1999 Workshop on Hot Topics in Operating Systems (HotOS)

Reliability Hierarchies

Peter M. