Failure Recovery Issues in Large Scale, Heavily Utilized Disk Storage Systems

Garth Gibson, Paul Nowoczynski*, Milo Polte, Lin Xiao
*Pittsburgh Supercomputing Center, Carnegie Mellon University

1 Abstract

Large data is increasingly important to large-scale computation and data analysis. Storage systems with petabytes of disk capacity are not uncommon in high-performance computing and internet services today and are expected to grow at 40-100% per year. These sizes and rates of growth render traditional, single-failure-tolerant (RAID 5) hardware controllers increasingly inappropriate. Stronger protection codes and parallel reconstruction based on parity declustering are techniques being employed to cope with weakening data reliability in these large-scale storage systems. The first tolerates more concurrent failures without data loss at the cost of increasing redundancy overhead. The second parallelizes failure recovery from the traditional per-subsystem hardware RAID reconstruction to parallel and distributed reconstruction over all disks and RAID controllers. This paper explores the differences and similarities between large-scale storage systems in high-performance computing (HPC) and data-intensive scalable computing (DISC) for internet services, and revises reliability models for these storage systems to incorporate stronger redundant encoding and the use of parallel reconstruction. A modern example, for systems of 1-5 petabytes, suggests that triplication can have as much as 10 times lower rates of lost data per year, even when its number of components has to be almost 3 times more for the same amount of user data, but that this difference may be as small as 1 to 10 bytes lost per year. Many might decide that this factor of ten is not significant in light of other sources of information loss.

2 The Problem: Huge Collections of Disks

With petascale computers now in use there is a pressing need to anticipate and compensate for a probable increase in failure and application interruption rates and in degrading performance caused by online failure recovery. Researchers, designers and integrators have generally had too little detailed information available on the failures and interruptions that even smaller terascale computers experience. The information that is available suggests that failure recovery will become far more common in the coming decade, and that the condition of recovering online from a storage device failure may become so common as to change the way we design and measure system performance.

In our prior work in the DOE Petascale Data Storage Institute (www.pdsi-scidac.org), we collected and analyzed a number of large data sets on failures in high-performance computing (HPC) systems [Schroeder06]. The primary data set we obtained was collected during 1995–2005 at Los Alamos National Laboratory (LANL) and covers 22 high-performance computing systems, including a total of 4,750 machines and 24,101 processors. These data cover node outages in HPC clusters, as well as failures in storage systems. To the best of our knowledge, this is the largest failure data set studied in the literature to date, both in terms of the time-period it spans, and the number of systems and processors it covers, and the first to be publicly available to researchers (see [LANL06] for access to the raw data). Using these data sets and large scale trends and assumptions commonly applied to future computing systems design, we projected onto the potential machines of the next decade our expectations for failure rates, mean time to application interruption, and the consequential application utilization of the full machine, based on checkpoint/restart fault tolerance and the balanced system design method of matching storage bandwidth and memory size to aggregate computing power [Grider06]. Not surprisingly, if the growth in aggregate computing power continues to outstrip the growth in per-chip computing power, more and more of the computer's resources may be spent on conventional fault recovery methods. We envisioned highly parallel simulation applications being denied as much as half of the system's resources in five years, for example, and recommended new research on application fault tolerance schemes for these applications – process pairs [McEvoy81] mirroring of all computation is such a scheme that would halt the degradation in utilization at 50% [Schroeder07b].

The leading short-term alternative – adding a new tier of memory that is less expensive than DRAM, probably based on NAND flash, to buffer checkpoints allows the copying from checkpoint memory to disk to take longer because the average rate of writing checkpoints is much lower than the instantaneous rate that primary memory is dumped to checkpoint memory. This is a one-time improvement, as the disk system bandwidth still has to improve at the rate of the time average checkpoint capture, unless checkpoint memory can be large enough that no checkpoint is retained for long. Eliminating disk from the checkpoint solution is made harder by two issues: 1) memory, primary and checkpoint, are a large fraction of total system cost, and 2) simulation visualization and data analysis, today mostly piggy-backed on checkpoints, may independently mandate periodic simulation state capture to disk for later, offline processing. It is unlikely that large scale HPC computer design will soon abandon its expectation that disk storage bandwidth track overall computational performance.

Our interest in large-scale cluster node failure originated in the key role of high bandwidth storage in checkpoint/restart strategies for application fault tolerance [Elnozahy02]. Although storage failures are often masked from interrupting applications by RAID technology [Patterson88], reconstructing a failed disk can impact storage performance

noticeably [Holland94] and, if too many failures occur, storage system recovery tools can take days to bring a large file system back online, perhaps without all of its user's original data. Moreover, disks have traditionally been viewed as perhaps the least reliable hardware component, due to the mechanical aspects of a disk. We have been able to obtain datasets describing disk drive failures occurring at HPC sites and at a large internet service provider. The data sets vary in duration from 1 month to 5 years and cover a total of more than 100,000 hard drives from four different vendors, and include SCSI, fibre-channel and SATA disk drives. For more detailed results see [Schroeder07a].

For modern drives, the datasheet MTTFs are typically in the range of 1-1.5 million hours, suggesting an annual failure and replacement rate (ARR) between 0.58% and 0.88%. In the data, however, field experience with disk replacements differs from datasheet specifications of disk reliability. Figure 1 shows the annual failure rate suggested by the datasheets (horizontal solid and dashed line), the observed ARRs for each of the datasets and the weighted average ARR for all disks less than five years old (dotted line). Figure 1 shows a significant discrepancy between the observed ARR and the datasheet AFR for all datasets. While the datasheet AFRs are between 0.58% and 0.88, observed ARRs vary from 0.5% to as high as 13.5%. That is, the observed ARRs are by up to a factor of 15 higher than datasheet AFRs. The average ARR over all datasets (weighted by the number of drives in each data set) is 3.01%. Even after removing all COM3 data, which exhibits the highest ARRs, the average ARR was still 2.86%, 3.3 times higher than 0.88%, the higher of the often quoted AFR range.

With this cluster and disk failure data we developed various projections. Some of these projections are not relevant for this proposal, but the implications on disk drive failure recovery are central.

First, our projections expect integrators to deliver petascale computers according to the long-standing trends shown on top500.org [top500]; that is, the aggregate compute performance doubles every year. Second, our projections assume that integrators will continue to build balanced systems; that is, storage size and bandwidth will scale linearly with memory size and total compute power [Grider06]. As a baseline for our projections, we model the Jaguar system at Oak Ridge National Laboratory (ORNL) after it is expanded to a Petaflop system in 2008. Jaguar then had around 11,000 processor sockets (dual-core Opterons), 45 TB of main memory and a storage bandwidth of 55 GB/s [Roth06]. While the architecture of other 2008 petascale machines, such as LANL's Roadrunner [Koch06], differs from Jaguar in its use of hybrid nodes employing vector/graphics co-processors, our predictions for its failure rates are little different from Jaguar, so we do not include separate lines for it on our graphs.

