Available online at www.sciencedirect.com

SCIENCE DIRECT®

J. Parallel Distrib. Comput. 66 (2006) 291–306

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

CEFT: A cost-effective, fault-tolerant parallel virtual file system

Yifeng Zhu[a,*], Hong Jiang[b]

[a]Department of Electrical and Computer Engineering, University of Maine, Orono, ME 04469-5708, USA
[b]Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE 68588-0115, USA

Received 12 June 2004; received in revised form 30 September 2005; accepted 28 October 2005
Available online 19 December 2005

Abstract

The vulnerability of computer nodes due to component failures is a critical issue for cluster-based file systems. This paper studies the development and deployment of mirroring in cluster-based parallel virtual file systems to provide fault tolerance and analyzes the tradeoffs between the performance and the reliability in the mirroring scheme. It presents the design and implementation of CEFT, a scalable RAID-10 style file system based on PVFS, and proposes four novel mirroring protocols depending on whether the mirroring operations are server-driven or client-driven, whether they are asynchronous or synchronous. The comparisons of their write performances, measured in a real cluster, and their reliability and availability, obtained through analytical modeling, show that these protocols strike different tradeoffs between the reliability and performance. Protocols with higher peak write performance are less reliable than those with lower peak write performance, and vice versa. A hybrid protocol is proposed to optimize this tradeoff.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Cluster computing; Parallel I/O; Reliability analysis; Markov process; CEFT; PVFS; Linux clusters

1. Introduction

Rapid advances in general-purpose communication networks have motivated the deployment of inexpensive commodity components to build competitive cluster-based storage solutions to meet the increasing demand of scalable computing. In the recent years, the bandwidth of these networks has been increased by two orders of magnitude [6,18,29], which has greatly narrowed the performance gap between them and the dedicated networks used in commercial storage systems, such as the fiber channels. The significant improvement in network bandwidth offers an appealing opportunity to provide cost-effective high-performance storage services by aggregating the capacity and bandwidth of all commodity disks that already exist as an integral part of each node in a typical cluster.

Parallel storage systems in a cluster aim to alleviate the I/O bottleneck for data-intensive scientific applications by providing efficient parallel access to the storage devices distributed across the entire cluster. One major concern in designing such systems is the fault-tolerance (or lack thereof). Assume that the Mean Time To Failure (MTTF) of a disk is three years and all the other hardware and software components of a cluster, such as network, memory, processors and operating systems, are fault-free, the MTTF in such a cluster-based storage system with 128 server nodes will be reduced to around nine days if the failures of storage nodes are independent of one another (3 years–128 $\approx$ 9 days). Needless to say, the MTTF will be further significantly reduced when the failures of the other components are considered. Similar to disk arrays [43], without fault tolerance, these storage systems built upon clusters are too unreliable to be useful.

To accommodate the fact that clusters tend to be error-prone since the reliability of a cluster is inversely proportional to the number of nodes that it has, this paper studies the incorporation of mirroring protocols into parallel storage systems in a cluster to improve the reliability and availability of cluster-based storage systems. More specifically, we present our design, implementation and performance evaluation of a RAID-10 style, cost-effective and fault-tolerant (CEFT) parallel virtual file system [63] in a cluster-environment. We have chosen PVFS [11] as a platform for our research that allows us to test our proposed protocols in a real file system. PVFS is a freely

* Corresponding author.
E-mail addresses: zhu@eece.maine.edu (Y. Zhu), jiang@cse.unl.edu (H. Jiang).

available parallel file system for Linux clusters that delivers scalable, high-bandwidth storage services to applications running in clusters. Although our current implementation is based on PVFS, our protocol designs can provide system designers with significant insights into the fault-tolerance design for general cluster-based storage systems.

The primary contributions of this work are three-fold. Firstly, this work proves the feasibility of providing high performance of storage services in a computational cluster without adding any additional hardware. Secondly, it develops an analytical model, based on Markov process, to evaluate the reliability of the proposed mirroring protocols, which can also be easily adopted to analyzing the reliability of mirroring schemes in any other non-centralized system. This model distinguishes itself from the conventional Markov models for a centralized RAID-10 system by capturing an important nature of non-centralized systems, that of loose coupling. In a cluster environment, the data duplication from one node to another node has to experience the queuing delay and network latency, whereas the duplication from one disk to another disk in a conventional centralized RAID-10 is almost instantaneous since these disks are closely coupled directly through fast data buses. Thirdly, it designs four different mirroring protocols that strike different tradeoffs between the reliability and performances.

The rest of this paper is organized as follows. In the next section, we discuss the related work. Then the design and implementation of our CEFT are presented in detail in Section 3. Section 4 describes four different mirroring protocols and Section 5 evaluates the performance of CEFT, with a focus on the write performance of these protocols under a microbenchmark, along with a summary discussion of read performance of CEFT based on a similar microbenchmark and a real application case study. In Section 6, a Markov-chain model is constructed to accurately analyze the reliability and availability of these protocols. Finally, Section 7 presents our conclusions and describes possible future work.

2. Related work

The proposed system has roots in a number of distributed and parallel file systems. The section presents a brief overview of this related work.

Swift [9], Zebra [25] and xFS [1] employ RAID-4/5 to improve redundancy. Swift conducts file stripping so that large files benefit from access parallelism. Zebra aggregates client's data first and then does striping on log-structured file systems to enhance small write performance. xFS removes the centralized file manager in Zebra and dynamically distributes the metadata management among multiple server nodes for the sake of performance and scalability. In these designs, the parity is calculated by client nodes. In I/O-intensive applications, the calculation of parity potentially wastes important computational resources on the client nodes, which are also computation nodes in a cluster environment. In addition, both systems can tolerate the failure of any single node. The failure of a second node causes them to cease functioning.

PIOUS [36] employs a technique of data declustering to exploit the combined file I/O and buffer cache capacities of networked computing resources. It provides minor fault tolerance with a transaction-based approach so that writes can be guaranteed to either completely succeed or completely fail.

Petal [33], a block level distributed storage system, provides fault tolerance by using *chained declustering* [27]. Chained declustering is a mechanism that reduces the reliability of RAID-1 to trade for balancing the workload on the remaining working nodes after the failure of one storage node [19]. In Petal, the failure of either neighboring node of a failed node will result in data loss, while only the failure of its mirrored node can make the data unavailable in RAID-1. In addition, Petal does not provide a file level interface and the maximum bandwidth achieved is 43.1 MB/s with 4 servers and 42 SCSI disks, which does not fully utilize the disk bandwidth.

RAIDx [28], a block level storage system designed for clusters, proposes a novel scheme called orthogonal striping and mirroring that degrades the reliability of RAID-10 to improve the write performance. In this scheme, the data blocks of one stripe and their redundancy blocks in the form of mirroring are placed orthogonally such that the former take residence on different disks while the latter are stored sequentially in a single disk. While RAIDx can tolerate only one disk failure, it significantly improves the write performance by reducing the number of write operations and exploiting the sequentiality exhibited in the redundant blocks. One major concern is that the fault tolerance provided in RAIDx is relatively weak for a cluster with hundreds of disks.

GPFS [52] is IBM's parallel shared-disk file system for clusters. The stripping among many disks that are connected over a switching fabric, a dedicated storage network, to the cluster nodes achieves high I/O performance. It utilizes dual-attached RAID controllers and file level duplication to tolerate disk failures. While CEFT requires no additional hardware in a cluster, GPFS typically needs dedicated switching fabric and RAID controllers.

Google file system (GFS) [20] is a scalable distributed file system that supports the heavy workload at the Google website and runs on a cluster with inexpensive commodity hardware. In GFS, a single master node is used to maintain the metadata and the traffic of high volume of actual file contents are diverted to bypass the master to achieve high performance and scalability. GFS takes an aggressive approach to provide fault tolerance, in which three copies of data are stored by default. GFS is tailored to meet the particular demands for Google's data processing and is not a general-purpose file system.

PVFS [11,31] is an open source RAID-0 style parallel file system for clusters. It partitions a file into stripe units and distributes these stripes to disks in a round robin fashion. PVFS consists of one metadata server and several data servers. All data traffic of file content flows between clients and data server nodes in parallel without going through the metadata server. The fatal disadvantage of PVFS is that it does not provide any fault-tolerance in its current form. The failure of any single server node will render the whole file system dysfunctional. Ref. [44] proposes a hybrid fault tolerance scheme based on

PVFS, which chooses to use RAID-5 style redundancy for large writes and RAID-1 style redundancy for small writes. Maintaining an optimal threshold to distinguish large writes from small writes for a diversity of workloads is a challenging issue that remains to be addressed.

The proposed CEFT parallel virtual file system is a RAID-10 style parallel file system, which first stripes the data across a group of storage nodes and then mirrors these data onto another group. In Ref. [63], we introduce four different mirroring protocols that strike different tradeoff between performance and reliability. In Ref. [64], we optimize the performance of write operations by exploiting the disparity of resource utilization between each mirroring pair. In Ref. [65], we utilize the redundancy in CEFT to improve the read performance by 100% by doubling the degree of the parallelism: reading the first half of a file from one storage group and the second half from the other group in parallel. In Ref. [68], we run a real data-intensive scientific application on CEFT and further prove that the read and write performance optimization techniques described above are highly efficient.

3. Implementation overview

3.1. The choice of fault tolerance designs

There are several approaches to providing fault tolerance in parallel file systems. One simple way is to strip data on multiple RAIDs that are attached to different cluster nodes. However, this approach provides moderate reliability since it cannot tolerate the crash of any cluster nodes.

Another possible approach to providing fault tolerance is to use parity-based redundancy. RAID 5 is a typical example that can tolerate one-node failures and some other parity schemes, such as EVENODD [5], RM2 [41] and RDP [15], can be deployed to tolerate two-node failures. However, these parity-based redundancies cannot satisfy the reliability requirement in a large cluster. In these schemes, a second or third node failure results in the temporary or permanent inaccessibility of all the data and the probability of such failures are not negligible in a cluster with hundreds or even thousands of nodes. In addition, small writes cause their performance to degrade. For example, a small RAID-5 write involves four I/Os, two to pre-read the old data and old parity and two to write the new data and old parity [12]. In a loosely coupled system, such as clusters, the four I/Os can cause significant delays. Finally, in a distributed system, the parity calculation should not be performed by any single node to avoid severe performance bottleneck; instead, it should be performed distributively. However, this distributed nature complicates the concurrency control since multiple nodes may need to read or update the shared parity blocks simultaneously.

