Design, Implementation and Performance Evaluation of A Cost-Effective, Fault-Tolerant Parallel Virtual File System

Yifeng Zhu*, Hong Jiang*, Xiao Qin*, Dan Feng†, David R. Swanson*

*Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE, Email: {yzhu, jiang, xqin, dswanson}@cse.unl.edu

†Department of Computer Science and Engineering
Huazhong University of Science and Technology, Wuhan, China, Email: dfeng@hust.edu.cn

Abstract—
Fault tolerance is one of the most important issues for parallel file systems. This paper presents the design, implementation and performance evaluation of a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS) that provides parallel I/O service without requiring any additional hardware by utilizing existing commodity disks on cluster nodes and incorporates fault tolerance in the form of disk mirroring. While mirroring is a straightforward idea, we have implemented this open source system and conducted extensive experiments to evaluate the feasibility, efficiency and scalability of this fault tolerant approach on one of the current largest clusters, where the issues of data consistency and recovery are also investigated. Four mirroring protocols are proposed, reflecting whether the fault-tolerant operations are client driven or server driven; synchronous or asynchronous. Their relative merits are assessed by comparing their write performances, measured in the real systems, and their reliability and availability measures, obtained through analytical modeling. The results indicate that, in cluster environments, mirroring can improve the reliability by a factor of over 40 (4000%) while sacrificing the peak write performance by 33-58% when both systems are of identical sizes (i.e., counting the 50% mirroring disks in the mirrored system). In addition, protocols with higher peak write performance are less reliable than those with lower peak write performance, with the latter achieving a higher reliability and availability at the expense of some write bandwidth. A hybrid protocol is proposed to optimize this tradeoff.

Keywords—**Fault-tolerance, reliability, availability, parallel I/O, Markov process**

I. Introduction

An attractive approach to alleviate the I/O bottleneck in clusters is to utilize the commodity disks that already exist as an integral part of each cluster node. Such disks in a large cluster collectively form a terabyte-scale capacity, which may satisfy the high I/O requirements of many data-intensive scientific applications when provided with efficient and reliable access. In fact, with the emergence of high-bandwidth and low-latency networks, such as the Myrinet and Gbps Ethernet, these independent storage devices can be connected together to potentially deliver high-performance and scalable storage.

Parallel file systems can in theory significantly improve the performance of I/O operations in clusters. One major concern of this approach 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 components of a cluster, such as network, memory, processors and software, are fault-free, the MTTF in a parallel file system with 128 server nodes will be reduced to around nine days if the failure of storage nodes is independent. Moreover, the MTTF will be further reduced when the failures of the other components are considered. Similar to disk arrays [1], without fault tolerance, these parallel file systems are too unreliable to be useful.

In this paper, we incorporate fault-tolerance into parallel file system by mirroring. More specifically, we present our design, implementation and performance evaluation of a RAID-10 style, cost-effective and fault-tolerant parallel virtual file system (CEFT-PVFS), an extension to the PVFS [2]. The analytical modelling results based on Markov process show that CEFT-PVFS can improve the reliability of the PVFS by a factor of over 40 times (4000%). While the mirroring scheme degrades the write performance by doubling the data flow, four mirroring protocols are designed with different write access schemes to achieve different tradeoffs between reliability gain and performance degradation. The write bandwidths of these four protocols are measured in a 128-node cluster while their reliability is evaluated by a new analytical model developed in this paper. Finally, a hybrid mirroring protocol is proposed to optimize the balance between the write performance (bandwidth) and the reliability.

The rest of this paper is organized as follows. We first discuss the related work in Section II. Then the design and implementation of our CEFT-PVFS are presented in detail in Section III. Section IV describes four different mirroring protocols and Section V shows the write performance of these protocols under a microbenchmark. In Section VI, a Markov-chain model is constructed to analyze the reliability and availability of these protocols. Finally, Section VII presents our conclusions and describes possible future work.

II. Related Work

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

Swift [3] and Zebra [4] employ RAID-4/5 to improve redundancy. Swift conducts file striping so that large files benefit from the parallelism. Zebra aggregates client's data first and then does striping on log-structured file systems to enhance small write performance. In both 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 [5] 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 [6], a block level distributed storage system, provides fault tolerance by using chained declustering [7]. 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 [8]. 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 43.1MB/s with 4 servers and 42 SCSI disks, which does not fully utilize the disk bandwidth.

GPFS [9] 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-PVFS requires no additional hardware in a cluster, GPFS typically needs fabric interconnections and RAID controllers.

PVFS [2][10] is an open source RAID-0 style parallel file system for clusters. It partitions a file into stripe units and distribute these stripes to disks in a round robin fashion. PVFS consists of one metadata server and serveral data servers. All data trafic 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.