First, individual disk bandwidth grows at a rate of about 20% per year, which is significantly slower than the 100% per year growth rate that top500.org predicts for total processing power. To keep up, the number of disk drives in a system will have to increase at an impressive rate. Figure 2 projects the number of drives in a system necessary to (just) maintain balance. The figure shows that, if current technology trends continue, by 2018 a computing system at the top of top500.org chart will need to have more than 800,000 disk drives. Managing this number of independent disk drives, much less delivering all of their bandwidth to an application, will be extremely challenging for storage system designers.

Second, disk drive capacity will keep growing by about 50% per year, thereby continuously increasing the amount of work needed to reconstruct a failed drive, and the time needed to complete this reconstruction when a single RAID controller does all of the work of each failed disk reconstruction. While other trends, such as decrease in physical size (diameter) of drives, will help to limit the increase in reconstruction time, these are single step decreases limited by the poorer cost effectiveness of the smaller disks. Overall we think it is realistic to expect an increase in reconstruction time of at least 10% per year using the current hardware RAID controller model. Assuming that reconstruction time was about 30 hours and that 3% of drives in a system fail per year on average (as shown in Figure 1), we projected the number of concurrent reconstructions going on in future HPC systems, as shown in Figure 3. The figure indicates that in the year 2018, on average, nearly 300 concurrent reconstructions will be in progress at any time.

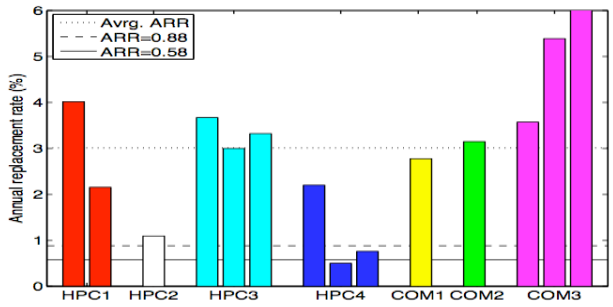


Figure 1: Comparison of data sheet annual failure rates (horizontal dotted lines) and the observed annual replacement rates of disks in the field.

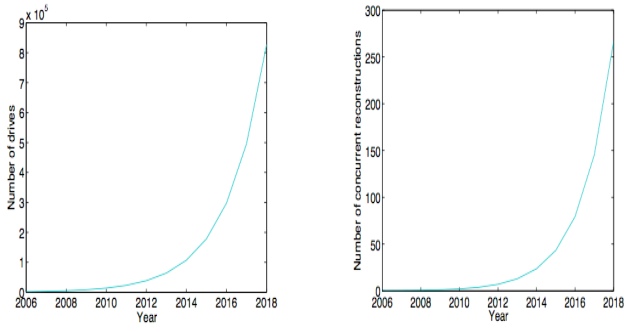


Figure 2 (left) and 3 (right). (2) Number of disk drives in the largest of future systems. (3) Number of concurrent reconstructions in the largest of future systems.

The operating model for HPC storage calls for it to fully mask internal errors – HPC fault tolerance requires storage to survive all failures, and to absorb and return checkpoints at full speed at all times. While in practice this does not require enterprise-class availability, five minutes of downtime per year or less, it does means that hundreds of concurrent recoveries at all times must be managed with essentially no degradation in availability and performance. Yet each recovery typically processes tens of terabytes, and the minimum time needed to simply read (or write) an entire disk is typically multiple hours, given 100% of the disk's resources. Even a single RAID controller dramatically slowed down by a reconstruction is likely to slow down the entire storage system by the same dramatic reduction because highly parallel, highly striped data access uses all storage devices evenly.

It seems quite clear that petascale storage systems designers will be spending a large fraction of their efforts on fault tolerance inside the storage systems on which petascale application fault tolerance depends.

3 Parallel Reconstruction

There are two principle techniques for addressing pressing concerns of ever larger numbers of disks in the storage system: (1) redundant data encodings tolerant of more failures [Plank09], and (2) acceleration of the failed disk reconstruction process. The former is taking place, as many large storage installations are using double-failure-tolerant RAID 6 and some vendors have begun to recognize that reliability is improved when more sets of N concurrent failures are recoverable even if every set of N concurrent failures are not recoverable [Hafner05].

Greater levels of concurrent failure tolerance by itself do nothing to address the duration of a reconstruction, and therefore the time-averaged number of concurrent reconstructions. Instead, one might reasonably expect that higher levels of failure tolerance will require more complex codes, slowing down reconstruction. Moreover, each RAID subsystem, typically containing one or more sets of 8 to 16 disks in a single RAID group, independently detects and recovers from disk failures, so failure recovery and performance degradation during failure recovery is not load balanced.

Parity declustering is a technique for distributing RAID groups evenly over all RAID controllers [Holland94]. Figure 4 shows an example parity declustering organization. Provided that replacement disk space and RAID controller functionality are also distributed, disk failure reconstruction is parallelized and reconstruction can be done much faster in large systems and evenly load balanced.

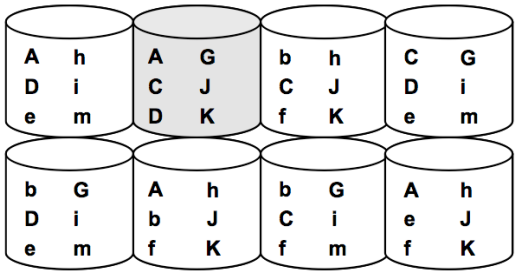


Figure 4. An example of parity declustering. Parity groups of four components, each designated by a single letter label, are spread over eight disks evenly.

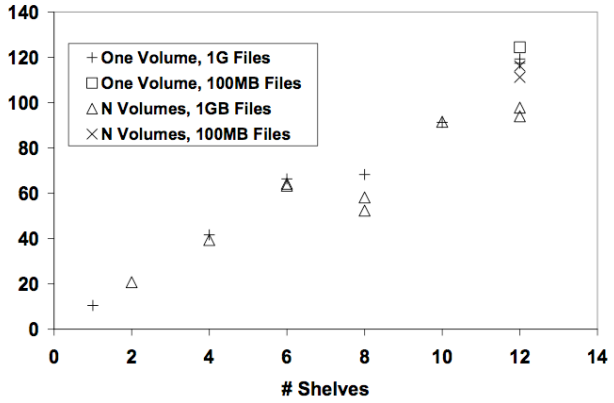


Figure 5: The speed that lost data can be reconstructed, in megabytes per second, as a function of the number of shelves (20 disks and a controller in each) in a PanFS storage system employing parity declustering and parallel reconstruction.

