The disk repair rate μ_{disk} has been obtained adding the time to replace the failed disk, estimated around 4 hours according to the mean time to repair for hardware in [34], and the time to reconstruct the entire disk on the basis of the information on the remaining disks at a reconstruction speed of 5 MB/second. The time to recover from the client failure $1/\mu_{client}$, instead, is composed by two terms, the time to detect the fault and the time to fix the parity. The detection time is the sum of the timeout that the server nodes use to determine the client failure plus the time to inform the recovery agent about the client failure. The time to fix the parity depends on the number of stripes involved in the failed write. All these times are measured on a prototype in which this scheme is implemented. In particular, the λ_{client} value has been chosen according the prediction of software failure rate for processing node in [32], while α and β have been extracted as mean values from the work [33] focused on the workload characterization of I/O intensive parallel applications. The numerical example, summarized in Figure 4, shows the remarkable improvements that can be obtained in terms of reliability by using the client failure treatment respect to the RAID-5 only technique. These improvements become more pronounced when the number of clients and server nodes increase in that the rate of the two transitions on the upper branch of the Markov model in Figure 3 increases correspondingly. The comparison with PVFS is simply resumed by the fact that, with a configuration of 6 servers, after two years it provides a probability of

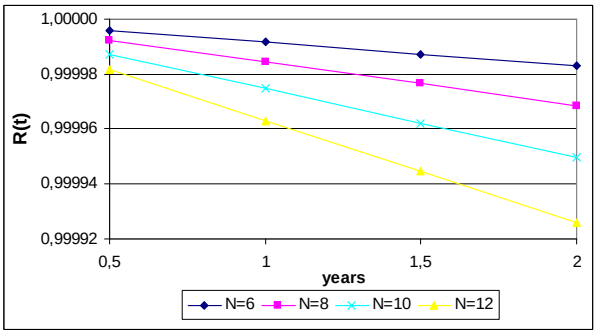

Figure 5. Reliability for different system configurations

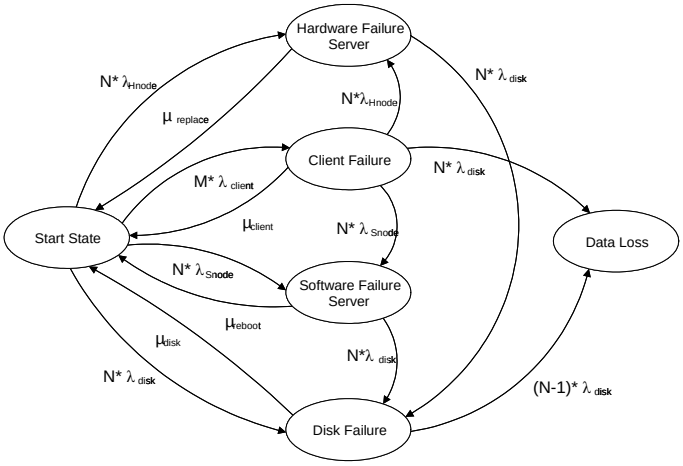

Figure 6. Markov model for the instantaneous availability

data loss equal to 0.19 respect to 0.000017 of the proposed strategy. In Figure 5, have been resumed the reliability values for PVFS with RAID-5 scheme and client failure treatment for different system configurations. In particular, it is possible to appreciate the dependence between the reliability and the number of server nodes. This dependence is due to the fact that a greater number of disks in the system increases the probability to incur in the second disk failure during the reconstruction of the previous failed one.