III. Implementation Overview

A. The Choice of Fault Tolerance Designs

There are several approaches to provide 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 is to provide the redundancy by computing the parity in the same way as RAID-5. RAID-5 can tolerate any self-identifying device failure while self-identification may not be available in a typical cluster environment. In addition, a small RAID-5 write involves four I/Os, two to pre-read the old data and parity and two to write the new data and parity [11]. In a loosely coupled system, such as clusters, the four I/Os cause a large delay. 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) [12] and Reed Soloman Coding [13], 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 is extensively used in P2P systems [14], it not suitable for GB/s scale cluster file systems since the dispersal and reconstruction require matrix multiplications and multiple disk accesses and generate a potentially large computational and I/O overhead.

In CEFT-PVFS, we choose to use mirroring to improve the reliability. As the storage capacity increases exponentially, 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. Another benefit from mirroring, which the other redundancy approaches can not 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[15].

B. Design of CEFT-PVFS

CEFT-PVFS 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 Figure 1. There is one metadata server in each group. To make the synchronization simple, 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 be duplicated to the backup group. Four duplication (or mirroring) protocols are designed and will be discussed in Section IV.

C. Metadata Management

CEFT-PVFS 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 access 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 au-

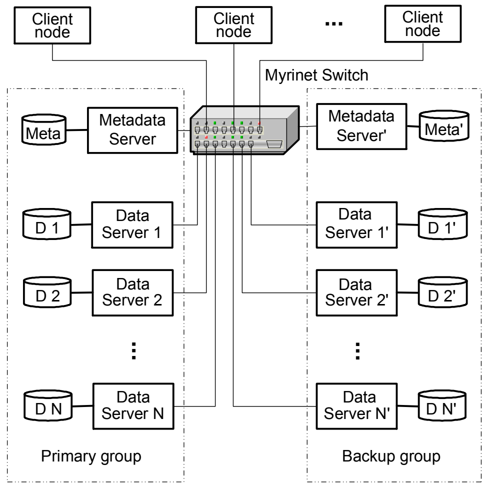

Fig. 1. Basic diagram of CEFT-PVFS.

TABLE I
Sample Metadata in CEFT-PVFS

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
⋮	⋮

thorization is implemented by checking the ownership and access mode. Table I gives a metadata example in CEFT-PVFS with 8 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 is striped. In this example, the file is striped across 3 data servers, i.e., Node 1, 2 and 7, with a striping block size of 64KB. 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 $1 \le i \le stripe\_width$.

D. 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 can not 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 is 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. This approach has two main disadvantages. One is that the total number of inodes on the metadata server is limited. When the file number is large, we may run out of inode numbers. The other, more significant disadvantage is that PVFS can not backup the meta server theoretically because 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-PVFS, we have changed the naming mechanism and instead used the MD5 sum [16] 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.

The calculation of MD5 will not introduce significant overhead in CEFT-PVFS. 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/sec 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.

E. Data Consistency

The I/O trace of scientific applications shows a frequent pattern that multiple clients concurrently access the same files [17]. In CEFT-PVFS, we employ an centralized byte-range locking mechanism to support the multiple-reader single-writer semantics. When the metadata server receives a write request, it looks at the desired portion of the destination file and checks whether this portion has already been locked by any other clients. If nobody locks this portion, the metadata will issue an write-lock to the client to permit the write access. Multiple read-locks can be issued to different read-only requests as long as no conflicting write-lock exists. Deadlock is avoided by using a two-phased locking, in which all locking operations in a transaction precede the first unlock operation [18]. To reduce the overhead of locking, 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 V, the metadata server is not likely the bottleneck under our measurements.

F. Data Recovery

After the reboot of a failed node, all the data on this node should be recovered. The recovery process in CEFT-PVFS is simple and fast since all the data after the checkpoint can be directly read from its mirrored server without doing any calculations. But consistency must be carefully enforced to eliminate any discrepancy between the primary and the backup caused by write requests from clients during the duplication process. A simple recovery method is to lock the primary server until the duplication has finished. However, this will make the I/O services unavailable for write requests (but still available for read requests) during the recovery. In the current implementation of CEFT-PVFS, the recovery process is designed by using "copy-on-write" on-line backup techniques [19]. The functional server will record the destination file names of every I/O write request that happens during the recovery period and put them in a waiting list. After finishing the duplication of the data after the checkpoint from the functional server to the rebooted server, the functional server will duplicate the files on the waiting list again to the rebooted server to eliminate the possible inconsistency caused during the recovery. As long as no files is in the waiting list, the recovery process completes. On the functional server, the recovery process holds a higher priority than I/O service process to guarantee that the recovery will eventually finish.

IV. Duplication Protocols

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

A. Protocol 1: Asynchronous Server Duplication

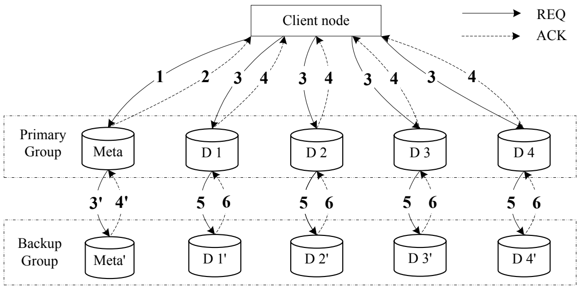

Fig. 2. The steps of duplication process for Protocol 1.