Still another possible approach is to use erasure coding, such as Rabin's Information Dispersal Algorithm (IDA) [48,3] and Reed Soloman Coding [34,45], to disperse a file into a set of pieces such that any sufficient subset allows reconstruction. Consequently, this approach is usually more space-efficient and reliable than RAID-5 and mirroring. While the erasure coding has been extensively used in P2P systems [49], it may not be suitable for GB/s scale cluster file systems since the dispersal and reconstruction require matrix multiplications and multiple disk accesses and generate a potentially significant computational and I/O overhead.

Hybrid algorithms are also an appealing approach. For example, AutoRAID [62] is a hybrid algorithm implemented in a single RAID controller that combine RAID 1 and RAID 5 into a two-level storage hierarchy, in which the upper level mirroring is employed for active data to achieve better performance and the lower level RAID-5 parity is used for inactive data and read-only data to lower storage cost. Data are adaptively migrated between these two layers in the background to balance the workload. CSAR [44] also combines RAID 0 and RAID 5 in a cluster-based storage, but it uses RAID 1 for small writes and RAID 5 for large writes. While these hybrid algorithms can potentially inherit the advantages of different fault tolerance schemes, a major challenge for storage designers is how to optimize the performance for a diversity set of the application workload. This challenge becomes more significant in a scientific computational environment since there is no clear consensus in characterizing the I/O requirements and workload patterns of scalable scientific applications [55,60].

In CEFT, we choose to use a simple yet effective scheme that mirrors striped data among different nodes to improve the reliability while maintaining a high aggregated throughput. As the storage capacity doubles every year [26], the storage cost decreases rapidly. By August 2003, the average price of commodity IDE disks has dropped below 0.5 US$/GB. Therefore, it makes perfect sense to "trade" 50% storage space for performance and reliability. Compared with the parity and erasure coding style parallel systems, our approach adds the smallest operational overhead and its recovery process and concurrency control are much simpler. Compared with the hybrid ones, our approach has significantly less complexity and does not suffer the performance degradation for a diversity set of scientific computation workload. Another benefit from mirroring, which the other redundancy approaches cannot achieve, is that the aggregate read performance can be doubled by doubling the degree of parallelism, that is, reading data from two mirroring groups simultaneously [65].

3.2. Design of CEFT

CEFT is a RAID-10 style parallel file system that mirrors the striped data between two groups of server nodes, one primary group and one backup group, as shown in Fig. 1. There is one metadata server in each group. To simplify the synchronization process, clients' requests go to the primary metadata server first. If the primary metadata server fails, all metadata requests will be redirected to the backup one. All following requests will directly go to the backup metadata server until the primary one is recovered and rejoins the system. For write requests, the data will first be written to the primary group and then duplicated to the backup group. Four duplication (or mirroring) protocols are designed and will be discussed in Section 4.

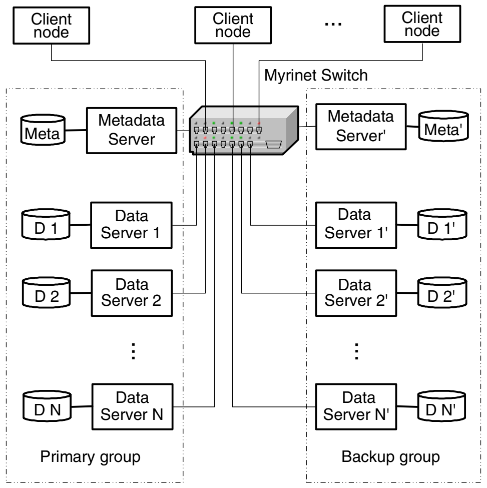


Fig. 1. Basic diagram of CEFT.

striping information	
strip_width	3
block_size	65536 bytes
location	1, 2, 7
data duplication status	
dstatus	3, 3, 3
others	
uid	213
gid	501
mode	rwx–r–r
size	786432 bytes
atime	Wed Jan 22 09:00:10 2003
⋮	⋮

Fig. 2. Sample metadata in CEFT.

3.3. Metadata management

CEFT maintains two metadata structures, system metadata and file metadata. The system metadata indicates the dead or live status of the data servers. When one data server is down, all I/O accesses will be redirected to its mirror server. Currently, a data server is simply thought to be down if the metadata server does not receive the periodic “heartbeat” message from this data server within a certain amount of time. The file metadata describes the striping information, the data mirroring status, and other conventional file information, such as ownership, access mode, and last access time, etc. Like UNIX file systems, the access authorization is implemented by checking the ownership and access mode. Fig. 2 shows a metadata example in CEFT with eight data servers in either storage group.

The striping information is described by the stripe width, the stride block size and the data location. The *location*, an array of size *stripe_width*, records the data server indices on which the data are striped. In this example, the file is striped across three data servers, i.e., Nodes 1, 2 and 7, with a striping block size of 64 KB. While the *strip_width* is given by clients, the values of *location* are assigned by the metadata server to approximately balance the disk space utilization on each data server.

The *dstatus*, an array of size *stripe_width*, describes the mirroring status between two groups of mirroring servers that a file is striped on. More precisely, it is defined according to the status of data blocks, shown as follows:

$$dstatus(i) = \begin{cases} 1 & \text{if on } location(i) \text{ of primary group,} \\ 2 & \text{if on } location(i) \text{ of backup group,} \\ 3 & \text{if on } location(i) \text{ of both groups,} \\ 0 & \text{if not on } location(i) \text{ of both groups,} \end{cases}$$

where $1 \leqslant i \leqslant stripe\_width$.

3.4. Metadata backup and the naming mechanism

Metadata server holds the most critical information about striping and authorization. The failure of the metadata server will crash the whole storage system. Therefore, the metadata server needs to be backed up to improve reliability. However, the original PVFS cannot achieve the backup of the metadata server due to the limitation of its naming mechanism for the striped files. In PVFS, the striped data in a data server are sieved together and stored as a file. In addition, the file name is chosen to be the inode number of the metadata file to guarantee the uniqueness of the file name in the data servers. One significant disadvantage of the naming mechanism based on inode numbers is that the system may mistakenly backup the meta server since the data of a new file will be falsely written into an existing file when the primary metadata server is down and the backup metadata server assigns the new file an inode number that has been used by the primary metadata server.

In the design of CEFT, we have changed the naming mechanism and instead used the MD5 sum [58] of the requested file name as the data file name. In this way, the metadata can be directly duplicated to any backup storage device to provide redundancy. An analysis similar to Ref. [47] can prove that in practice the problem of MD5 hash collisions does not arise in our naming mechanism.

The calculation of MD5 will not introduce significant overhead in CEFT. First, we only need to calculate the MD5 of file names, which are typically 5–20 bytes. While we measured that the MD5 program can calculate with a speed of 200 MB/s on a single node, the calculation of a file name usually takes only 25–100 ns. Second, the MD5 calculation is not the bottleneck since it is performed distributively by client nodes. Each client node calculates the MD5 of its destination file name and sends the result along with its I/O requests to the metadata server so that the metadata server can directly extract it from the requests.

3.5. Data consistency

The I/O traces of scientific applications show a frequent pattern in which multiple clients concurrently access the same files [30]. In CEFT, we employ a centralized byte-range

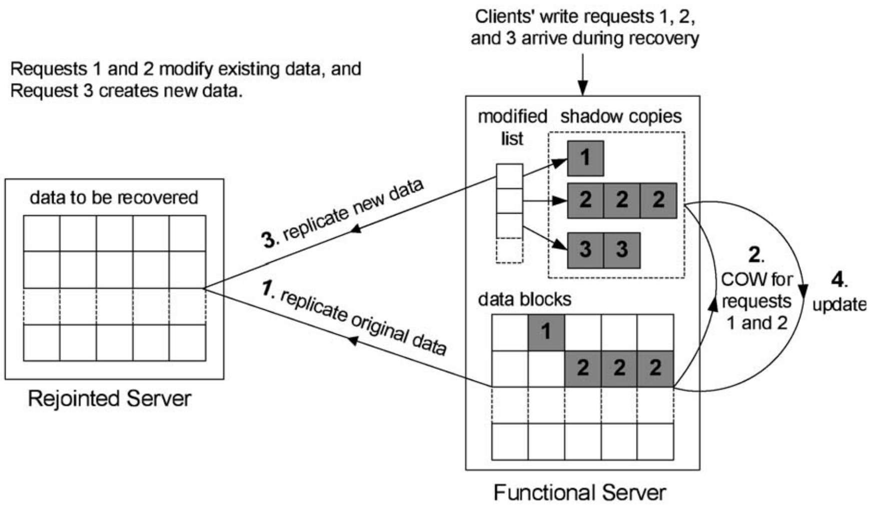

Clients' write requests 1, 2, and 3 arrive during recovery
Requests 1 and 2 modify existing data, and Request 3 creates new data.
modified list
shadow copies
data to be recovered
3. replicate new data
1. replicate original data
2. COW for requests 1 and 2
4. update
data blocks
Rejointed Server
Functional Server

Fig. 3. The procedure of data recovery invoked when a failed server is rebooted after it has been successfully repaired.

lease-based mechanism to allow simultaneous accesses to different portions of a single file while maintaining the multiple-reader single-writer semantics to each requested data portion. A lease is essentially a timed lock that gives its holder specified rights over the property for a limited period of time [23]. Leases are not based on file blocks, instead, they are based on the logical starting and ending addresses in bytes within destination files, thus allowing a more flexible and fine grained consistency control. When the metadata server receives a write request, it looks at the desired portion (addressed in bytes) of the targeted file and checks whether all the bytes in the desired range have not been locked by other clients. If no, the metadata server will issue an exclusive write-lease to the client to permit the write access. Multiple read-leases can be issued to different read-only requests as long as no conflicting write-lease exists. The lease mechanism can reduce the overhead of consistency maintenance. After the clients are granted the access, they continue to hold this access grant for a short period of time in a hope to save the negotiation with the metadata server for the immediate accesses of the same data. This access grant is revoked by the metadata server before the short period expires if other clients are waiting. The centralized management of locking certainly limits the parallelism of I/O operations. However, as discussed in Section 5, the metadata server is not likely the bottleneck under our measurements, a similar observation was found in the GFS [20].

3.6. Data recovery

After the reboot of a failed server, all the data on this server should be recovered. The recovery process in CEFT is simple and fast since all the data can be directly read from its mirrored server without doing any calculations. However, write requests may arrive at the functional servers within the period of recovery and the interleaving of these write operations and the duplication operations can potentially render the data on the server being recovered inconsistent. A simple remedy is to lock the corresponding functional servers and prohibit write operations until the duplication has finished. Clearly, this will make the I/O services unavailable for writes during the recovery process while the data on the servers are still accessible for reads.