Parallel reconstruction, enabled by parity declustering, provides a powerful tool for coping with the predicted hundreds of concurrent reconstructions in large systems. Parallel reconstruction causes reconstruction speed to scale up in proportion to the size of the storage system. Implemented in the Panasas PanFS storage system, Figure 5 shows how reconstruction speed increases linearly with large storage systems. Because parity groups are evenly distributed over all disks, each block lost on a failed disk is equally likely to be associated with every other disk by parity group, and provided many parity groups are reconstructed in parallel, all disks contribute to the reconstruction effort equally.

With parallel reconstruction, reconstructions can be done faster in large clusters, yielding much less overlapping of reconstructions. Even more powerfully, this technique scales with system size, so its utility becomes more compelling as systems get larger.

However, data reliability is impacted by parity declustering in two countering ways. While shorter reconstructions reduces the window of time that future disk failures may render current reconstructions unable to recover data, distributing parity groups exposes potential data loss to more combinations of disk failures. That is, if the parity groups of a double-failure-tolerant storage system are declustered, then each time three disks are concurrently failed, there is a good chance that some parity group included all three of these disks, while in a traditional multiple RAID 6 storage system, if at least two of the failed disks are in different RAID 6 arrays, there is no chance that data is not protected by RAID.

Early work on parity declustering found that these two factors – smaller time windows of vulnerability to additional failures and larger sets of disks entwined in the same parity

groups – cancelled each other, causing data reliability to be neither hurt nor improved by parity declustering [Holland94].

Parity declustering and parallel reconstruction, coupled with more powerful redundant encodings, are the principle tools for making ever-larger storage systems reliable and performant.

4 Data Intensive Storage Systems

While parallel file systems used in HPC clusters and supercomputers was the starting point for this research, we are concerned with all large-scale storage systems. In particular, Data-Intensive Scalable Computing (DISC) systems such as used by internet services clusters are of comparable scale and performance to HPC systems. Figure 6 compares the system architecture of most HPC systems (a) to DISC systems (b).

Most DISC applications are characterized by parallel processing of massive datasets stored in the underlying shared storage system. Such distributed programming abstractions are provided by purpose-built frameworks like MapReduce [Dean04], Hadoop [Hadoop] and Dryad [Isard07]. These frameworks divide a large computation into many tasks that are assigned to run on nodes that store the desired input data, and avoiding a potential bottleneck resulting from shipping around terabytes of input data. Hadoop, and its data intensive file system HDFS [Borthakur09a], are open-source implementations of Google's MapReduce and GoogleFS [Ghemawat03]. Our exploration of DISC systems is based on Hadoop's use of the HDFS data intensive file system.

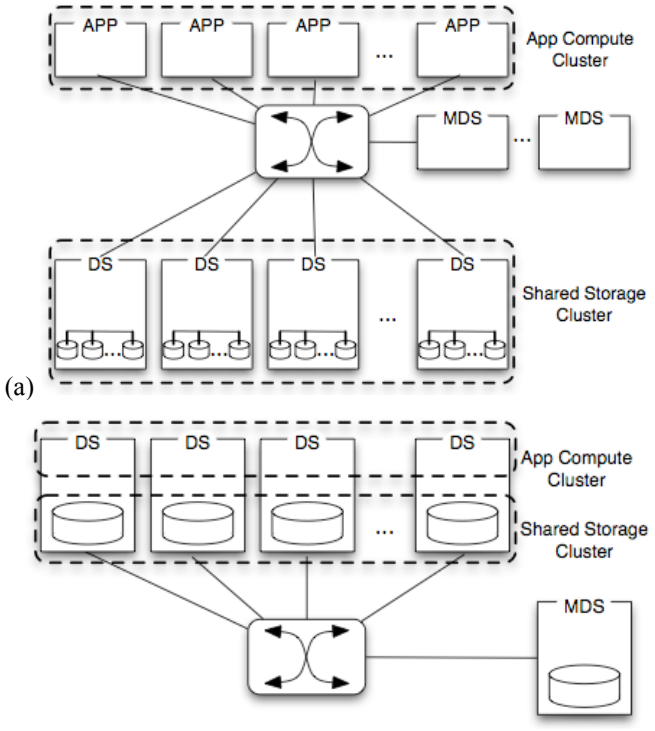


Figure 6: HPC and DISC storage models. Most HPC storage (a) separates data servers (DS) from compute servers in dedicated subsystems, each internally protected by RAID controllers. DISC systems (b), however, often fold storage back into the application servers. Both systems separate metadata servers (MDS).

4.1 Parallel versus Data Intensive File Systems

At a high level HDFS's architecture resembles an HPC parallel file system. HDFS stores file data and metadata on two different types of servers. All files are divided into chunks that are stored on different data servers. The file system metadata, including the per-file chunk layout, is stored on the metadata server(s). For single writer workloads, HDFS differs from HPC parallel file systems primarily in its layout and fault tolerance schemes.

HDFS assigns chunks to compute nodes at random, while HPC file systems use a round robin layout over dedicated storage servers, and HDFS exposes a file's layout information to Hadoop. This exposed layout allows the Hadoop's job scheduler to allocate tasks to nodes in a manner that (1) co-locates compute with data where possible, and (2) load balances the work of accessing and processing data across all the nodes. Thus, the scheduler can mask sub-optimal file layout resulting from HDFS's random chunk placement policy with lots of work at each node [Tantisiriroj08]. The second big difference between HDFS and HPC file systems is its fault tolerance scheme: it uses triplication instead of RAID. We address this difference in the next section.

Given the growing importance of the Hadoop MapReduce compute model, we ask, "Could we use a mature HPC parallel file system in-place of a custom-built DISC file system like HDFS?" While most HPC file systems use separate compute and storage systems for flexibility and manageability, most HPC parallel file systems can also be run with data servers on each compute node.

We built a non-intrusive shim layer to plug a real-world parallel file system (the Parallel Virtual File System, PVFS [PVFS2]), into the Hadoop framework storing data on compute nodes [Tantisiriroj08]. This shim layer queries file layout information from the underlying parallel file system and exposes it to the Hadoop layer. The shim also emulates HDFS-style triplication by writing, on behalf of the client, to three data servers with every application write.

	HDFS	vanilla PVFS	PVFS shim
`grep` performance (over a 64GB data-set on 32 nodes)			
Read throughput (MB/s)	579.4	244.9	597.1
Avg CPU utilization	43%	27%	43%
Completion time (m:s)	1:45	4:08	1:46

Figure 7: By exposing the file layout mapping through a non-intrusive shim layer, a production parallel file system (PVFS) can match the performance of HDFS on a widely used Hadoop-style workload.