4.4 Availability

To evaluate the system availability for the improved PVFS we use the Markov model in Figure 6 that represents an extension of the one in Figure 3 in that there are also the states in which the system is unavailable during the recovery procedures. Also this Markov model, like the one used for the reliability study, presents an absorbing state that corresponds to the system failure due to the data loss. For that reason we cannot study the steady state availability but only the instantaneous availability of the system that represents the probability that the system is properly functioning at time *t*. The system become unavailable during the recovery procedures from server and client failures. In the distributed RAID-5 scheme, in fact, the absence of one server, even though does not force necessarily the system to go down, exposes the system to the second disk failure that means system failure. A way to reduce the system vulnerability is to recover it as soon as possible in spite of the system availability. We decide to stop the service just after the server failure detection to speedup the recovery action and preserve the system consistency. The server failure have been divided in hardware and software on the basis of the different recovery actions. Furthermore, another distinction has been done for the disk failure that represents a particular case of hardware failure in that it needs the component replacement but also the reconstruction of the data. We assume that the software failure should be recovered with a system reboot, while the hardware failures need the human intervention for

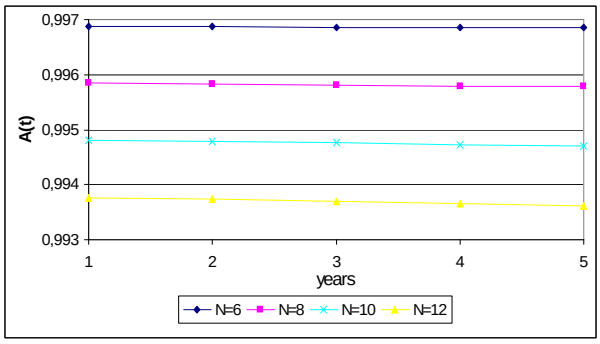

Figure 7. Availability for different system configurations

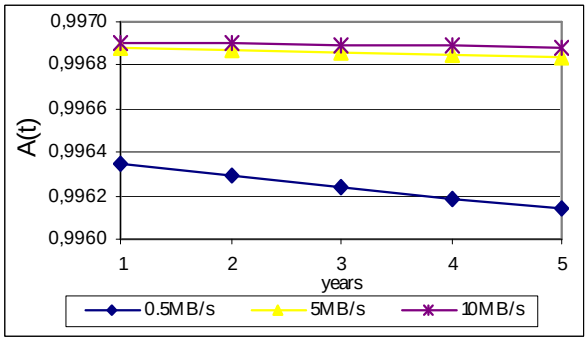

Figure 8. Sensitivity analysis of Availability

replacement. This characterization and the relative values used in the Markov model of Figure 6, has been taken from the work [34]. As for the client failure, it entails a temporary unavailability of the file portion involved in the failed write operation during the recovery procedure although all the other clients can access the remaining files. Furthermore, the client failure influences the availability also bacause it expose the system to the failure under single disk failure for the short time necessary for his recovery. In Figure 7, instead, is shown the dependence of system availability from the number of server nodes over a period of three years. This dependence can be summarized by an availability decrease of 0.001 for each increment of 2 nodes for the first year that will increase over the time. The availability provided by PVFS can be estimated through his reliability in that in the absence of a repair, availability *A(t)* is simple equal to the reliability *R(t)*. If we consider as system fault the only disk failure ignoring the server failure we can use the reliability values, related to 6 server nodes on Figure 4, as an upper bound of his availability and compare it with the availability of the enhanced PVFS for the same configuration on Figure 7. So, with the proposed strategy the availability is always greater than 0.99685 over a period of five years while PVFS just after six months provides an availability under 0.94880.

4.5 Sensitivity Analysis

The time to recover from the disk failure is composed by the time to replace the disk and the time to reconstruct the whole data. It represents a crucial parameter for the dependability characteristic of the system. In fact, the shorter this interval of vulnerability, the lower is the probability of a second concurrent disk failure. The reconstruction time is strongly dependent on the particular system architecture (i.e., disk access time, network bandwidth) and on the possible optimizations in the reconstruction procedure. We present a sensitivity analysis in which we vary the reconstruction speed and observe the relative changes in the corresponding dependability. The values of reconstruction speed have been chosen on the basis of the measurements performed on the prototype of CSAR-2 for a system configuration of 6 servers and 8 clients. In particular, 0.5, and 5 MB/sec are the reconstruction speed relative to the Gigabit Ethernet and Myrinet network, respectively, while 10 MB/sec is the speed of an hypothetical system with higher disk and network performance. Figure 8 shows that the instantaneous availability of CSAR-2 relatively to 5 and 10 MB/sec are quite similar and both are substantially better than the