To allow uninterrupted services, an on-line snapshot technique, called “copy-on-write (COW)” [13], is deployed in CEFT, as shown in Fig. 3. When a server is repaired and rejoins the system, the recovery process is automatically invoked. First, data are replicated from the mirrored functional server (Step 1). When a write request arrives on the functional server to modify existing data during the recovery period, a shadow copy of the target data blocks is created and updates are then performed on the shadow copy (Step 2). When a write request creates new data blocks, the new data are also saved into the shadow region, leaving the old content intact. For every I/O write request that arrives during the recovery period, the name of the destination file and the touched byte region within that file are recorded into a list, called the modified list, in the order of the requests’ arrivals. After the old data have been cloned, the new data, pointed to by the data structure in the modified list, will be replicated to the newly repaired server in a sequential order, to eliminate the possible inconsistency that resulted during the recovery process (Step 3). As soon as no files are left in the waiting list, the recovery process can begin to write back the modified or new data from the shadow region into the old image (Step 4). On the functional server, the recovery process assumes a higher priority than the I/O service process to guarantee that the recovery will eventually finish. After the recovery finishes successfully, the COW functions are turned off and the waiting list is reset.

The modified list is saved into disk devices to reduce the possibility of incoherence caused by the failure of the functional server during the recovery processes. However, since typically there is no non-volatile RAM in off-the-shelf cluster nodes, some items in a modified list may be lost from the cache in the events of unpredictable system crashes. One of our future

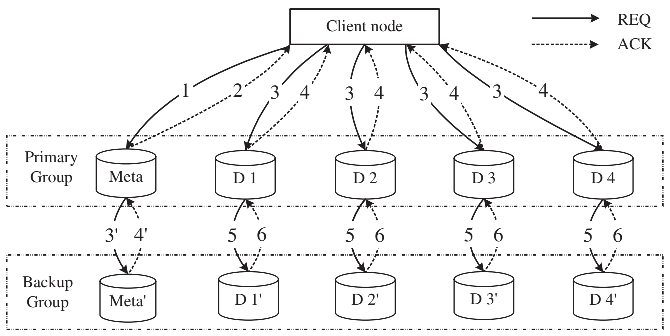


Fig. 4. The steps of duplication process for Protocol 1.

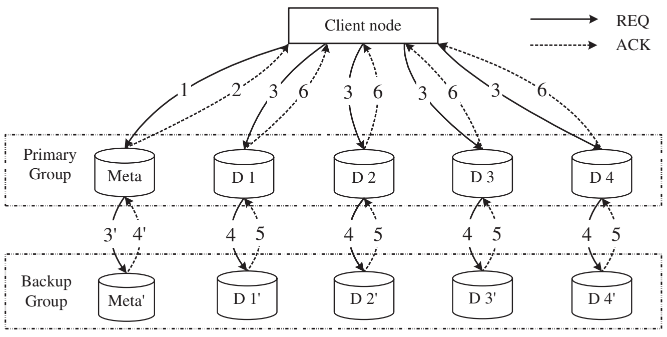


Fig. 5. The steps of duplication process for Protocol 2.

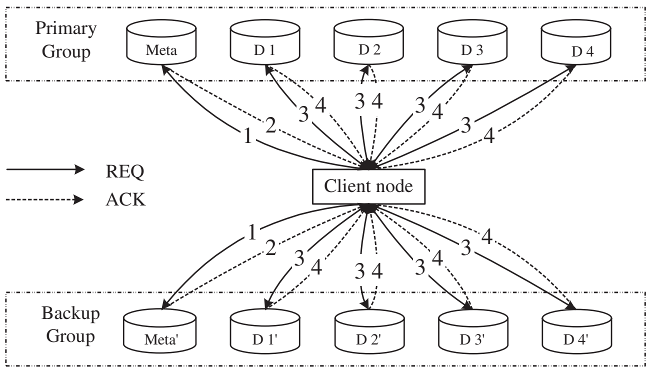


Fig. 6. The steps of duplication process for Protocols 3 and 4.

research directions is to design and implement an efficient tool to check the integrity of the file data and produce a coherent file system state.

4. Duplication protocols

Once a naming mechanism, metadata management, data consistency control and data recovery are in place to facilitate fault tolerance, several different protocol possibilities for data duplication (mirroring) exist for detailed implementation. We have investigated four distinct protocols, which are detailed in this section.

4.1. Protocol 1: asynchronous server duplication

Fig. 4 shows the steps of the duplication process. First, the client fetches the striping information from the metadata server (Steps 1 and 2). Then it writes the data to the primary servers simultaneously (Step 3). Once the primary server receives the data, it immediately sends back an acknowledgment to inform the client of the completion of the I/O process (Step 4). The duplication operation will be performed by the primary servers in the background (Steps 5 and 6). After a backup server receives and stores the data from its primary server, it will send a request to both metadata servers to change the corresponding flag in the *dstatus* array to indicate the completion of the duplication operation. This duplication process can be considered as asynchronous I/O. A potential problem with this protocol is that the new data will be lost if the primary node fails during the duplication operation.

4.2. Protocol 2: synchronous server duplication

Protocol 2 is shown in Fig. 5. As in Protocol 1, the duplication operation is performed by the primary servers. The difference is that the primary servers postpone the acknowledgment to the client until their corresponding backup servers signal the completion of duplication. In addition, the duplication process is pipelined on each data server to speed up the write performance, a technique similar to the one used in the GFSs [20]. More specifically, as soon as a block of striped data from any client arrives at the memory of the primary server, these data will be immediately duplicated to the backup server without waiting for the whole data from that client to reduce disk accesses. This protocol can always guarantee that the data are duplicated to both servers before the client finishes writing. However, this guarantee, and thus an enhanced reliability, comes at the expense of write performance, as analyzed and discussed later in the paper.

4.3. Protocol 3: asynchronous client duplication

In this protocol, the duplication task is assigned to the client, as shown in Fig. 6. After fetching, the client can write to the primary and backup servers simultaneously. The duplication process is regarded as successful after receiving at least one acknowledgment among each pair of mirrored servers. Obviously, there is a potential problem if the slower server in the pair fails before acknowledgment. This problem is similar, but not identical to that in Protocol 1.

4.4. Protocol 4: synchronous client duplication

Protocol 4 is similar to Protocol 3, but it will wait for the acknowledgments from both the primary and the backup servers in each mirrored pair. This protocol can always guarantee that the new data will be stored in both servers of the pair before the completion of I/O access. Similar to the trade-off between Protocols 1 and 2, there is an obvious performance-reliability trade-off between Protocols 3 and 4.

4.5. Cache effect

In these protocols, the file system caches on the servers are fully utilized to improve the overall I/O performance. Thus there is a possibility, however small, that data can be lost when the disk or the server node crashes before the cache data are written onto the disks. Nevertheless, if a "hard" reliable storage system is required, we can potentially use techniques such as forced disk writes in these four duplication protocols. While the four protocols with forced disk writes improve the reliability, the penalty on the I/O performance is too heavy to make the forced disk writes appealing. In addition, even if the forced disk writes are used, these four duplication protocols still present different performance and reliability.

5. Experimental results in CEFT

5.1. Experimental environments

The performance results presented here are measured on the PrairieFire cluster [46] where CEFT has been implemented and installed, at the University of Nebraska—Lincoln. At the time of our experiment, the cluster had 128 computational nodes, each with two AMD Athlon MP 1600 processors, 1 GB of RAM, a Myrinet card and a 20 GB IDE(ATA100) hard drive. Under the same network and system environment as CEFT, the *ttcp* [57] benchmark reports a TCP bandwidth of 112 MB/s using a 1KB buffer with 46% CPU utilization. The disk write bandwidth is 32 MB/s when writing 2 GB of data, according to the Bonnie [7] benchmark.

5.2. Benchmark

A simple benchmark, similar to the one used in Refs. [11,36,56,16], was used to measure the overall concurrent write performance of this parallel file system. Fig. 7 gives a simplified MPI program of this benchmark. The overall and raw write throughput are calculated. The overall write throughput includes the overhead of contacting the metadata server while the raw write throughput does not include the open and close time and measures the aggregate throughput of the data servers exclusively. In both measurements, the completion time of the slowest client is considered as the overall completion time. While this benchmark may not reveal complete workload patterns of real applications, it allows a detailed and fair comparison of the performance of PVFS and the four duplication protocols.

The aggregate write performance is measured under three server configurations, 8 data servers mirroring 8, 16 data servers mirroring 16, and 32 data servers mirroring 32, respectively. With the metadata servers included, the total numbers of servers in the three configurations become 18, 34 and 66. In the three sets of tests, each client node writes a total amount of 16 MB to the servers, i.e., it writes 2, 1 and 0.5 MB to each server node, respectively, which are the approximate amounts of data written by a node during the checkpointing process of a real astrophysics code [51]. During the measurements, there were other computation applications running on our cluster, which shared the node resources, such as network, memory, processors and I/O devices, with the CEFT, and thus the aggregate write performance was probably degraded. In order to reduce the influence of these applications on the performance of these protocols, many measurements were repeated at different times and the average value is calculated after discarding the five highest and five smallest measurements.

```
for all clients:
 1. synchronize with all clients using MPI barrier;
 2. $t_1$ = current time;
 3. open a file;
 4. synchronize with all clients using MPI barrier;
 5. $t_2$ = current time;
 6. loop to write data;
 7. $t_3$ = current time;
 8. close the file;
 9. $t_4$ = current time;
10. $ct_1 = t_4 - t_1$; /* overall completion time */
11. $ct_2 = t_3 - t_2$; /* raw completion time */
12. send $ct_1$ and $ct_2$ to client 0;

for client 0:
 1. find maximum of $ct_1$ and $ct_2$ respectively;
 2. calculate overall write throughput using maximum $ct_1$;
 3. calculate raw write throughput using maximum $ct_2$;
```

Fig. 7. Pseudocode of the benchmark.

5.3. The metadata server overhead

The overall and raw write throughput is measured in CEFT with a configuration of eight data servers mirroring eight under two access patterns: all clients concurrently write different files and all clients concurrently write different portions of the same file. Fig. 8 plots the overall and raw write performance of Protocol 2 as a function of the number of client nodes, in which all clients write data into the same file and different files, respectively.

As the experiment indicates, the aggregate write performance increases with the number of client nodes and reaches its maximum values when the cache at the data server side achieves best utilization. When the client number continues to increase, the aggregate write performance will decrease since on the data server side the context-switching overhead among different I/O requests increases while the benefit of cache decreases. The aggregate throughput will eventually saturate the disk throughput.