Figure 8 shows that for a typical Hadoop application (grep running on 32 nodes), the performance of shim-enabled Hadoop-on-PVFS is comparable to that of Hadoop-on-HDFS. By simply exposing a file's layout information, PVFS enables the Hadoop application to run twice as fast as it would without exposing the file's layout.

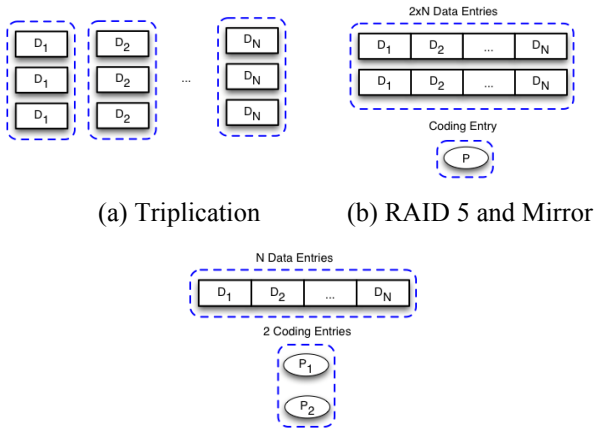
Most parallel large-scale file systems, like PVFS, already expose the file layout information to client modules but do not make it available to client applications. For example, the new version 4.1 of NFS (pNFS) delegates file layout to client modules to allow the client OS to make direct access to striped files [NFSv4.1]. If these layout delegations were exposed to client applications to use in work scheduling decisions, as done in Hadoop or MapReduce, HPC and pNFS file systems could be significantly more effective in DISC system usage.

4.2 Replication versus RAID in DISC Systems

To tolerate frequent failures, each data block in a data intensive file system is typically triplicated and therefore capable of recovering from two simultaneous node failures. Though simple, a triplication policy comes with a high overhead cost in terms of disk space: 200%. Traditional RAID systems typically exhibit capacity overheads between 10% and 25% -- about 10 times smaller! We have built *DiskReduce*, an experimental application of RAID in HDFS to save storage capacity.

In HDFS, files are divided into blocks, typically 64 MB, each stored on a data node. Each data node manages all file data stored on its persistent storage. It handles read and write requests from clients and performs "make replica" requests from the metadata node. There is a background process in HDFS that periodically checks a missing blocks and, if found, assigns a data node to replicate the block having too few copies.

DiskReduce exploits HDFS's background re-replication to replace copies with lower overhead RAID encoding. In a manner reminiscent of early compressing file systems [Cate91], all blocks are initially triplicated; that is, uncompressed. Where the background process looks for insufficient number of copies in HDFS, DiskReduce instead looks for blocks not encoded, and replaces copies with encodings. Because it is inherently asynchronous, DiskReduce can further delay encoding when space allows, to allow accesses temporally local to the creation of the data choice among multiple copies for readback.



(a) Triplication (b) RAID 5 and Mirror

(c) RAID 6

Figure 8: Codewords protecting user data against any double node failure in DISC storage systems using DiskReduce.

We have prototyped DiskReduce as a modification to Hadoop Distributed File System (HDFS) version 0.20.0. Currently, the DiskReduce prototype supports only two encoding schemes [Plank08]: "RAID 6" and "RAID 5 and Mirror", in which a RAID5 encoding is augmented with a second complete copy of the data.

Our prototype runs in a cluster of 63 nodes, each containing two quad-core 2.83GHz Xeon processor, 16 GB of memory, and four 7200 rpm SATA 1 TB Seagate Barracuda ES.2 disks with 32MB buffer. Nodes are interconnected by 10 Gigabit Ethernet. All nodes run the Linux 2.6.28.10 kernel and use the ext3 file system for storing HDFS blocks.

To get a feel for its basic encoding functionality in our prototype, we set up a 32 node partition and had each node write a 16 GB file into a DiskReduce-modified HDFS spread over the same 32 nodes using RAID groups of eight data blocks each.

Figure 9 shows the storage usage and encoding bandwidth consumed for the encoding of this 512GB of data. While this experiment is simple, it shows the encoding process removing 400GB and 900GB for the RAID 5 and mirror and RAID 6 schemes, respectively, bringing overhead down from 200% to 113% and 25%, respectively.

Based on a talk about our previous DiskReduce work, a userspace RAID 5 and mirror encoding scheme has been implemented on top of HDFS by HDFS developers [Borthakur09b]. We are working closely with Hadoop and HDFS developers to further explore RAID 6 encodings, delaying of encoding to enable reading soon after write the full benefit of multiple copies, and to quantify this benefit, and delaying of deletion to trade capacity against the encoding cleanup of a partial RAID set delete.

Similarly HPC parallel file systems have begun to imple-

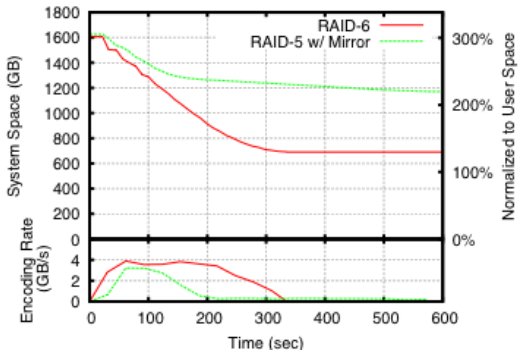


Figure 9: Storage capacity utilized and rate capacity is recovered by Disk Reduce background encoding.

ment RAID over nodes, instead of just RAID in hardware, led by a spin off of our prior work in scalable file systems [Welch08]. In our long-term vision for data intensive storage systems we see a convergence of the semantic power of HPC parallel file systems with the high degrees of node failure tolerance in data intensive file systems.

5 Reliability Modeling

The reliability models built for RAID are based on non-declustered storage. The original model [Patterson88] changed little when RAID 6 was commercialized [Corbett04], although a higher rate of secondary failures was added as a "correlation" factor on the disk failure rate. Moreover, both the original and revised models retain the simple notion that reliability should be measured as the time until any data is lost, because loss of any data is catastrophic, and differentiation of the amount lost is not pertinent. Mean time until disk failure, the inverse of annual failure rate, is measured in decades to centuries, and mean time until data loss in RAIDs is orders of magnitude larger. Yet there is no direct utility to a model of centuries until data loss because computing devices are rarely operated more than a decade, and few have warrantees for more than five years.

The most useful interpretation of mean time until data loss assumes data loss events have a Poisson distribution, so the probability of data loss in a short time period (days or weeks in practice) is proportional to the reciprocal of mean time until data loss. In this sense doubling the mean time until data loss halves the probability of data loss in a short time period.