Figure 2 shows the steps of the duplication process. First, the client fetches the striping information from the metadata server (Step 1 & 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 (Step 5 & 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.

B. Protocol 2: Synchronous Server Duplication

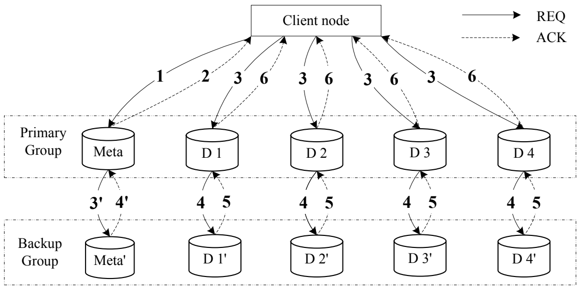

Fig. 3. The steps of duplication process for Protocol 2.

Protocol 2 is shown in Figure 3. 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 speedup the write performance, i.e., as soon as a block of striped data from any client arrives at the memory of the primary server, this 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 is 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 to be analyzed and discussed later in the paper.

C. Protocol 3: Asynchronous Client Duplication

In this protocol, the duplication task is assigned to the client, as shown in Figure 4. 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.

D. 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

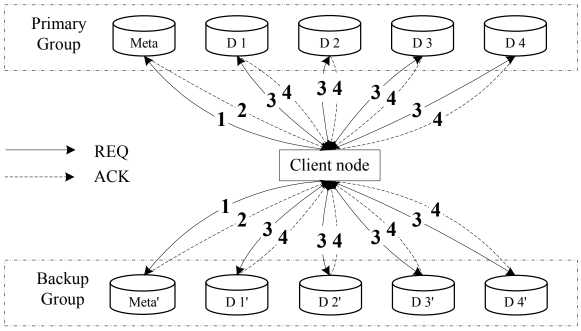

Fig. 4. The steps of duplication process for Protocols 3 and 4.

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.

E. 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 is 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 are 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.

V. Experimental Results on Write Performances

A. Experimental Environments

The performance results presented here are measured on the PrairieFire cluster [20] where CEFT-PVFS 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, 1GByte of RAM, a Myrinet card and a 20GB IDE(ATA100) hard drive. Under the same network and system environment as CEFT-PVFS, the *ttcp* [21] 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 2GBs of data, according to the Bonnie [22] benchmark.

B. Benchmark

A simple benchmark, similar to the one used in Ref. [2][5][23][24], was used to measure the overall concurrent write performance of this parallel file system. Figure 5 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.

```
for all clients:
    synchronize with all clients using MPI barrier;
    t1 = current time;
    open a file;
    synchronize with all clients using MPI barrier;
    t2 = current time;
    loop to write data;
    t3 = current time;
    close the file;
    t4 = current time;
    ct1 = t4 - t1; /* overall completion time */
    ct2 = t3 - t2; /* raw completion time */
    send ct1 and ct2 to client 0;

for client 0:
    /* find the slowest client */
    find  maximum of ct1 and ct2 respectively;
    calculate overall write throughput using maximum ct1;
    calculate raw write throughput using maximum ct2;
```

Fig. 5. Pseudocode of the benchmark

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 16MB to the servers, i.e., it writes 2MB, 1MB and 0.5MB 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 [25]. 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-PVFS, 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 5 highest and 5 smallest measurements.

C. The Metadata Server Overhead

The overall and raw write throughput is measured in CEFT-PVFS with a configuration of 8 data servers mirroring 8 under two access patterns: all clients concurrently write different files and all clients concurrently write different portions of the same file. Figure 6 and 7 plot the overall and raw write performance of Protocol 2 as a function of the number of client nodes, in which each measurement is repeated 20 times. 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. 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 [10][2] that the metadata server only introduces insignificant performance degradation and is not the performance bottleneck in a moderate-size cluster.

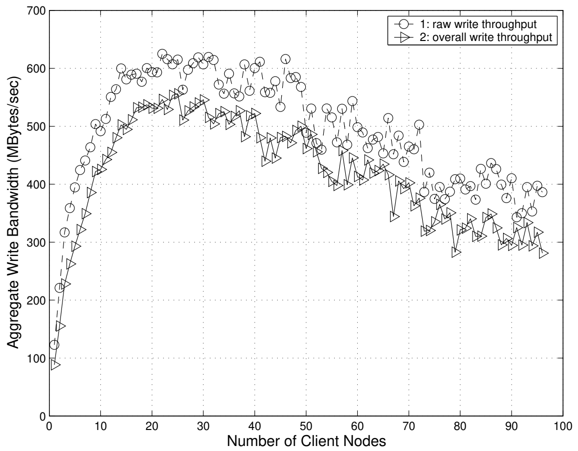

Fig. 6. Aggregate write performance when each client writes to a different file using Synchronous Server Duplication with 8-mirroring-8 data servers (20 measurements, discarding 5 highest and 5 smallest).