An important observation from these figures is that the performance gap between overall and raw write throughput does not increase significantly with the total number of clients. This implies that the metadata server is most likely not the performance bottleneck even when that client number is 100, close to the total available client number of 128 in our cluster.

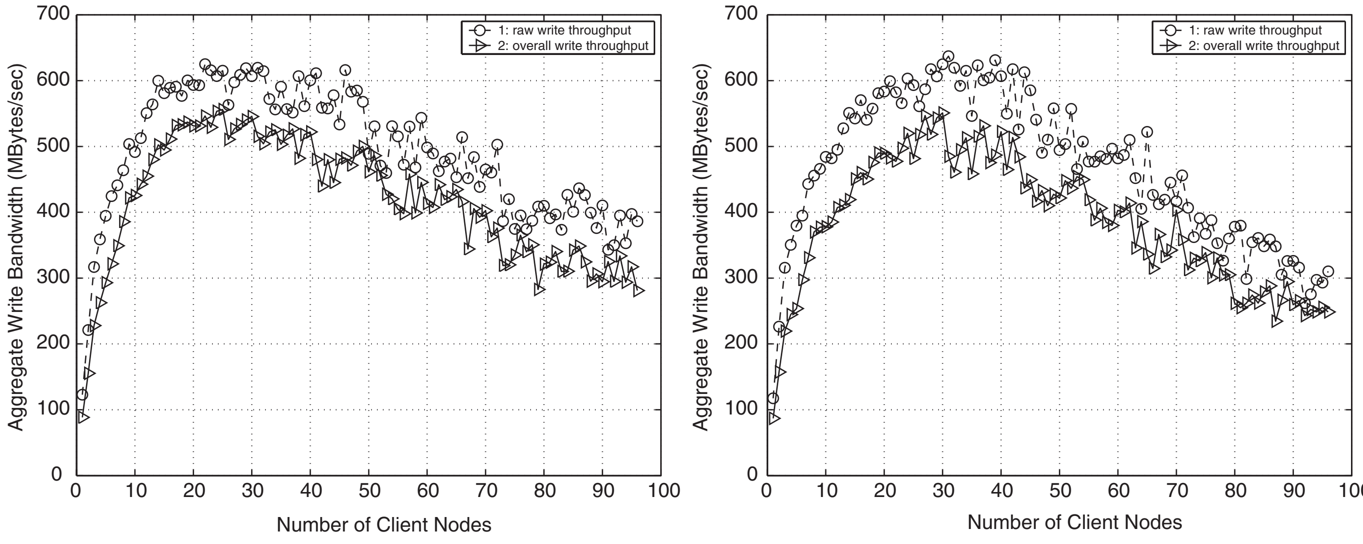


Fig. 8. Aggregate write performance when all clients write to the same file and different files, respectively, using Synchronous Server Duplication with 8-mirroring-8 data servers (20 measurements, discarding 5 highest and 5 smallest).

Experimental results of the other three protocols also show the same pattern of performance gap between the overall and raw throughputs. This further validates the claim made in [31,11] that the metadata server only introduces insignificant performance degradation and is not the performance bottleneck in a moderate-size cluster. Similar observation is made in GFS that also employs the design of a single metadata server (called master) to provide terabyte-scale storage. GFS runs across thousands of disks on over a thousand machines and it is concurrently accessed by hundreds of clients. Their experiments show that the metadata server is not the performance bottleneck under the heavy web searching workload in Google.

5.4. Write performances of the four duplication protocols

The overall write performance of the four duplication protocols and PVFS are measured in the three server configurations using the benchmark and workload described previously. Figs. 9–11 show their average performances over 70 measurements, in which the 5 highest and 5 lowest are discarded. When there is only one client node, Protocols 1–3 perform almost identically, where the bottleneck is likely to be the TCP/IP stack on the client node. In contrast, Protocol 4 performs the worst since it is at a double-disadvantage: first, the client node that is already the bottleneck must perform twice as many writes; second, it has to wait for the slowest server node to complete the write process.

In Protocol 2, the write process from the clients to the primary group and the duplication process from the primary group to the backup group are pipelined and thus the performance is only slightly inferior to that of Protocol 1 when the primary server is lightly loaded (e.g., with fewer than five clients). As the workload on the primary server increases, the performance of Protocol 2 lags further behind that of Protocol 1.

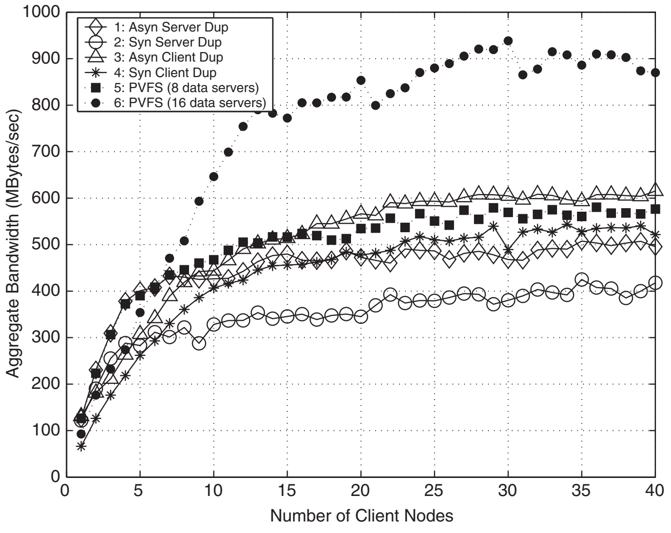


Fig. 9. Write performance when 8 I/O data servers mirror another 8 I/O data servers (70 measurements, discarding 5 highest and 5 smallest).

When the number of client nodes is smaller than the number of server nodes, Protocols 1 and 2 outperform Protocols 3 and 4, since more nodes are involved in the duplication process in the first two protocols than in the last two. On the other hand, when the number of client nodes approaches and surpasses the number of server nodes in one group, the situation reverses itself so that Protocols 3 and 4 become superior to Protocols 1 and 2. To achieve a high write bandwidth, we have designed a hybrid protocol, in which Protocol 1 or 2 is preferred when the client node number is smaller than the number of server nodes in one group, and otherwise Protocol 3 or 4 is used. When the reliability is considered, this hybrid protocol can be further modified to optimize the balance between reliability and write bandwidth. This will be explained in detail later in this paper.

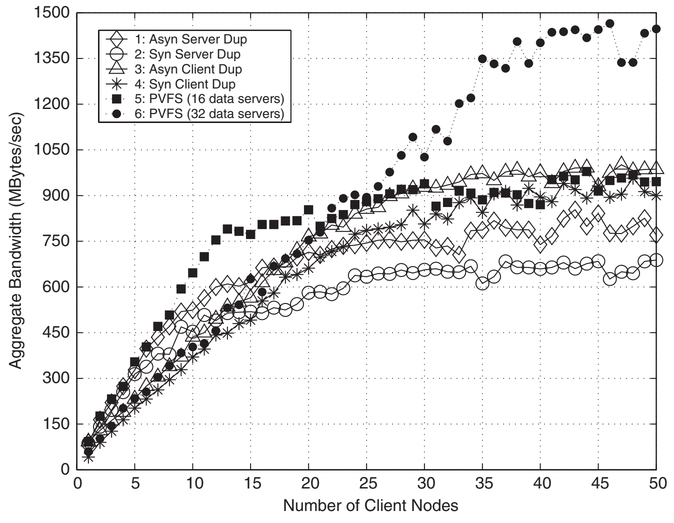


Fig. 10. Write performance when 16 I/O data servers mirror another 16 I/O data servers (70 measurements, discarding 5 highest and 5 smallest).

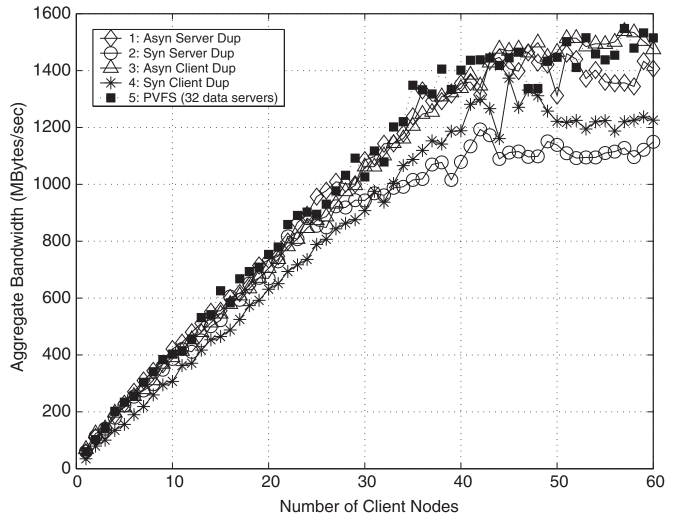


Fig. 11. Write performance when 32 I/O data servers mirror another 32 I/O data servers (70 measurements, discarding 5 highest and 5 smallest).

Table 1 summarizes the average peak aggregate write performance of the four protocols in the saturation region, along with their performance relative ratio to the PVFS with half the number of data servers and the same number of data servers, respectively. The aggregate write performance of Protocol 1 is nearly 30%, 28% and 25% better than that of Protocol 2 under the three server configurations, respectively, with an average improvement of 27.7%. The performance of Protocol 3 is nearly 14%, 7% and 23% better than that of Protocol 4, under the three configurations, respectively, with an average improvement of 14.7%. While the workload on the primary and backup groups are well balanced in Protocols 3 and 4 due to the duplication symmetry initiated by the client nodes, in Protocols 1 and 2 the primary group bears twice the amount of workload as the backup group because of the asymmetry in the duplication process. As a result, the peak performance of Protocol 3 is better than that of Protocol 1, while Protocol 4 outperforms Protocol 2 consistently.

In addition, experiment results show that the peak performance of Protocol 4 is only around 7% less than PVFS with half the number of servers. As shown in Fig. 9, when the total number of clients is less than four, the write bandwidth of Protocol 4 is around 50% of PVFS due to the doubled network traffic at the client side. When the number of clients increases, the performance gap between CEFT and PVFS begins to decrease. This is because the bottleneck gradually shifts from the client side to the server side. When the bottleneck completely shifts to the servers, the doubled network traffic on each client does not have significantly negative impact on the aggregate bandwidth. The 7% overhead, we believe, is mainly caused by the delay in waiting for the acknowledgements from both servers.