Yet even these simplistic models are rarely used in practice. The most practiced model for reliability is the maximum number of concurrent failures of any disks that is always recoverable based on the redundant encoding. In this sense the metric in practice is: any single disk failure loses data, all single disk failures are tolerated, all double disk failures are tolerated, and, in the future, all triple disk failures are tolerated. With this weak model, the 200% overhead (three copies) triplicated storage of DISC systems has equivalent failure protection to the 25% overhead (with groups of eight disks) RAID 6 storage in HPC systems. Of course, no one expects this to be true, and the traditional models for RAID reinforce this expectation [Patterson88, Corbett04].

In this work we improve traditional models for redundant disk storage reliability in two ways: the inclusion in the model of parity declustering and expanded metrics for the amount of data lost annually rather than simply the expected number of loss events.

5.1 Large Scale Storage Failure Model

In this work we focus on catastrophic failure of magnetic disks. This is not the only failure such disks experience. For example, unrecoverable read errors, or latent sector errors are non-catastrophic failures in which a small amount of data on the disk is not readable, although rewriting the storage region may be subsequently readable [Bairavasundaram07]. Although such failures are sometimes repaired by RAID reconstruction, they are also minimized by scrubbing for them in the background, by selecting disk products with stronger per-sector codes and by multiple sector protection codes inside each disk [Dholakia06].

When RAID reconstruction is used to recover from latent sector errors there is one important special case: each latent sector error experienced during the reconstruction of the maximum tolerable disk failures causes a parity stripe to be unrecoverable. In traditional RAID systems the loss of a single parity stripe causes the enclosing volume to be unavailable until manual repair can be invoked. This failure mode is often quoted as the driving reason for the introduction into enterprise storage systems of double failure tolerating codes, RAID 6 [Corbett04].

Alternatively, many of the declustered parity and replicated storage solutions can suffer the loss of a single file without taking the enclosing volume being taken offline. In these systems latent sector errors are another source of small periodic data loses. In the following model, these sources of data loss have not yet been incorporated. The model is also not designed to model failover of an entire data center to a backup data center, as is done in business continuity solutions and alternative internet service centers. Our models are intended to be used to contribute to decisions about when to invoke these much more expensive replication solutions.

Disk failures, disk repairs, data loss events are often simplistically modeled as Poisson events, with exponential interarrival times [Patterson88, Corbett04]. Although it is demonstrable that these events fail statistical tests for having a Poisson distribution [Schroeder07a, Pinheiro07], we seek generalized models of fundamental differences, and do not want to overwhelm the interpretation of these fundamental differences with a level of detailed modeling that is both product specific and likely to impart a contribution smaller than the inherent errors introduced by unmodeled factors such as human error, for example. We follow the example of a higher rate of Poisson disk failures after a first failure in a run used in the models given when RAID 6 products were introduced [Corbett04].

We employ a renewal reward stochastic Markov model to compare triplication and RAID 6 reliability in large storage systems.

The model parameters (rates are per hour) are:

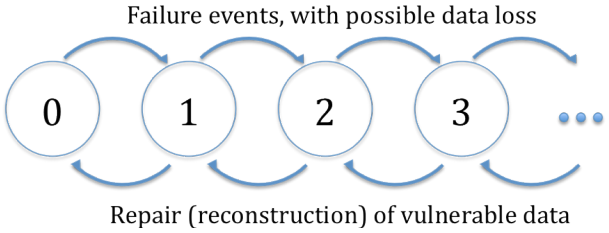


Figure 10: Basic model of disk failures and repair completions, modeled as Poisson processes. On each failure there is a probability of data loss, which accumulates into the overall expectation of the rate of data loss.

N = total number of disks
d = total number of data blocks stored in each disk
G = number of unreplicated data blocks in a group (not including blocks with computed "check" codes)
D = total number of non-redundant user data blocks
μ = (Poisson) repair rate during a single failure repair
λ = (Poisson) failure rate for a single disk
c = correlation of disk failures; after the 1st in a run of failures, each disk experiences a failure rate is cλ.

In the zeroeth state the total rate of new disk failures is Nλ and there is no chance of data loss. In all other states each disk failure rate is elevated by correlated failure events. Since we are modeling very large systems the failure rates in non-zero states is approximately cNλ.

Repair, the process of reconstructing the data lost when a disk fails, starts with a detection and diagnosis phase. Quite often storage nodes can be non-responsive for non-trivial amounts of time; for example, rebooting a node can easily be a multiple minute operation. Systems overly eager to reconstruct an unavailable disk can induce large amounts of unnecessary work, since unavailable nodes often return to service. For this reason, practical repair rates need to take into consideration a lower bound on repair time based such detection and diagnosis phases. Anecdotal evidence suggests this lower bound is multiple minutes, and perhaps as large as a large fraction of an hour.

Once reconstruction is started, it could proceed as fast as the time it takes the entire system to read the amount of data equivalent to the size of G disks, dG blocks, spread evenly over all disks (G is 1 for replication schemes). Many installations will want this to be throttled to consume a small fraction of the bandwidth of each disk, so that user workload (such as taking checkpoints) will not be significantly degraded. Assuming that this throttling limits the total bandwidth used for reconstruction, the repair rate, μ, will be the same from all states. However, with triplication only one block has to be read to reconstruct each failed block but with RAID 6 each failed block is reconstructed as a function of G surviving blocks, so RAID 6 has G times as much reading work to do, and a correspondingly slower repair rate (when throttled to a bandwidth limit).

Triplication and RAID 6, our canonical comparison, yield different amounts of user data from the same number of disks, because of their different encodings. Triplication schemes provide D = dN/3 user data blocks, since two blocks of every triple contain identical copies and cannot freely vary; while RAID 6 schemes provide D = dNG/(G+2) user data blocks, since 2 of every G+2 blocks in a group are do not contain freely variable user data.

The expected amount of data lost in each failure event accumulates to the expected data loss rate for the model. With double failure tolerant codes like triplication and RAID 6, no data loss is possible in states 0 and 1, but once in states 2 or larger, each new disk failure may lead to an unreconstructable block because that group of blocks may suddenly have three lost blocks, while the code can only reconstruct the lost data of up to two lost blocks.

With a triplication encoding, in state *i* (that is, there are *i* concurrent disks being reconstructed), when another disk fails each of its d blocks is independently put at risk based on the location of its two replicas. There are (N-1) choose 2 combinations of these two replicas, all equally likely, and *i* choose 2 of these combinations will yield data loss. So for triplication, the model "reward", the expected number of blocks lost, is (Equation 1):

$$E[data\ loss|(i+1)^{th}\ failure] \quad = \quad d\frac{\binom{i}{2}}{\binom{N-1}{2}}$$

For the RAID 6 encoding the calculation is more complex. For each block on the newly failed disk, we condition on the state of the other G+1 blocks in its group. If zero or only one of these blocks is also failed, no data is lost.