one relative to 0.5 MB/sec. The reason is that when the reconstruction is very fast, the mean time for a human intervention, fixed to 4 hours in our experiments, become the predominant factor. Only with the reconstruction speed of 0.5 MB/sec, the bottleneck is represented by the network bandwidth and most of the recovery time is spent in the data reconstruction. This observation suggests that, in order to improve the availability, the network speed is as important as the time to replace the hardware of the cluster. Furthermore, the amount of availability improvement achieved over one year by increasing the reconstruction speed from 0.5 to 5 MB/sec is equal to 0.00053 (from 0,99634 to 0,99688) which entails a reduction of the combined system outages in a year of 15 percent. The reliability graph is not reported because it is qualitatively similar to the availability one.

5 Conclusions and Future Work

This paper analyzes the dependability issues related to the striped file systems. We enhance CSAR, a PVFS version augmented with a distributed RAID5 scheme, by adding fault detection and recovery mechanisms and evaluate the reliability and availability improvements by means of dependability models using the parameter of CSAR-2 prototype. The dependability analysis of the system shows a 4 nine reliability while the sensitivity analysis based on the data reconstruction speed shows a reduction of the combined system outage time up to 15 percent. Future work will aim to: *i)* extend the dependability results with more system configurations; *ii)* assess the performance cost of CSAR-2 by means of standard benchmark.

6 Acknowledgments

We wish to thank Manoj Pillai at Ohio State University for the insightful discussions we had with him and for his assistance with the CSAR code.

References

[1] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: a parallel file system for Linux clusters. In Proc. of the 4th Annual Linux Showcase and Conference, Pages: 317-327, Atlanta, GA, 2000. Best Paper Award.

[2] M. Stonebraker, G. A. Schloss, Distributed RAID-a new multiple copy algorithm, Proceedings of Sixth Int. Conf. on Data Engineering, 5-9 Feb. 1990, Pages: 430-437.

[3] J. S. Plank, M. G. Thomason, A practical analysis of low-density parity-check erasure codes for wide-area storage applications, Dependable Systems and Networks, 2004 International Conference on , 28 June-1 July 2004

[4] Z. Yifeng, J. Hong, Q. Xiao, D. Feng, D. R. Swanson, Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS). Proceedings of 3rd IEEE/ACM Int. Symposium on Cluster Computing and the Grid, 12-15 May 2003, Pages:730 - 735

[5] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Young. Serverless network file systems. ACM Transactions on Computer Systems, Feb. 1996.

[6] J. Hartman and J. Ousterhout. The Zebra striped network file system. ACM Transactions on Computer Systems, Aug. 1995.

[7] J. H. Hartman, I. Murdock, and T. Spalink. The swarm scalable storage system. Proceedings of the 19th International Conference on Distributed Computing Systems, May 1999.

[8] S. A. Moyer and V. S. Sunderam, PIOUS: A scalable parallel I/O system for distributed computing environments, in Proceedings of the Scalable High-Performance Computing Conference, 1994, pp. 71-78.

[9] M. Pillai, M. Lauria, CSAR: Cluster Storage with Adaptive Redundancy, ICPP-03, pp. 223-230, October 2003, Kaohsiung, Taiwan, ROC.

[10] P. Triantafillou and C. Neilson, Achieving strong consistency in a distributed file system, IEEE Trans. on Software Engineering, Vol. 23, Issue: 1, Jan. 1997, pp.35-55

[11] Intel Scalable Systems Division. Paragon system user's guide. Order Number 312489-004, May 1995.