Compared with the PVFS with the same number of data servers, the server driven Protocols 1 and 2 improve the reliability at the expense of 46–58% write bandwidth and the client driven Protocols 3 and 4 cost around 33% and 41% write bandwidth, respectively. Compared with the PVFS with half the number of data servers, as shown in Table 1, such cost is not only acceptable in most cases, but it is also at times negligible or even negative, especially for Protocol 3. In Protocol 3, when the total number of clients is large enough, the extra work of duplication at the client side will not influence the aggregate write performance since the data servers have already been heavily loaded and their I/O bandwidth have been saturated. Furthermore, the application running on a client node will consider its write operations completed as long as the client has received at least one acknowledgment among each mirroring pair, although some duplication work may still proceed, transparent to the application. Since the data servers are not dedicated and their CPU, disks, memory and network load are different, Protocol 3 chooses the response time of the less heavily loaded server in each mirroring pair and thus surpasses the PVFS with half the number of data servers.

5.5. *Read performance and real application benchmark*

A similar microbenchmark is also used to evaluate the read performance [65,66]. In addition, we propose to use the techniques of doubling the degree of parallelism and hot-spot skipping to improve the aggregate read performance. The read performance is boosted by scheduling requests on both mirroring groups in order to double the degree of parallelism. In the case that a node becomes a hot spot, this node is skipped and all the data are read from its mirror node. Extensive experiments in a real cluster environment, where each data server is not dedicated but time-shared with compute tasks, indicate that both techniques are highly effective.

We also conduct a case study for a popular read-I/O intensive application, namely, parallel BLAST [17], and use this application as a benchmark to evaluate the techniques proposed in CEFT [68,67]. We aim to investigate the performance impact of the degree of I/O parallelism and the contention of the I/O resource on scientific applications. Experiments show that CEFT

Table 1
Average peak write performance and ratio to the performances of PVFS with half nodes

Protocol	Number of data servers in one group					
	8		16		32	
	MB/s	%	MB/s	%	MB/s	%
1 (Server asynchronous duplication)	492	87	796	86	1386	94
2 (Server synchronous duplication)	391	68	660	71	1114	75
3 (Client asynchronous duplication)	604	106	974	104	1501	101
4 (Client synchronous duplication)	528	93	905	97	1218	82
5 (PVFS with half # of nodes)	567	100	929	100	1482	100
6 (PVFS with same # of nodes)	929	164	1482	160	—	—

can exploit parallel I/O to significantly reduce the running time of this application, and the read optimization techniques of doubling the degree of parallelism and skipping hot-spot nodes are highly effective in improving the aggregate throughput.

6. Reliability and availability analysis

In this section, a Markov-chain model is constructed to analyze the reliability and availability of the four duplication protocols, and to compare their reliability with that of the PVFS.

Markov models have been used to analyze the reliability of RAID-1 in Refs. [21,39,2,8,37]. However, none of these models distinguishes the primary disk failures from the backup disk failures, i.e., they assume that all the data on a failure disk can be recovered from its mirror disk. This assumption holds true in a tightly coupled array of disks, such as RAID, because data on primary and backup disks are always kept consistent with the help of hardware. However, this assumption may not be true in our loosely coupled distributed system, such as clusters, in which the failure of a primary server and a backup server have different implications. For example, in Protocol 1, if a primary server fails before the completion of duplication, the backup server will lose the data that has not been duplicated. But the system does not lose any data if only a backup node fails. Therefore, in our system, the primary and the backup server nodes are not symmetrical in terms of their failure implications and the classic RAID model cannot be used. In addition to being able to reflect the asymmetry, our model should be general enough so that the reliability of all four protocols can be derived directly. In the following sections, we take Protocol 1 as an example to show how the Markov-chain model is developed and how it can be applied to other protocols by appropriately changing some relevant definitions.

To simplify the analysis, the following assumptions are made:

(1) In this model, we neglect the data loss caused by the failures of nodes or disks that happen before the data in the cache are written onto the disks since the cache size is relatively small and the local file system on each data server usually periodically flushes modified (dirty) blocks back to disks. In most UNIX/Linux file systems, every 30 s, all dirty blocks that have not been modified in the last 30 s are written back onto disks [38,35,10]. We understand that this assumption is somewhat unrealistic and may lead to an overestimate of the reliability.
(2) Network and node failures are all independent and follow an exponential distribution. Ref. [22] has studied the exponential, Weibull, and Gamma models of disk lifetime distributions and concluded that exponential distributions are sufficient. The failure distribution of a cluster node, which incorporates both hardware and software failures, can also be reasonably modeled as exponential distribution [61,54]. This assumption might not be realistic in some situations, such as power surges, burst of I/O tasks, etc.
(3) Write requests arrive at the primary server from the clients following the Poisson process, with an exponentially distributed inter-arrival time whose mean value is referred to in this paper as the mean-time-to-write (*MTTW*). Refs. [40,59,32] provide justifications for the assumption that the I/O access patterns in scientific applications exhibit Poisson arrival rates and thus can be modeled as Markov processes. Ref. [24] shows that, strictly speaking, file system traffic is not self-similar in nature and this further assures us the appropriateness of the Poisson assumption.
(4) Similarly, the duplication time is also assumed as a random variable, following an exponential distribution, whose value depends on the data size, network traffic, workload on both the primary server and the backup server, etc. Its mean time interval is referred to as the mean-time-to-duplicate (*MTTD*) in this paper.

Table 2 presents some basic notations, while others will be introduced appropriately during the discussion.

6.1. Calculation of $\mathbb{P}_c$

According to the given assumptions, we know that write requests arrive in the duplication queue with an arrival rate of λ_w and leave the queue with a duplication rate of μ_d. For the system to be stable, it is implied that $\lambda_w < \mu_d$, otherwise the length of the duplication queue will grow to infinity, causing the system to saturate. If the number of requests in the queue is zero, we say that the data in the primary node are consistent with the backup node. This duplication queue can be modeled by an $M/M/1$ queuing model [53,14]. In the model, the probability of the consistent state, i.e., the probability of an empty queue,

Table 2
Notation

N	Total number of nodes in one group
S	Total number of Markovian states
i, j	Index of Markovian states, $1 \leqslant i, j \leqslant S$
m, n	Number of failed nodes, $0 \leqslant m, n \leqslant N$
λ	Failure rate per node
λ_s	Failure rate of the network switch
λ_w	Arrival rate of write requests per server
μ	Repair rate per node
μ_d	Duplication rate
$MTTF_{\text{node}} = \frac{1}{\lambda}$	Mean time to failure per node
$MTTF_{\text{switch}} = \frac{1}{\lambda_s}$	Mean time to failure per switch
$MTTW = \frac{1}{\lambda_w}$	Mean time to write
$MTTR_{\text{node}} = \frac{1}{\mu}$	Mean time to repair per node
$MTTD = \frac{1}{\mu_d}$	Mean time to duplicate
$MTTDL$	Mean time to data loss
$\mathbf{M}$	Markovian fundamental matrix
$\mathbf{Q} = [q_{ij}]_{S\times S}$	Markovian truncated matrix
$\mathbb{P}_c$	Probability that a primary node is consistent with its mirror node
$\mathbb{P}(mPnB)$	Probability of the system being still functional when m primary nodes and n backup nodes have failed
$\binom{n}{k} = \frac{n!}{(n-k)!k!}$	Binomial coefficient

can be calculated as follows:

$$\mathbb{P}_c = 1 - \frac{\lambda_w}{\mu_d} = 1 - \frac{MTTD}{MTTW}. \quad (1)$$

Although $\mathbb{P}_c$ is derived based on the duplication process of Protocol 1, this term can also be used in other protocols. In Protocols 2 and 4, all data have already been duplicated to the mirror nodes at the time when the client nodes complete the writing access. Thus *MTTD* can be thought to be 0. In Protocol 3, at the time the client finishes the writing process, there is still a chance that a primary node is not consistent with its backup node. Similarly, it can also be modeled as $M/M/1$ theoretically if we redefine *MTTD* as the difference between the time instants when data are stored in the faster server and when data are stored in the slower server node.

6.2. Markov-chain model for reliability evaluation

Fig. 12 shows the Markov state diagram for Protocol 1, which can also be applied to the other protocols. In this diagram, $i : mPnB$ signifies that the state number/index is i, and there are m and n failed nodes in the primary and backup groups, respectively. All the states shown are working states, with the exception of *DL*, which is the data loss state. The total number of states in the Markov state diagram is denoted by S and is equal to $(N+1)(N+2)/2$. The Markov chain begins with State 1 ($1 : 0P0B$), followed by State 2 ($2 : 1P0B$), and so on.

To facilitate the solution to this model, we derive a function, given in Eq. (2), that maps from the system state with m failed

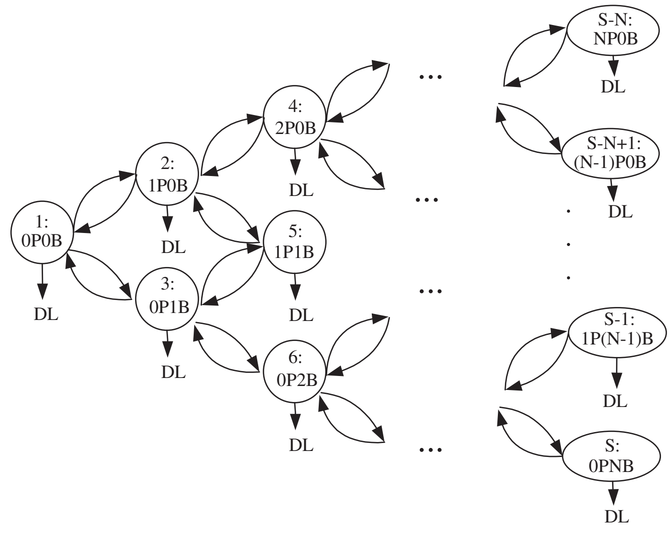


Fig. 12. Markov state diagram for Protocol 1.

primary nodes and n failed backup nodes to the state index i of the Markov state diagram:

$$i = \tfrac{1}{2}(m+n)(m+n+1) + (n+1). \quad (2)$$

Similarly, the inverse mapping function is given in

$$n = i - 1 - \frac{x(x+1)}{2}, \quad (3)$$

$$m = x - n, \quad (4)$$

where $x = \left\lceil \frac{\sqrt{8i+1}-3}{2} \right\rceil$.