If the newly failed block is the third failed block in the group, then these three blocks are newly unreconstructable, and with probability G/(G+2) each of these contains user data. Given a specific newly failed block, there are N-1 choose (G+2-1) other blocks in the group. Of these we want two other failed and G-1 not failed, so we want the product of *i* choose 2 ways to pick the failed blocks and N-*i*-1 choose G-1 ways to select the non failed blocks from the possible non-failed disks.

For each case were the newly failed block is part of a group that has already lost data, that is, suffered three loses already, the amount additionally lost by this newly failed block is 1, prorated by the odds this is a data block, G/(G+2). These cases correspond to j = 3 through *i* pre-existing failed blocks with probabilities *i* choose *j* times N-*i*-1 choose G+2-*j* over N-1 choose G+1, using the same counting argument as above. Combining these counting arguments, for RAID 6, the expected number of blocks is (Equation 2):

$$E[data\ loss|(i+1)^{th}\ failure]$$

$$= \quad \frac{dG}{G+2}\left(1 - \frac{\binom{i}{0}\binom{N-i-1}{G+1} + \binom{i}{1}\binom{N-i-1}{G} - 2\binom{i}{2}\binom{N-i-1}{G-1}}{\binom{N-1}{G+1}}\right)$$

5.2 Numeric Examples

Our model was coded into Mathematica and run through a few interesting cases.

Consider storage systems of one to five petabytes of user data (D) built from 2 terabyte disks, capable of sustaining 50 MB/second each, where each disk and the entire system 80% utilized. For illumination of reliability in DISC storage systems in particular, as with DiskReduce we have software for triplication and RAID 6, we use disk blocks of 64 megabytes, typical in DISC storage such as HDFS. Parameter d is determined as 2 TB / 64 MB * 80%.

Disk failure rates are set by our prior work to 2% failing per year and repair rates are set to 10% of a peak rate determined by the amount to be read divided by the number of disks, each providing 50 MB/second. We have not yet explored failure correlations other than c=1.

RAID 6 groups contain eight data blocks and two check blocks.

Figure 11 shows this comparison when it takes about an hour for the storage system to decide to initiate reconstruction. The primary implication in Figure 11 is the relatively low rate of data loss in all cases: less than 10 bytes per year. This is easily tolerable, and suggests that real world causes of data loss are probably beyond the scope of this model, or that the reliability difference between RAID 6 and triplication is unlikely to matter in practice.

The secondary implication in Figure 11 is the different implications of increasing total data size. Because the amount of user data is held constant, there are almost three times more disks in a system using triplication than one using RAID 6. This means that overall disk failure rates grow three times faster in triplication, yet repair rates in both are linear in the number of disks, and as the systems get larger repair is increasingly dominated by the detection and diagnosis phase of repair.

It is probably not a surprise to see that RAID 6 annual data loss rates in small systems are much larger than in systems using triplication, perhaps because of the benefit as the system gets larger of fewer and fewer data loss cases as percentage of total cases is larger with the initially higher loss rates of RAID 6.

6 Related Work

Almost all enterprise and high performance computing storage systems protect data against disk failures using a variant of the erasure protecting scheme known as RAID [Patterson88]. Presented originally as a single disk failure tolerant scheme, RAID was soon enhanced by various double disk failure tolerance encodings, collectively known as RAID 6, including two-dimensional parity [Gibson89], P+Q Reed Solomon codes [Reed60, Chen94], XOR-based EvenOdd [Blaum95], and NetApp's variant Row-Diagonal Parity [Corbett04]. Lately research is turned to greater reliability through codes that protect more, but not all, sets of larger than two disk failures [Hafner05], and the careful evaluation of the tradeoffs between codes and their implementations [Plank09].

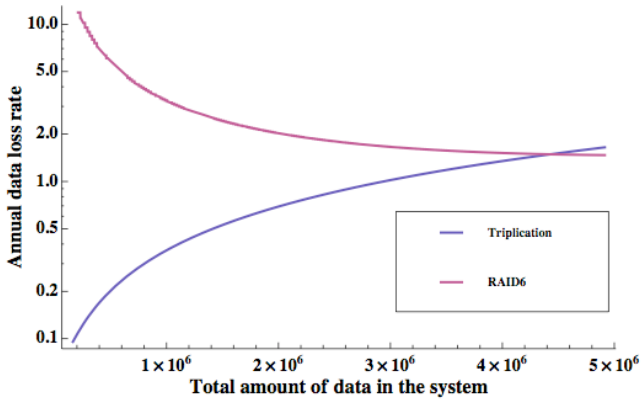


Figure 11: Annual data loss rate, in bytes, as a function of the total amount of user data, in gigabytes, for triplication and RAID 6. Repair bandwidth is limited to 10% of peak repair bandwidth and the minimum repair time is 1 hour, including failure detection and diagnosis time.

Networked RAID has also been explored, initially as a block storage scheme [Long94], then later for symmetric multi-server logs [Hartman94], Redundant Arrays of Independent Nodes [Bohossian01], peer-to-peer file systems [Weatherspoon32] and is in use today in the PanFS supercomputer storage clusters [Welch08]. DiskReduce explores similar techniques, specialized to the characteristics of large-scale data-intensive distributed file systems.

File caching has been widely used in distributed systems to improve performance [Howard88, Nelson88]. A cache representation of data on disk is often applied to balance performance requirement and storage limit given temporal locality in data access. The file system in [Cate91] combines file caching and compression in a two-level: one sector of the disk holds uncompressed data which can be accessed at normal disk speed and a sector holds compressed data which needs to be uncompressed before access. The least recently used files are automatically compressed. AutoRAID [Wilkes96] proposes a two-level storage hierarchy implemented in RAID controllers. Active data is kept in one level mirrored (with two copies) to improve reading bandwidth (e.g. read data from the copy closer, or read half of the data from each copy); while inactive data is stored in the lower level with RAID 5 protection. Migration of data between two levels is performed in the background based on the least-recently-written data dynamically determined. DiskReduce also introduces a two-layer representation: data in triplication and RAID-6 encoded.

In this work, design choices are made based on statistics (e.g. file size distribution, file access pattern) collected from clusters for cloud computing. As a comparison, the file size distribution in supercomputing file systems are reported in [Dayal08]. The access pattern and deletion pattern of UNIX BSD 4.2 file system is reported by a trace study in mid 80s [Ousterhout85]: most of the file accessed are open only a short time and accessed sequentially; most new information is deleted or overwritten within a few minutes of its creation.