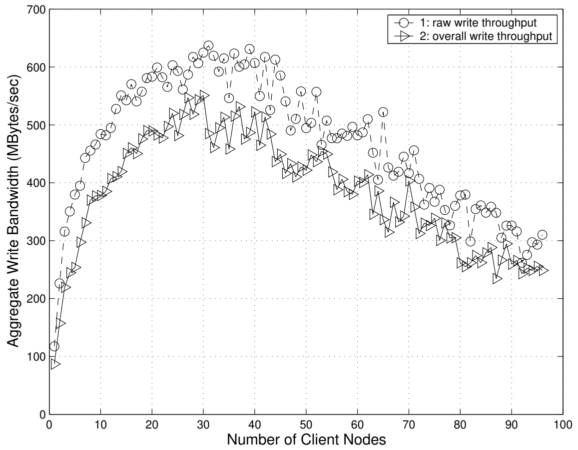

Fig. 7. Aggregate write performance when all clients write to the same file using Synchronous Server Duplication with 8-mirroring-8 data servers (20 measurements, discarding 5 highest and 5 smallest).

D. 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. Figures 8, 9 and 10 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, 2, and 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.

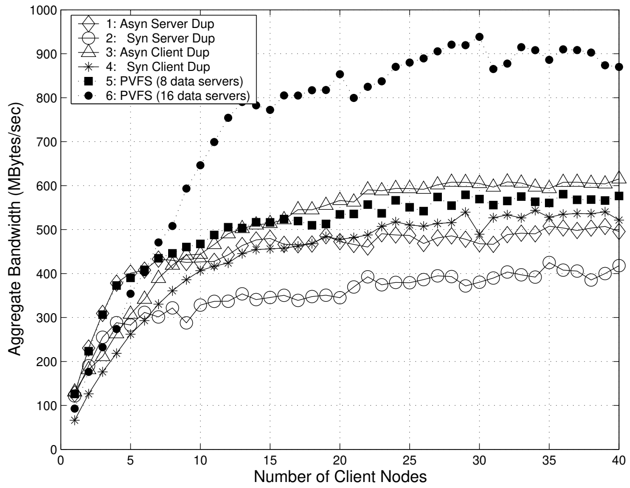

Fig. 8. Write performance when 8 I/O data servers mirror another 8 I/O data servers (70 measurements, discarding 5 highest and 5 smallest).

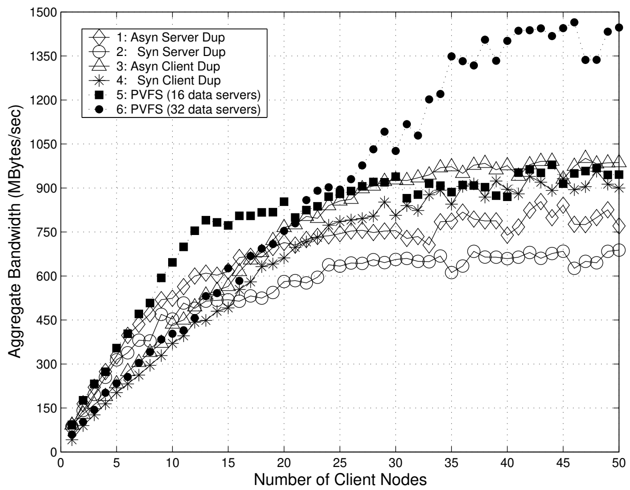

Fig. 9. Write performance when 16 I/O data servers mirror another 16 I/O data servers (70 measurements, discarding 5 highest and 5 smallest).

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

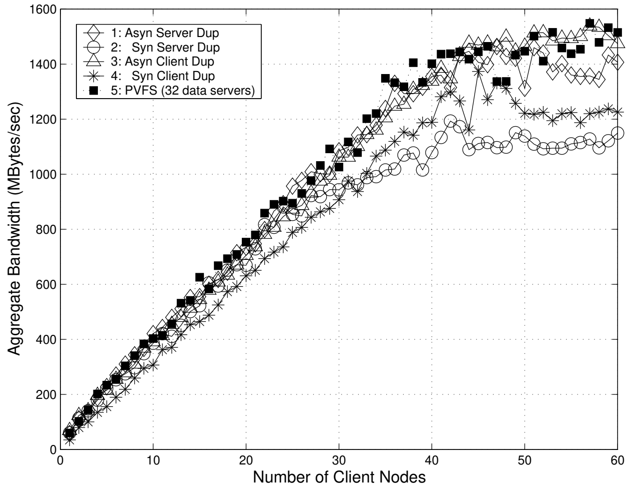

Fig. 10. Write performance when 32 I/O data servers mirror another 32 I/O data servers (70 measurements, discarding 5 highest and 5 smallest).

the performance is only slightly inferior to that of Protocol 1 when the primary server is lightly loaded (e.g., with fewer than 5 clients). As the workload on the primary server increases, the performance of Protocol 2 lags further behind that of Protocol 1.