Fig. 13 shows the transition rate between the neighboring states. In the diagram, $\mathbb{P}_{ij}$ denotes the probability that the system remains functional, also referred to as *safety probability*, given that one more primary node fails while m primary nodes and n backup nodes have already failed. Similarly, $\mathbb{P}_{ik}$ denotes the probability, or safety probability, of the system remaining functional when one more backup node fails while m primary nodes and n backup nodes have already failed. $\mathbb{P}_{ij}$ can be calculated as

$$\begin{aligned}\mathbb{P}_{ij} &= \mathbb{P}((m+1)PnB \mid mPnB) \\ &= \frac{\mathbb{P}(((m+1)PnB) \cap (mPnB))}{\mathbb{P}(mPnB)} \\ &= \frac{\mathbb{P}((m+1)PnB)}{\mathbb{P}(mPnB)}, \end{aligned} \quad (5)$$

where $\mathbb{P}$ is the safety probability when m nodes in the primary group and n nodes in the backup group fail simultaneously. The calculation of $\mathbb{P}$ is as follows:

$$\mathbb{P}(mPnB) = \begin{cases} \dfrac{\binom{N}{m+n}2^{m+n}}{\binom{2N}{m+n}} & \text{if } m+n \leqslant N, \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

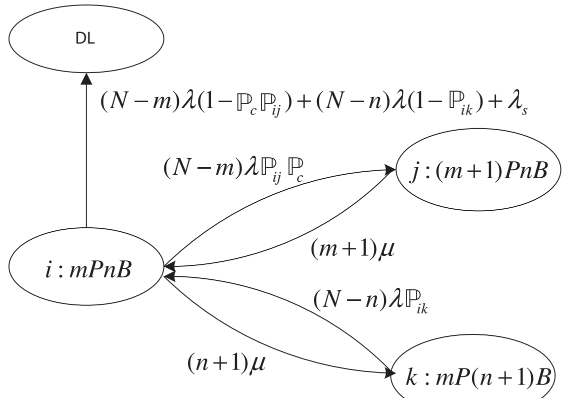

DL
$(N-m)\lambda(1-\mathbb{P}_c\mathbb{P}_{ij})+(N-n)\lambda(1-\mathbb{P}_{ik})+\lambda_s$
$(N-m)\lambda\mathbb{P}_{ij}\mathbb{P}_c$
$j:(m+1)PnB$
$(m+1)\mu$
$i:mPnB$
$(N-n)\lambda\mathbb{P}_{ik}$
$(n+1)\mu$
$k:mP(n+1)B$

Fig. 13. The transition probability between different states.

Similarly, we have

$$\mathbb{P}_{ik} = \frac{\mathbb{P}(mP(n+1)B)}{\mathbb{P}(mPnB)}. \tag{7}$$

The transition probability from State i to the data loss state, denoted as $q_{i,\mathrm{DL}}$, can be calculated as

$$\begin{aligned} q_{i,\mathrm{DL}} &= \text{loss caused by one more primary node failure} \\ &\quad + \text{loss caused by one more backup node failure} \\ &\quad + \text{loss caused by network switch failure} \\ &= (N-m)\lambda[(1-\mathbb{P}_{ij}) + \mathbb{P}_{ij}(1-\mathbb{P}_\mathrm{c})] \\ &\quad +(N-n)\lambda(1-\mathbb{P}_{ik}) + \lambda_\mathrm{s} \\ &= (N-m)\lambda(1-\mathbb{P}_\mathrm{c}\mathbb{P}_{ij}) \\ &\quad +(N-n)\lambda(1-\mathbb{P}_{ik}) + \lambda_\mathrm{s}. \end{aligned} \tag{8}$$

The stochastic transitional probability matrix is defined as $\mathbf{Q} = [q_{ij}]$, where $1 \leqslant i, j \leqslant S$ and q_{ij} is the transition probability from State $i : m_i P n_i B$ to State $j : m_j P n_j B$. In summary, q_{ij} can be calculated as follows:

If $i < j$, then

$$q_{ij} = \begin{cases} (N-m_i)\lambda\mathbb{P}_{ij}\mathbb{P}_\mathrm{c} & \text{if } m_j = m_i + 1 \text{ and } n_j = n_i, \\ (N-m_i)\lambda\mathbb{P}_{ij} & \text{if } m_j = m_i \text{ and } n_j = n_i + 1, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

If $i > j$, then

$$q_{ij} = \begin{cases} m_j\mu & \text{if } m_j = m_i + 1 \text{ and } n_j = n_i, \\ n_j\mu & \text{if } m_j = m_i \text{ and } n_j = n_i + 1, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

If $i = j$, then

$$q_{ii} = 1 - \sum_{i=1, j\neq i}^{j\leqslant S} q_{ij} - q_{i,\mathrm{DL}}. \tag{11}$$

If $\mathbb{P}_\mathrm{c} = 1$, i.e., the primary node and backup node are always kept consistent, like in RAID-1, and a fault-free network is assumed, the model shown in Fig. 12 can be simplified to the classic RAID-1 model [2], as shown in Fig. 14. This is proven by the fact that numerical results generated by both models with the same set of input parameters are identical.

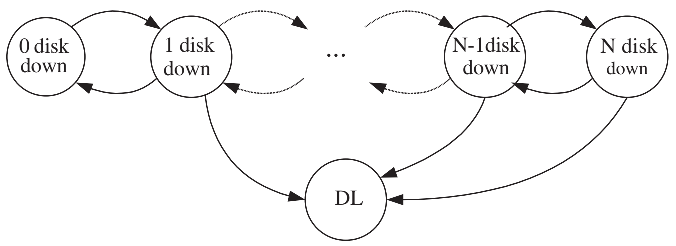


Fig. 14. Classic Markov state diagram of RAID-1.

6.3. Calculation of MTTDL

MTTDL can be obtained from the fundamental matrix **M**, which is defined by [4].

$$\mathbf{M} = [m_{ij}] = [I - \mathbf{Q}]^{-1}, \tag{12}$$

where m_{ij} represents the average amount of time in State j before entering the data loss state, when the Markov chain starts from State i.

The total amount of time expected before being absorbed into the data loss state is equal to the total amount of time it expects to make to all the non-absorbing states. Since the system starts from State 1, where there are no node failures, MTTDL is the sum of the average time spent on all states j ($1 \leqslant j \leqslant S$), i.e.,

$$MTTDL = \sum_{j=1}^{S} m_{1j}. \tag{13}$$

When $MTTD = 0$ and $MTTF_\mathrm{switch} = \infty$, our model becomes the classic model for RAID-1. If $MTTD = \infty$ and $MTTF_\mathrm{switch} = \infty$, it then becomes the classic model for RAID-0. When using the same *MTTF* and *MTTR* to calculate the *MTTDL* of RAID-0 and RAID-1 as Ref. [2], our model shows identical results to those given in the above references.

To further validate our model, Fig. 15 shows the relationship between *MTTDL* and *MTTD* under different workload conditions in an CEFT where there are eight data server nodes in either group. The *MTTDL* in this figure is calculated based on our model built above. This figure indicates that the *MTTDL* decreases with an increase in *MTTD*. With the same *MTTD* but increasing *MTTW*, *MTTDL* increases. All of these performance trends are intuitive and realistic.

6.4. Reliability analysis

The numerical results, calculated according to the Markov chain model, show the significant impact of the mean-time-to-duplication on the whole system reliability, measured in terms of mean-time-to-data-loss, under different workload conditions. As the model indicates, the reliability of CEFT depends on the write frequencies of the client nodes. The more frequently the client nodes write data into the storage nodes, the higher the

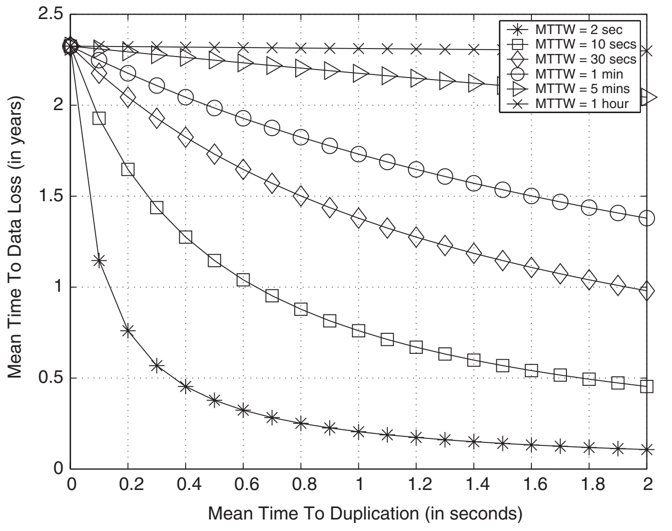


Fig. 15. Influence of *MTTD* on *MTTDL* of eight mirroring eight data servers under different workloads ($MTTF = 1$ year, $MTTF_{\text{switch}} = 3$ years, $MTTR = 2$ days and $MTTW = 5$ min).

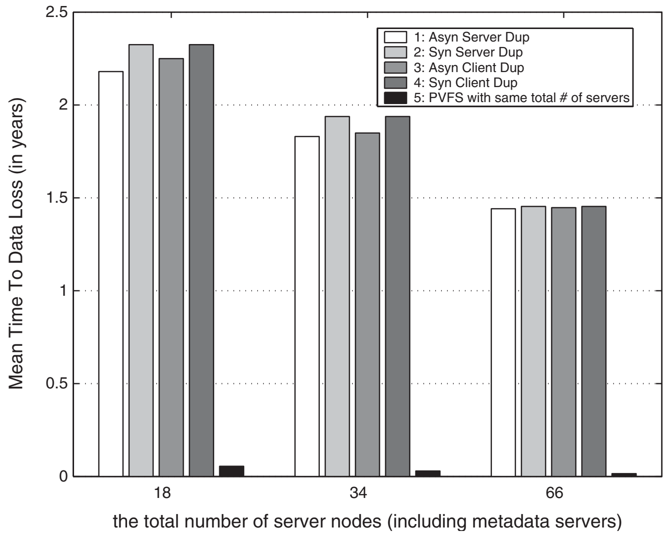


Fig. 16. Reliability comparison of CEFT and PVFS.

probability that the primary storage group remains inconsistent with the backup group, thus giving rise to increased likelihood of data loss due to the failure of some nodes in the storage group. The write frequency, measured as mean-time-to-write, is highly dependent on the applications running on the client nodes.

To quantitatively compare the reliability of the four duplication protocols, we evaluate their reliability in the scenario of a simple benchmark presented in Section 5. Although this simple benchmark does not reflect all applications that run on CEFT, it gives a quantitative and fair comparison of these duplication protocols. We recorded the time instants of all the events on all server and client nodes and stored them into the files so that we could calculate the *MTTW* and *MTTD* of this simple benchmark. The *MTTD* of Protocol 1 can be directly calculated from the trace files. The *MTTD* of Protocols 2 and 4 can be regarded as 0 since the data are consistent as soon as the client node finishes the write process. To obtain the *MTTD* of Protocol 3 is tricky because the duplication process is performed by the client nodes. In Protocol 3, we define *MTTD* as the mean time difference between the arrivals of the acknowledgments from the primary node and the backup node.