Previous works address how to reduce the cost of maintaining RAID consistency (e.g. parity), without compromising data reliability. AFRAID [Savage96] always applies data update in real time but shifting parity update to idle cycles and therefore eliminates the small- update penalty by hiding the cost, with slight loss of data availability. Paritypoint [Cormen93] claims it is important for parallel file systems being able to turn off parity on a per-file basis so that applications can disable parity update for temporary files and increase I/O independence. Data is immutable in DiskReduce so RAID set is only updated after deletion. The check blocks are updated asynchronously like AFRAID, but deleted blocks are marked in real time. Therefore the consistency is always guaranteed.

In DiskReduce after deleting blocks in a RAID set, the space has to be reclaimed by garbage collection. This is similar to Log-structured file systems [Rosenblum91] where different heuristics are studied to gather the freed segments into clean segments with small overhead [Blackwell95].

Finally, our basic approach of adding erasure coding to data-intensive distributed file systems has been introduced into the Google File System [Presotto08] and, as a result of an early version of this work, into the Hadoop Distributed File System [Borthakur09a]. This paper studies the advantages of deferring the act of encoding.

7 Conclusions and Future Work

Large scale storage systems in high performance computing (HPC) and data intensive scalable computing (DISC) for internet services are too large and too heavily utilized to rely on hardware RAID 5 in each subsystem for data reliability. Redundancy codes need to protect against at least all double disk failures, and reconstruction of each failed disk needs to be parallelized to exploit all the storage and RAID controller capabilities of the entire storage system. Other differences between parallel file systems for HPC and data intensive file systems for internet service are superficial and in the process of being minimized. Notably, the storage substrate for MapReduce (HDFS in Hadoop) can be replaced with a HPC parallel file system (PVFS) and performance for DISC applications is not reduced. And RAID 5 and RAID 6 redundancy encoding can be added to HDFS.

In order to better understand the implications of stronger failure protection codes and parallel reconstruction on storage reliability, we have developed a simple model for the annual rate of data loss and applied it to triplicated and RAID 6 storage systems. We advocate annual rate of data loss instead of mean time until data loss because it is directly measurable, it differentiates between codes that lose data in smaller amounts rather than just less often, and because it draws attention to rates of data loss that are endurable rather than treating any loss as too expensive to endure.

In an example analysis of storage systems with a few petabytes of user data using 2 terabyte disks, DISC-style 64 MB blocks, RAID 6 groups of 8 data blocks and 2 check blocks, disks with annual failure rates of 2% and parallel reconstruction throttled to 10% of its peak bandwidth, the primary implication of our models is that data loss rates are measured in terms of a few bytes per year. This alone suggests that the high capital costs of triplication are not worth the factor of 10 improvement in reliability, if that improvement is from 10 to 1 byte lost per year.

This project was an initial exploration of reliability models for large-scale storage systems. The results included in this report are promising; reliability modeling for HPC and internet services can be unified in a manner indicating that it should apply to other large scale storage systems. A metric based on the amount of data lost annually is both viable and exposes intriguing new considerations – that small rates of data loss may be endurable, and apparently large differences, in terms of ratios, may "round off" to negligible differences. Further exploration is recommended.

8 Acknowledgements

This work was primarily supported through a National Archives and Records Administration supplement to National Science Foundation Cooperative Agreement NSF OCI-0852543. Additional support was provided in part by the Department of Energy, under award number DE-FC02-06ER25767, by the Los Alamos National Laboratory, under contract number 54515-001-07, by the National Science Foundation under awards CNS-0546551 and SCI-0430781, and by research awards from the Betty and Gordon Moore Foundation, Google and Yahoo!.

We also thank the member companies of the PDL Consortium (including APC, DataDomain, EMC, Facebook, Google, Hewlett-Packard, Hitachi, IBM, Intel, LSI, Microsoft, NEC, NetApp, Oracle, Seagate, Sun, Symantec, and VMware) for their interest, insights, feedback, and support.

9 References

[Bairavasundaram07] Bairavasundaram, L. N., Goodson, G. R., Pasupathy, S., Schindler, J. 2007. An analysis of latent sector errors in disk drives. In Proceedings of the 2007 ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '07), San Diego, 2007.

[Blackwell95] Blackwell, T., Harris, J., Seltzer, M. Heuristic cleaning algorithms for log-structured file systems. In Proc. of the 1995 Winter USENIX Technical Conference (New Orleans, LA, January 1995), pp. 277–288.

[Blaum95] Blaum, M., Brady, J., Bruck, J., Menon, J. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. IEEE Trans. Computers. v 44, 1995.

[Bohossian01] Bohossian, V., Fan, C. C., LeMahieu, P. S., Riedel, M. D., Xu, L., Bruck, J. Computing in the rain: A reliable array of independent nodes. IEEE Trans. Parallel and Distributed Systems, 2, 2001.

[Borthakur09a] Borthakur, D., "The hadoop distributed file system: Architecture and design." 2009. http://hadoop.apache.org/common/docs/current/hdfs_design.html.

[Borthakur09b] Borthakur, D. "HDFS and erasure codes." Aug 2009. http://hadoopblog.blogspot.com/2009/08/hdfs-and-erasure-codes-hdfs-raid.html.

[Carns00] Carns, P. H., Walter B. Ligon, I., Ross, R. B., Thakur, R. Pvfs: a parallel file system for linux clusters. In USENIX ALS'00: Proc. of the 4th Annual Linux Showcase & Conference, 2000.

[Cate91] Cate, V., T. Gross. "Combining the concepts of compression and caching for a two-level file system." In ASPLOS-IV, April 1991.

[Chen94] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., Patterson, D. A. Raid: High- performance, reliable secondary storage. In ACM Computing Surveys, 1994.

[Corbett04] Corbett, P., English, B., Goel, A., Grcanac, T., Kleiman, S., Leong, J., Sankar, S. Row-diagonal parity for double disk failure correction. In USENIX FAST, 2004.

[Cormen93] Cormen, T., and Kotz, D. Integrating theory and practice in parallel file systems. In Proc. of the 1993 DAGS/PC Symposium, 1993.

[Dayal08] Dayal, S. Characterizing HEC storage systems at rest. Tech. Rep. CMU-PDL-08-109, Carnegie Mellon University, 2008.

[Dean04] Dean, J., S. Ghemawat, "Simplified Data Processing on Large Clusters." In 6th Symposium on Operating Systems Design and Implementation (OSDI'04), 2004.

[Dholakia06] Dholakia, A., Eleftheriou, E., Hu, X., Iliadis, I., Menon, J., Rao, K. 2006. Analysis of a new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors. In Proc. of the Joint Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '06/Performance '06, Saint Malo, France, 2006.

[Elnozahy02] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys, 34(3), 2002.