[12] P. F. Corbett, D. G. Feitelson, J. P. Prost, Ge. S. Almasi, S. J. Baylor, A. S. Bolmarcich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. Herr, J. Kavaky, T. R. Morgan, and A. Zlotek. Parallel file systems for the IBM SP computers. IBM Systems Journal, 34(2) : 222 - 248, January 1995.

[13] Rajesh Bordawekar, Steven Landherr, Don Capps, and Mark Davis. Experimental evaluation of the Hewlett-Packard Exemplar file system. ACM SIGMETRICS Performance Evaluation Review, 25(3):21 - 28, December 1997.

[14] XFS: A next generation journalled 64-bit filesystem with guaranteed rate I/O. http://www.sgi.com/Technology/xfswhitepaper.html.

[15] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification: RFC 1831. Internet Engineering Task Force, Network Working Group, June 1995.

[16] Peter J. Braam. The Coda distributed file system. Linux Journal, No.50, June 1998.

[17] Peter J. Braam, Michael Callahan, and Phil Schwan. The InterMezzo filesystem. In Proceedings of the O'Reilly Perl Conference 3, August 1999.

[18] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, pages 109 - 126. ACM Press, December 1995.

[19] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O'Keefe. A 64-bit, shared disk file system for Linux. In Proceedings of the Seventh NASA Goddard Conference on Mass Storage Systems. IEEE Computer Society Press, March 1999.

[20] J. Huber, C. L. Elford, D. A. Reed, A. A. Chien, and D. S. Blumenthal. PPFS: A high performance portable parallel file system. In Proceedings of the 9th ACM International Conference on Supercomputing, pages 385 - 394, Barcelona, July 1995. ACM Press.

[21] N. Nieuwejaar and D. Kotz. The Galley parallel file system. Parallel Computing, 23(4):447 - 476, June 1997.

[22] D. Kotz and N. Nieuwejaar. Dynamic file-access characteristics of a production parallel scientific workload, in Proceedings of Supercomputing '94,Washington, DC, 1994, pp. 640-649, IEEE Computer Society Press.

[23] J. Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In Proceedings of ACM ASPLOS. Nov. 2000.

[24] 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 (CSUR), Vol. 26, no. 2, pp. 145-185, 1994.

[25] K. Hwang, H. Jin, and R. S. C. Ho, Orthogonal Striping and Mirroring in Distributed RAID for I/O-Centric Cluster Computing, IEEE Trans. on Parallel and Distributed Systems, Vol. 13, no. 1, Jan. 2002.

[26] H. Hsiao and D. DeWitt, Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines, 6th Int. Data Eng. Conf., pp.456-465, 1990.

[27] S. A. Vanstone, and P. C. Van Oorschot, An Introduction to Error Correcting Codes with Applications. Klewer Academic Publishers, 1989.

[28] R. G. Gallager, Low-Density Parity-Check Codes. PhD thesis, Massachusetts Institute of Technology, 1963.

[29] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, Improved low-density parity-check codes using irregular graphs. IEEE Trans. on Information Theory 47, 2 (Feb. 2001), 585-598.

[30] M. Pillai, and M. Lauria, A High Performance Scheme for Cluster File Systems, Proc. of the IEEE Int. Conf. on Cluster Computing. 2003.

[31] K. S. Trivedi, SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator, Proceedings of Int. Conf. on Dependable Systems and Networks. 23-26 June 2002 Page(s):544.

[32] V. B. Mendiratta, Reliability analysis of clustered computing systems, Proceedings of The 9th Int. Symp. on Software Reliability Engineering, Nov. 1998 pp. 268-272.

[33] E. Smirni, D. A. Reed, Workload Characterization of Input/Output Intensive Parallel Applications, Proceedings of the 9th Int. Conf. on Computer Performance Evaluation: Modelling Techniques and Tools, 1997. Vol.1245, Pages: 169-180

[34] H. Sun , J. J. Han, H. Levendel, A generic availability model for clustered computing systems. Proceedings of Pacific Rim Int. Symposium on Dependable Computing. 17-19 Dec. 2001 Page(s):241-248.