We assume that $MTTF = 1$ year, $MTTF_{\text{switch}} = 3$ years and $MTTR = 2$ days. In the simple benchmark, $MTTW = 1$ min. We calculate the *MTTDL* curve as a function of the number of server nodes for the four protocols under the three server configurations. Fig. 16 compares the reliability between CEFT and PVFS and compared with their *MTTDL*, on average the four duplication protocols improve the reliability of PVFS by a factor of 41, 64 and 96 in the three server configurations, respectively. In addition, Protocol 1 is 93%, 93% and 99% of Protocols 2 and 4 under the three different server configurations, respectively, with an average degradation of 5%. Protocol 3 is 96%, 94% and 99% of Protocols 2 and 4, with an average degradation of 3.3%.

6.5. Availability analysis

Availability is defined in this paper to be the fraction of time when a system is operational. More precisely, it is defined as follows:

$$Availability = \frac{MTTF}{MTTF + MTTR}. \tag{14}$$

Figs. 17 and 18 give the availability comparisons between the four duplication protocols and PVFS within the same scenarios as the reliability analysis. While the availability of PVFS is only 0.91, 0.85 and 0.73 in the three server configurations, respectively, the availability of CEFT with four duplication protocols are all above 0.99. Similarly with the reliability comparisons, Protocols 2 and 4 achieve a better availability than Protocols 1 and 3. Note that a small difference in the availability does have a significant impact in practice [42].

6.6. Optimization of the tradeoffs

As the measurement and analytical results indicate, if the number of client nodes is smaller than the number of server nodes, server-driven protocols tend to have a higher write performance than the client-driven protocols since more nodes are involved in sharing the duplication work. Between the server-driven protocols, the synchronous one is preferred because it has a higher reliability with only slightly lower bandwidth. On the contrary, if the total number of the client nodes is greater than that of the server nodes, the client-driven protocols are better than their server-driven counterparts. Between the client-driven protocols, the asynchronous client duplication is the most favorable since it has the highest write performance and the second best reliability. These observations lead us to propose a hybrid protocol to optimize the tradeoff between the reliability and bandwidth performances.

A scientific application is usually required to specify the total number of parallel jobs or clients it needs before running in a

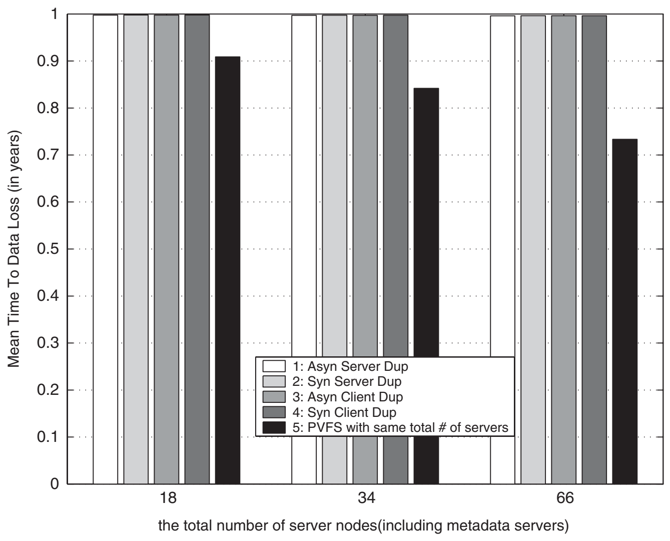


Fig. 17. Availability comparison of CEFT and PVFS.

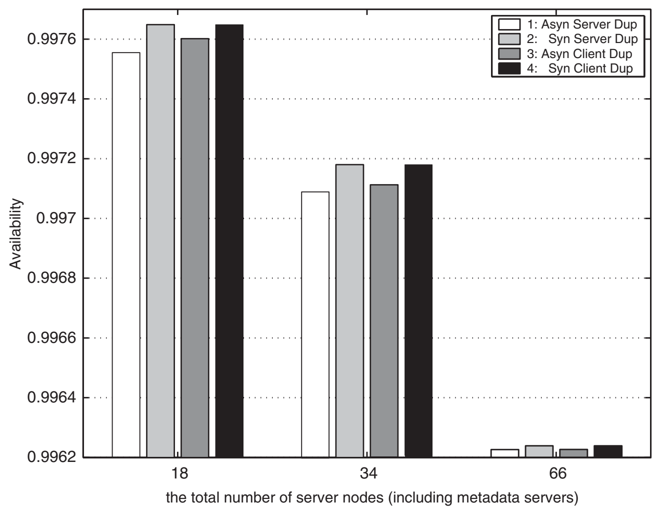


Fig. 18. Availability comparison of four duplication protocols.

cluster. In the hybrid duplication protocol, each client compares the total server number in one storage group with the total number of parallel clients of the current application. If the server number exceeds the client number, the synchronous server duplication is used to mirror the data. Otherwise, the asynchronous client duplication is preferred. In this way, this hybrid protocol always tries to achieve a considerably high write performance or reliability with little degradation of the other.

7. Conclusion and future work

This paper presents the design and implementation of CEFT, a RAID-10 style file system based on PVFS, and proposes four duplication protocols. The efficiencies of these protocols are examined by measuring their write performance in a real cluster and analyzing their reliability and availability based on Markov process modeling.

The study in this paper shows that these proposed protocols have a write performance penalty 33–58% when compared with PVFS with the same total number of servers. In addition, these duplication protocols strike different balances between reliability and write performance. A protocol that has higher bandwidth is most likely to be inferior in reliability. Between the server-driven protocols, the asynchronous one achieves a write performance that is 27.7% higher than the synchronous one, which comes at the expense of an average of 5% reliability degradation. Similarly, between the client-driven protocols, the asynchronous one has a write performance that is 14.7% higher than the synchronous one, while paying a premium of an average of 3.3% reduction in reliability. We also proposed a hybrid protocol that optimizes the tradeoff between the reliability and write performance. In this hybrid protocol, if the total number of jobs of a data-intensive application is less than the server number of one storage group, the synchronous server duplication is used to mirror the data. Otherwise, the asynchronous client duplication is preferred.

None of the proposed protocols employs high-cost but more reliable techniques such as “forced writes” to the disks, and the data that have not been flushed from the cache buffer to the disks will be lost when a node fails. We will further investigate the tradeoff when considering “forced writes”.

Further work is needed to enrich the interfaces of CEFT to applications. While PVFS has three types of interfaces for applications, including native I/O library, MPI I/O library based on ROMIO [50] and NFS type interfaces, CEFT provides its own native I/O libraries. Although both parallel and non-parallel applications can use this native interfaces to successfully access data in CEFT, we are implementing the standard MPI I/O functions and NFS-type kernel modules so that applications can directly run over CEFT without modifying their source code.

Acknowledgments

This work was partially supported by an NSF Grant (EPS-0091900), a University of Maine MEIF startup fund (5254923), a Nebraska University Foundation Grant (26-0511-0019), a University of Nebraska–Lincoln Academic Priority Grant, and A Chinese NSF 973 Project Grant (2004cb318201). Work was completed using the Research Computing Facility at University of Nebraska, Lincoln. Special thanks are also due to the Argonne National Labs and Clemson University for making the source code of PVFS publicly available.

References

[1] T.E. Anderson, M.D. Dahlin, J.M. Neefe, D.A. Patterson, D.S. Roselli, R.Y. Wang, Serverless network file systems, ACM Trans. Comput. Systems 14 (1) (1996) 41–79.
[2] S.H. Baek, B.W. Kim, E.J. Joung, C.W. Park, Reliability and performance of hierarchical RAID with multiple controllers, in: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, 2001, pp. 246–254.
[3] A. Bestavros, Ida-based redundant arrays of inexpensive disks, in: Proceedings of the First International Conference on Parallel and Distributed Information Systems, IEEE Computer Society Press, 1991, pp. 2–9.

[4] R. Billinton, R.N. Allan, Reliability Evaluation of Engineering System: Concepts and Techniques, Perseus Publishing, 1992.

[5] M. Blaum, J. Brady, J. Bruck, J. Menon, Evenodd: an optimal scheme for tolerating double disk failures in raid architectures, in: Proceedings of the 21st Annual International Symposium on Computer Architecture, IEEE Computer Society Press, 1994, pp. 245–254.

[6] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, W.-K. Su, Myrinet: a gigabit-per-second local area network, IEEE Micro 15 (1) (1995) 29–36.

[7] Bonnie, <http://www.textuality.com/bonnie/>, September 2002.

[8] W.A. Burkhard, J. Menon, Disk array storage system reliability, in: Symposium on Fault-Tolerant Computing, 1993, pp. 432–441.

[9] L.-F. Cabrera, D.D.E. Long, Swift: using distributed disk stripping to provide high I/O data rates, Comput. Systems 4 (4).

[10] R. Card, T. Ts'o, S. Tweedie, Design and implementation of the second extended file system, in: Proceedings of the First Dutch International Symposium on Linux, 1994.

[11] P.H. Carns, W.B. Ligon III, R.B. Ross, R. Thakur, PVFS: a parallel file system for Linux clusters, in: Proceedings of the 4th Annual Linux Showcase and Conference, USENIX Association, Atlanta, GA, 2000, pp. 317–327 (best Paper Award).

[12] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, D.A. Patterson, RAID: high-performance, reliable secondary storage, ACM Comput. Surveys (CSUR) 26 (2) (1994) 145–185.

[13] A. Chervenak, V. Vellanki, Z. Kurmas, Protecting file systems: a survey of backup techniques, in: Proceedings Joint NASA and IEEE Mass Storage Conference, 1998.

[14] J.W. Cohen, The Single Server Queue, North-Holland, Amsterdam, 1982.

[15] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, S. Sankar, Row-diagonal parity for double disk failure correction, in: Proceedings of the USENIX FAST '04 Conference on File and Storage Technologies, Network Appliance, Inc., USENIX Association, San Francisco, CA, 2004, pp. 1–14.

[16] R. Cristaldi, G. Iannello, F. Delfino, The cluster file system: integration of high performance communication and I/O in clusters, in: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), Berlin, Germany, 2002, pp. 160–169.

[17] A.E. Darling, L. Carey, W. Chun Feng, The design, implementation, and evaluation of mpiBLAST, in: Proceedings of Cluster World Conference & Expo, 2003.