[Fan09] Fan, B., W. Tantisiriroj, L. Xiao, G. Gibson, "DiskReduce: RAID for Data-Intensive Scalable Computing," Proc. of the Fourth Petascale Data Storage Workshop (PDSW09), 2009.

[Ghemawat03] Ghemawat, S., H. Gobioff, S.-T. Lueng, "Google File System." In 19th ACM Symposium on Operating Systems Principles (SOSP'03).

[Gibson09] Gibson, G., B. Fan, S. Patil, M. Polte, W. Tantisiriroj, L. Xiao, "Understanding and Maturing the Data-Intensive Scalable Computing Storage Substrate," 2009 Microsoft eScience Workshop, Pittsburgh, PA, October, 2009.

[Gibson89] Gibson, G. A., Hellerstein, L., Karp, R. M., Katz, R. H., and Patterson, D. A. Failure correction techniques for large disk arrays. ACM ASPLOS (1989).

[Grider06] G. Grider. HPC I/O and File System Issues and Perspectives. Presentation at ISW4, LA-UR-06-0473, Slides available at http://www.dtc.umn.edu/disc/isw/presentations/isw46.pdf, 2006.

[Hafner05] Hafner, J.L., "WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems." In USENIX Conference on File and Storage Technologies (FAST'05), 2005.

[Hadoop] Hadoop. Apache Hadoop Project. http://hadoop.apache.org/

[Hartman94] Hartman, J., Ousterhout, J. The zebra striped network file system. In Proc. 14th ACM SOSP, 1994.

[Holland94] Holland, M., G. A. Gibson, D. P. Siewiorek, "Architectures and Algorithms for On-line Failure Recovery in Redundant Disk Arrays," J. of Distributed & Parallel Databases, 2(3), 1994.

[Howard88] Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Side- botham, R. N., West, M. J. Scale and performance in a distributed file system. ACM Trans. Comput. Syst. 6, 1, 1988.

[Isard07] Isard, M., M. Budiu, Y. Yu, A. Birrell, D. Fetterly. "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks." In 2007 Eurosys Conference, 2007.

[Koch06] K. Koch. The new roadrunner supercomputer: What, when, and how. Presentation at SC'06, 2006.

[LANL06] Data is available at:. http://www.lanl.gov/projects/computerscience/data/, 2006.

[Long94] Long, D., Montague, B. R., Cabrera, L.-F. Swift/raid: A distributed raid system. ACM Computing Systems, 3, 1994.

[McEvoy81] D. McEvoy. The architecture of tandem's nonstop system. In ACM 81: Proceedings of the ACM '81 conference, 1981.

[Nelson88] Nelson, M., Welch, B., Ousterhout, J. Caching in the sprite network file system. ACM Transaction on Computer Systems 6(1), 1988.

[NFSv4.1] IETF. NFSv4.1 specification. http://tools.ietf.org/wg/nfsv4.

[Ousterhout85] Ousterhout, J., Costa, H. D., Harrison, D., Kunze, J., Kupfer, M., Thompson, J. A trace-driven analysis of the unix 4.2 bsd file system. In Proc. 10th ACM Symposium on Operating Systems Principles,1985.

[Patil09] Patil, S., G.A. Gibson, G.R. Ganger, J. Lopez, M. Polte, W. Tantisiriroj, L. Xiao, "In Search of an API for Scalable File Systems: Under the Table or Above It?" Workshop on Hot Topics in Cloud Computing (HotCloud09), San Diego, CA, June 2009.

[Patterson88] D. Patterson, G. Gibson, R. Katz. A case for redundant arrays of inexpensive disks (RAID). In Proc. of the ACM SIGMOD International Conference on Management of Data, 1988.

[Pinheiro07] Pinheiro, E., Weber, W., Barroso, L. A. Failure trends in a large disk drive population. In Proc. of the 5th USENIX Conf. on File and Storage Technologies (FAST07), San Jose, 2007.

[Plank99] Plank, J. S. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. Software - Practice & Experience 27(9), 1999.

[Plank08] Plank, J. S., Simmerman, S., Schuman, C. D. Jerasure: A library in c/c++ facilitating erasure coding for storage applications - version 1.2. Tech. Rep. UT-CS-08-627, University of Tennessee Department of Computer Science, August 2008.

[Plank09] Plank, J.S., J. Luo, C.D. Schuman, L. Xu, Z. Wilcox-O'Hearn. "A performance evaluation and examination of open-source erasure coding libraries for storage." In USENIX Conference on File and Storage Technologies (FAST'09), 2009.

[PVFS2] PVFS2. Parallel Virtual File System, Version 2. http://www.pvfs2.org/

[Roth06] P. C. Roth. The Path to Petascale at Oak Ridge National Laboratory. In Petascale Data Storage Workshop Supercomputing'06, 2006.

[Reed60] Reed, I. S., Solomon, G. Polynomial codes over certain finite fields. In Journal of the Society for Industrial and Applied Mathematics, 8, 1960.

[Rosenblum91] Rosenblum, M., Ousterhout, J. K. The design and implementation of a log-structured file system. ACM Transactions on Computer Systems 10, 1991.

[Ross09] Ross, Sheldon. A First Course in Probability. 8th edition. Prentice Hall. 2009.

[Savage96] Savage, S., Wilkes, J. Afraid: A frequently redundant array of independent disks. In Proc. of annual conference on USENIX Annual Technical Conference, 1996.

[Schroeder06] B. Schroeder G. Gibson. A large-scale study of failures in high-performance computing systems. In Proc. of the 2006 International Conference on Dependable Systems and Networks (DSN'06), 2006.

[Schroeder07a] Schroeder, B., G. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" Proc. of the 5th USENIX Conference on File and Storage Technologies (FAST'07), 2007.

[Schroeder07b] Schroeder, B., G. A. Gibson, "Understanding Failures in Petascale Computers," Journal of Physics: Conference Series, 78, (SciDAC07), 2007.

[Tantisiriroj08] Tantisiriroj, W., S. V. Patil, G. Gibson. "Data intensive file systems for internet services: A rose by any other

name…" Tech. Report CMU-PDL-08-114, Carnegie Mellon University, Oct. 2008.

[Top500] Top 500 supercomputing sites. http://www.top500.org, 2007.

[Weatherspoon02] Weatherspoon, H., Kubiatowicz, J. Erasure coding vs. replication: A quantitative comparison. In Proc. of IPTPS, 2002.

[Welch08] Welch, B., M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, B. Zhou. "Scalable Performance of the Panasas Parallel File System. In USENIX Conference on File and Storage Technologies (FAST'08), 2008.

[Wilkes96] Wilkes, J., Golding, R., Staelin, C., Sullivan, T. The hp autoraid hierarchical storage system. ACM Trans. Comput. Syst. 14(1), 1996.