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.

Table II 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 Protocol 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.

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 II, 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 surpass the PVFS with half the number of data servers.

VI. 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 Ref. [26] [27] [28] [29] [30]. 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 is 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 can not 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

TABLE II
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		

TABLE III
NOTATION

N	total number of nodes in one group
S	total number of Markovian states
i, j	index of Markovian states, $1 \leq i, j \leq S$
m, n	number of failed nodes, $0 \leq m, n \leq N$
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
$MTTF_{node} = \frac{1}{\lambda}$	mean time to failure per node
$MTTF_{switch} = \frac{1}{\lambda_s}$	mean time to failure per switch
$MTTW = \frac{1}{\lambda_w}$	mean time to write
$MTTR_{node} = \frac{1}{\mu}$	mean time to repair per node
$MTTD = \frac{1}{\mu_d}$	mean time to duplicate
$MTTDL$	mean time to data loss
M	Markovian fundamental matrix
$Q = [q_{ij}]_{S\times S}$	Markovian truncated matrix
P_c	probability that a primary node is consistent with its mirror node
$\hat{P}(mPnB)$	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

failure of nodes or disks that happens before the data in the cache is written onto the disks since its size is relative small. We understand that this assumption is somewhat unrealistic and may make us overvalue the reliability.
2. Network and node failures are all independent and follow an exponential distribution. 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 ($MTTW$).
4. The duplication time is 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.

Tab. III presents some basic notations, while others will be introduced appropriately during the discussion.

A. Calculation of 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 is consistent with the backup node. This duplication queue can be modeled by an $M/M/1$ queuing model [31][32]. In the model, the probability of the consistent state, i.e., the probability of an empty queue, can be calculated as follows:

$$\begin{aligned} P_c &= 1 - \frac{\lambda_w}{\mu_d} \\ &= 1 - \frac{MTTD}{MTTW} \end{aligned} \quad (1)$$

Although P_c is derived based on the duplication process of Protocol 1, this term can also be used in other protocols. In Protocol 2 and 4, all data has 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 the 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 is stored in the faster server and when data is stored in the slower server node.

B. Markov-Chain Model for Reliability Evaluation

Figure 11 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 group, 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 Eqn. 2, that maps from the system state with m failed primary nodes and n failed backup nodes to the state index i of the Markov state diagram:

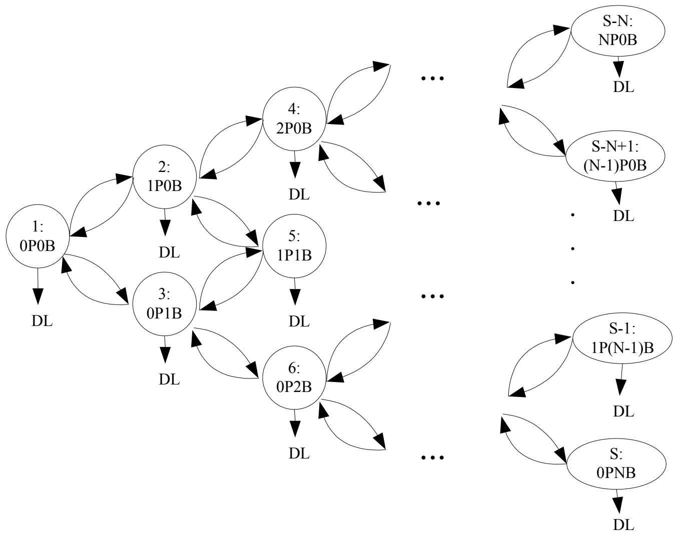

Fig. 11. Markov state diagram for Protocol 1.

$$i = \frac{1}{2}(m+n)(m+n+1) + (n+1) \tag{2}$$

Similarly, the inverse mapping function is given in Eqn. 3 and 4.

$$n = i - 1 - \frac{x(x+1)}{2} \tag{3}$$
$$m = x - n \tag{4}$$

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

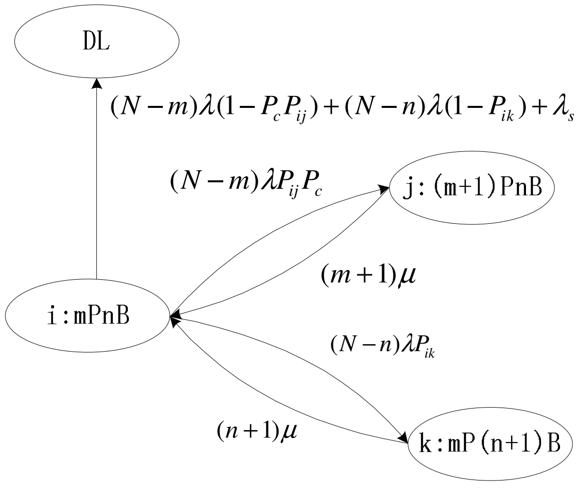

Fig. 12. The transition probability between different states.