[18] C. Eddington, Infinibridge: an infiniband channel adapter with integrated switch, IEEE Micro 22 (2) (2002) 48–56.

[19] J. Gafsi, E.W. Biersack, Modeling and performance comparison of reliability strategies for distributed video servers, IEEE Trans. Parallel Distrib. Systems 11 (4) (2000) 412–430.

[20] S. Ghemawat, H. Gobioff, S.-T. Leung, The google file system, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003, pp. 29–43.

[21] G.A. Gibson, Redundant disk arrays: reliable, parallel secondary storage, Ph.D. Thesis, U.C., Berkeley, Berkeley, CA, March 1991.

[22] G.A. Gibson, Redundant Disk Arrays: Reliable, Parallel Secondary Storage, MIT Press, Cambridge, MA, 1992.

[23] C. Gray, D. Cheriton, Leases: an efficient fault-tolerant mechanism for distributed file cache consistency, in: Proceedings of the Twelfth ACM Symposium on Operating Systems Principles, ACM Press, 1989, pp. 202–210.

[24] S.D. Gribble, G.S. Manku, D. Roselli, E.A. Brewer, T.J. Gibson, E.L. Miller, Self-similarity in file systems, in: Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, 1998, pp. 141–150.

[25] J.H. Hartman, J.K. Ouserhout, The zebra striped network file system, ACM Trans. Comput. Systems 13 (3) (1995) 274–310.

[26] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, third ed., Morgan Kauffmann, San Francisco, CA, 2002.

[27] H.-I. Hsiao, D. DeWitt, Chained declustering: a new availability strategy for multiprocessor database machines, in: Proceedings of 6th International Data Engineering Conference, 1990, pp. 456–465.

[28] K. Hwang, H. Jin, R.S. Ho, Orthogonal striping and mirroring in distributed raid for I/O-centric cluster computing, IEEE Trans. Parallel Distrib. Systems 13 (1) (2002) 26–44.

[29] IEEE p802.3ae, 10 GB/s Ethernet Task Force, <http://grouper.ieee.org/groups/802/3/ae/>, 2003.

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

[31] W.B.L. III, R.B. Ross, Implementation and performance of a parallel file system for high performance distributed applications, in: Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (HPDC), Syracuse, New York, 1996.

[32] L.-W. Lee, P. Scheuermann, R. Vingralek, File assignment in parallel I/O systems with minimal variance of service time, IEEE Trans. Comput. 49 (2) (2000) 127–140.

[33] E.K. Lee, C.A. Thekkath, Petal: distributed virtual disks, in: Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, 1996, pp. 84–92.

[34] F.J. MacWilliams, J.J.A. Sloane, The Theory of Error-Correcting Codes, North-Holland, Amsterdam, 1977.

[35] M.K. McKusick, W.N. Joy, S.J. Leffler, R.S. Fabry, A fast file system for unix, ACM Trans. Comput. Systems 2 (3) (1984) 181–197.

[36] S.A. Moyer, 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.

[37] Y. Nam, D.W. Kim, T.Y. Choe, C. Park, Enhancing write I/O performance of disk array RM2 tolerating double disk failures, in: International Conference on Parallel Processing, 2002, pp. 211–218.

[38] M.N. Nelson, B.B. Welch, J.K. Ousterhout, Caching in the sprite network file system, ACM Trans. Comput. Systems 6 (1) (1988) 134–154.

[39] Y. Oh, S. Kim, J.-H. Kim, A fault-tolerant continuous media disk array under arbitrary-rate search, IEEE Trans. Consumer Electron. 46 (2) (2000) 334–342.

[40] J. Oly, D.A. Reed, Markov model prediction of I/O requests for scientific applications, in: Proceedings of the 16th International Conference on Supercomputing, 2002, pp. 147–155.

[41] C. Park, Efficient placement of parity and data to tolerate two disk failures in disk array systems, IEEE Trans. Parallel Distrib. Systems 6 (11) (1995) 1177–1184.

[42] D.A. Patterson, Availability and maintainability: new focus for a new century, in: USENIX Conference on File and Storage Technologies (FAST '02), Keynote Address, Monterey, CA, 2002.

[43] D.A. Patterson, G. Gibson, R.H. Katz, A case for redundant arrays of inexpensive disks (RAID), in: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, 1988, pp. 109–116.

[44] M. Pillai, M. Lauria, A high performance redundancy scheme for cluster file systems, in: Proceedings of IEEE International Cluster Computing, 2003, pp. 216–223.

[45] J.S. Plank, A tutorial on Reed–Solomon coding for fault-tolerance in RAID-like systems, Software, Practice and Experience 27 (9) (1997) 995–1012.

[46] Prairiefire Cluster at UNL, <http://rcf.unl.edu>, September 2002.

[47] S. Quinlan, S. Dorward, Venti: a new approach to archival storage, in: Proceedings of the Conference on File and Storage Technologies, USENIX Association, 2002, pp. 89–101.

[48] M.O. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance, J. ACM 36 (2) (1989) 335–348.

[49] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, J. Kubiatowicz, Pond: the OceanStore prototype, in: Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), San Francisco, CA, 2003, pp. 1–14.

[50] R.B. Ross, Providing parallel I/O on Linux clusters, in: Second Annual Linux Storage Management Workshop, Miami, FL, 2000.

[51] R. Ross, D. Nurmi, A. Cheng, M. Zingale, A case study in application I/O on Linux clusters, in: Proceedings of SC2001, Denver, CO, 2001, pp. 1–17.

[52] F. Schmuck, R. Haskin, GPFS: a shared-disk file system for large computing clusters, in: Proceedings of the Conference on File and Storage Technologies (FAST '02), 2002, pp. 231–244.

[53] H. Singh, R.D. Gupta, On the probability that *k*th customer finds an M/M/1 queue empty, in: Advances in Applied Probability, vol. 24, 1992, pp. 238–239.

[54] N. Singpurwalla, The failure rate of software: does it exist?, IEEE Trans. Reliability 44 (3) (1995) 463–469.

[55] E. Smirni, R.A. Aydt, A.A. Chien, D.A. Reed, I/O requirements of scientific applications: an evolutionary view, in: Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, Syracuse, NY, 1996, pp. 49–59.

[56] H. Taki, G. Utard, MPI-IO on a parallel file system for cluster of workstations, in: Proceedings of the IEEE Computer Society International Workshop on Cluster Computing, Melbourne, Australia, 1999, pp. 150–157.

[57] Test TCP, <ftp://ftp.arl.mil/pub/ttcp/>, September 2002.

[58] J.D. Touch, Performance analysis of MD5, in: ACM Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, New York, NY, USA, 1995, pp. 77–86.

[59] N. Tran, D.A. Reed, Arima time series modeling and forecasting for adaptive I/O prefetching, in: Proceedings of the 15th International Conference on Supercomputing, 2001, pp. 473–485.

[60] F. Wang, Q. Xin, B. Hong, S.A. Brandt, E.L. Miller, D.D.E. Long, T.T. McLarty, File system workload analysis for large scale scientific computing applications, in: Proceedings of the Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies, IEEE Computer Society Press, College Park, MD, 2004.

[61] S. Welke, B. Johnson, J. Aylor, Reliability modeling of hardware/software systems, IEEE Trans. Reliability 44 (3) (1995) 413–418.

[62] J. Wilkes, R. Golding, C. Staelin, T. Sullivan, The HP AutoRAID hierarchical storage system, ACM Trans. Comput. Systems 14 (1) (1996) 108–136.

[63] Y. Zhu, H. Jiang, X. Qin, D. Feng, D. Swanson, Design, implementation, and performance evaluation of a cost-effective fault-tolerant parallel virtual file system, in: The International Workshop on Storage Network Architecture and Parallel I/Os, in conjunction with the IEEE Twelfth International Conference on Parallel Architectures and Compilation Techniques, New Orleans, LA, 2003.

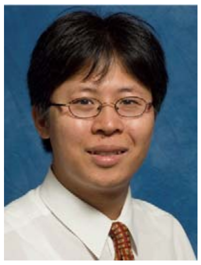
[64] Y. Zhu, H. Jiang, X. Qin, D. Feng, D. Swanson, Scheduling for improved write performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS), in: Proceedings of Cluster World Conference and Expo, San Jose, California, 2003.

[65] Y. Zhu, H. Jiang, X. Qin, D. Feng, D. Swanson, Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS), in: Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGRID), Workshop on Parallel I/O in Cluster Computing and Computational Grids, Tokyo, Japan, 2003, pp. 730–735.

[66] Y. Zhu, H. Jiang, X. Qin, D. Feng, D. Swanson, Exploiting redundancy to boost performance in a RAID-10 style cluster-based file system, Cluster Computing: The Journal of Networks, Software Tools and Applications, accepted for publication.

[67] Y. Zhu, H. Jiang, X. Qin, D. Feng, D. Swanson, A case study of parallel I/O for biological sequence analysis on Linux clusters, Internat. J. High Performance Comput. Networking 1 (4) (2004) 214–222.

[68] Y. Zhu, H. Jiang, X. Qin, D. Swanson, A case study of parallel I/O for biological sequence analysis on Linux clusters, in: Proceedings of IEEE International Conference on Cluster Computing, Hong Kong, 2003, pp. 730–735.



Yifeng Zhu received his B.Sc. degree in Electrical Engineering in 1998 from Huazhong University of Science and Technology, Wuhan, China; the M.S. and Ph.D. degree in Computer Science from University of Nebraska—Lincoln, in 2002 and 2005, respectively. He is an assistant professor in the Electrical and Computer Engineering Department at University of Maine. His main research interests are cluster computing, grid computing, computer architecture and systems, and parallel I/O storage systems. Dr. Zhu is a Member of ACM, IEEE, the IEEE Computer Society, and the Francis Crowe Society.

Hong Jiang received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, Wuhan, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Toronto, Canada; and the Ph.D. degree in Computer Science in 1991 from the Texas A&M University, College Station, Texas, USA. Since August 1991 he has been at the University of Nebraska—Lincoln, Lincoln, Nebraska, USA, where he is Professor and Vice Chair in the Department of Computer Science and Engineering. His present research interests are computer architecture, parallel/distributed computing, cluster and grid computing, computer storage systems and parallel I/O, performance evaluation, real-time systems, middleware, and distributed systems for distance education. He has over 100 publications in major journals and international conferences in these areas and his research has been supported by NSF, DOD and the State of Nebraska. Dr. Jiang is a Member of ACM, the IEEE Computer Society, and the ACM SIGARCH.