Figure 12 shows the transition rate between the neighboring states. In the diagram, 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, 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. P_{ij} can be calculated as Eqn. 5.

$$\begin{aligned} P_{ij} &= \hat{P}((m+1)PnB \,|\, mPnB) \\ &= \frac{\hat{P}((m+1)PnB \cap mPnB)}{P(mPnB)} \\ &= \frac{\hat{P}((m+1)PnB)}{P(mPnB)} \end{aligned} \tag{5}$$

where $\hat{P}$ is the safety probability when m nodes in the primary group and n nodes in the backup group fail simultaneously. Eqn. 6 gives the calculation of $\hat{P}$.

$$\hat{P}(mPnB) = \begin{cases} \frac{\binom{N}{m+n}2^{m+n}}{\binom{2N}{m+n}} & \text{if } m+n \le N; \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Similarly, we have

$$P_{ik} = \frac{\hat{P}(mP(n+1)B)}{\hat{P}(mPnB)} \tag{7}$$

The transition probability from State i to the data loss state, denoted as $q_{i,DL}$, can be calculated as Eqn. 8.

$$\begin{aligned} q_{i,DL} &= \text{loss caused by one more primary node failure} \\ &\quad + \text{loss caused by one more backup node failure} \\ &\quad + \text{loss caused by network switch failure} \\ &= (N-m)\lambda[(1-P_{ij}) + P_{ij}(1-P_c)] \\ &\quad + (N-n)\lambda(1-P_{ik}) + \lambda_s \\ &= (N-m)\lambda(1-P_cP_{ij}) \\ &\quad + (N-n)\lambda(1-P_{ik}) + \lambda_s \end{aligned} \tag{8}$$

The stochastic transitional probability matrix is defined as $Q = [q_{ij}]$, where $1 \le i, j \le S$ and q_{ij} is the transition probability from State $i : m_iPn_iB$ to State $j : m_jPn_jB$. In summary, q_{ij} can be calculated as follows.

If $i < j$, then

$$q_{ij} = \begin{cases} (N-m_i)\lambda P_{ij}P_c & \text{if } m_j = m_i+1 \text{ and } n_j = n_i; \\ (N-m_i)\lambda P_{ij} & \text{if } m_j = m_i \text{ and } n_j = n_i+1; \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

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

$$q_{ii} = 1 - \sum_{i=1, j \ne i}^{j \le S} q_{ij} - q_{i,DL} \tag{11}$$

If $P_c = 1$, 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 Figure 11 can be simplified to the classic RAID-1 model [28], as shown in Figure 13. This is proven by the fact that numerical results generated by both models with the same set of input parameters are identical.

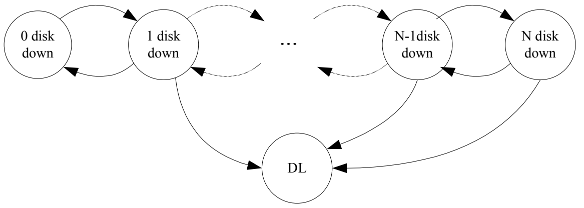

Fig. 13. Classic Markov state diagram of RAID-1.

C. Calculation of $MTTDL$

$MTTDL$ can be obtained from the fundamental matrix M, which is defined by the following equation [33].

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

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, $(1 \leq j \leq S)$, i.e.,

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

When $MTTD = 0$ and $MTTF_{switch} = \infty$, our model becomes the classic model for RAID-1. If $MTTD = \infty$ and $MTTF_{switch} = \infty$, 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. [28], our model shows identical results to those given in the above references.

To further validate our model, Figure 14 shows the relationship between $MTTDL$ and $MTTD$ under different workload conditions in an CEFT-PVFS where there are 8 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.

D. 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-PVFS depends on the write frequencies of the client nodes. The more frequently the client nodes write data into the storage nodes, the higher the 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.

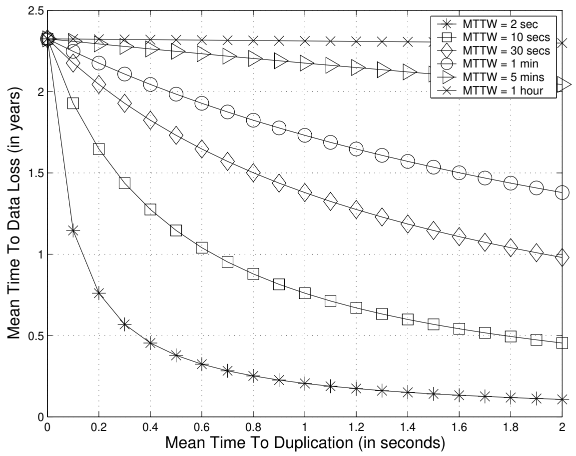

Fig. 14. Influence of $MTTD$ on $MTTDL$ of 8 mirroring 8 data servers under different workloads.($MTTF = 1$ year, $MTTF_{switch} =$ 3 years, $MTTR = 2$ days and $MTTW = 5$ minutes)

To quantitatively compare the reliability of the four duplication protocols, we evaluate their reliability in the scenario of a simple benchmark presented in Section V. Although this simple benchmark does not reflect all applications that run on CEFT-PVFS, 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 Protocol 2 and 4 can be regarded as 0 since the data is 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, $MTTF_{switch} =$ 3 years and $MTTR = 2$ days. In the simple benchmark, $MTTW = 1$ minute. We calculate the $MTTDL$ curve as a function of the number of server nodes for the four protocols under the three server configurations. Figure 15 compares the reliability between CEFT-PVFS and PVFS and Compared with their $MTTDL$, on the 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 Protocol 2 and 4 under the three different server configurations, respectively, with an average degradation of 5%. Protocol 3 is 96%, 94% and 99% of Protocol 2 and 4, with an average degradation of 3.3%.

E. Availability Analysis

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

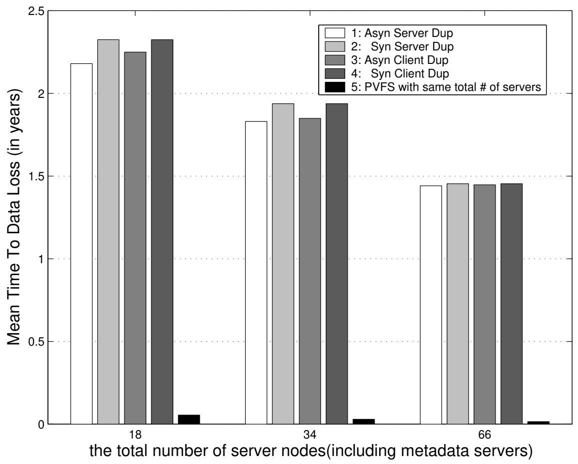

Fig. 15. Reliability comparison of CEFT-PVFS and PVFS

defined as follows.

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

Figure 16 and Figure 17 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-PVFS with four duplication protocols are all above 0.99. Similarly with the reliability comparisons, Protocol 2 and 4 achieve a better availability than Protocol 1 and 3.

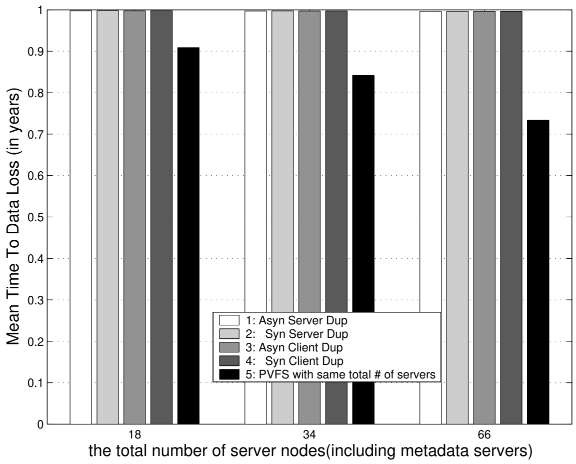

Fig. 16. Availability comparison of CEFT-PVFS and PVFS

F. 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

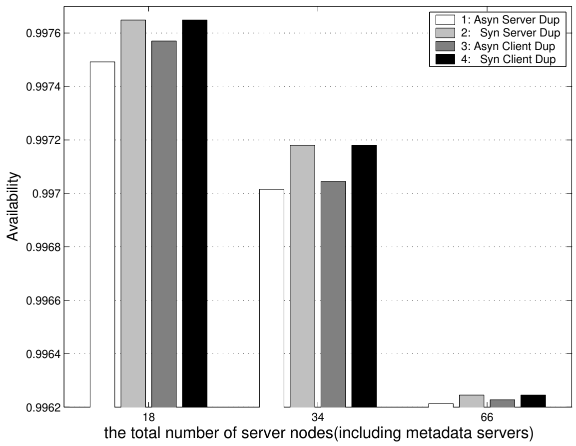

Fig. 17. Availability comparison of four duplication protocols

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 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.

VII. Conclusion and Future Work

In this paper, we demonstrate the feasibility and scalability of building a considerably reliable storage system with multi-terabyte capacity without any additional hardware cost in a cluster while maintaining high I/O performance to alleviate the I/O bottleneck for parallel applications.

Four different duplication protocols are designed to implement the fault tolerance in the parallel file system. Our experimental results and analytical analysis lead us to conclude that these protocols can improve the reliability over the original PVFS 40 times while degrading the peak write performance only around 33% in the best case, and around 58% in the worst case 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 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 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 thus can potentially lose data if a disk or node fails while data is being copied from the I/O buffer (cache) on the processor to the disk. We will further investigate the tradeoff when considering "forced writes".

VIII. Acknowledgments

This work was partially supported by an NSF Grant (EPS-0091900), a Nebraska University Foundation Grant (26-0511-0019) and University of Nebraska - Lincoln Academic Priority Grant. Work was completed using the Research Computing Facility at University of Nebraska - Lincoln.

References

[1] David A. Patterson, Garth Gibson, and Randy 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.

[2] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur, "PVFS: A parallel file system for linux clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, 2000, pp. 317–327, USENIX Association, Best Paper Award.

[3] Luis-Felipe Cabrera and Darrel D. E. Long, "Swift: using distributed disk stripping to provide high I/O data rates," *Computing Systems*, vol. 4, no. 4, 1991.

[4] J. H. Hartman and J. K. Ouserhout, "The Zebra striped network file system," *ACM Transactions on Computer Systems*, vol. 13, no. 3, pp. 274–310, Aug. 1995.

[5] Steven 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.

[6] Edward K. Lee and Chandramohan 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.

[7] Hui-I Hsiao and David DeWitt, "Chained Declustering: A new availability strategy for multiprocessor database machines," in *Proceedings of 6th International Data Engineering Conference*, 1990, pp. 456–465.

[8] J. Gafsi and E. W. Biersack, "Modeling and performance comparison of reliability strategies for distributed video servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 4, pp. 412–430, 2000.

[9] Frank Schmuck and Roger Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the Conference on File and Storage Technologies (FAST' 02)*, Jan. 2002, pp. 231–244.

[10] W. B. Ligon III and 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, Aug. 1996.

[11] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson, "RAID: high-performance, reliable secondary storage," *ACM Computing Surveys (CSUR)*, vol. 26, no. 2, pp. 145–185, 1994.

[12] Michael O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, Apr. 1989.

[13] James S. Plank, "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems," *Software, Practice and Experience*, vol. 27, no. 9, pp. 995–1012, 1997.

[14] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz, "Pond: the OceanStore Prototype," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, Mar. 2003.

[15] Yifeng Zhu, Hong Jiang, Xiao Qin, Dan Feng, and David Swanson, "Improved read performance in a Cost-Effective, Fault-Tolerant Parallel Virtual File System (CEFT-PVFS)," in *Proceeding of IEEE/ACM Workshop on Parallel I/O in Cluster Computing and Computational Grids, in conjunction with IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, Tokyo, Japan, May 2003.

[16] Joseph 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, Oct. 1995, pp. 77–86.

[17] David Kotz and Nils 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.

[18] K. P. Eswaran, J. N. Gray, R. A. Loric, and L. L. Traiger, "The notions of consistence and predicate locks in a database system," *Commun. ACM*, , no. 11, 1976.

[19] Ann Chervenak, Vivekenand Vellanki, and Zachary Kurmas, "Protecting file systems: A survey of backup techniques," in *Proceedings Joint NASA and IEEE Mass Storage Conference*, Mar. 1998.

[20] "Prairiefire Cluster at UNL," http://rcf.unl.edu, Sept. 2002.

[21] "Test TCP," ftp://ftp.arl.mil/pub/ttcp/, Sept. 2002.

[22] "Bonnie," http://www.textuality.com/bonnie/, Sept. 2002.

[23] Hakan Taki and Gil 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.

[24] Rosario Cristaldi, Giulio Iannello, and Francesco Delfino, "The cluster file system: Integration of high performance communication and I/O in clusters," in *Proceed of the 2nd IEEE/ACM international symposium on Cluster computing and the grid*, Berlin, Germany, May 2002.

[25] R. Ross, D. Nurmi, A. Cheng, and M. Zingale, "A case study in application I/O on linux clusters," in *Proceedings of SC2001*, Denver, CO, Nov. 2001, pp. 1–17.

[26] Garth A. Gibson, *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, Ph.D. thesis, U. C., Berkeley, Berkeley, CA, Mar. 1991.

[27] Yuyoung Oh, Sungsoo Kim, and Jai-Hoon Kim, "A fault-tolerant continuous media disk array under arbitrary-rate search," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 2, pp. 334–342, May 2000.

[28] Sung Hoon Baek, Bong Wan Kim, Eui Joung Joung, and Chong Won 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.

[29] Walter A. Burkhard and Jai Menon, "Disk array storage system reliability," in *Symposium on Fault-Tolerant Computing*, 1993, pp. 432–441.

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

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

[32] J. W. Cohen, *The single server queue*, North Holland, 1982.

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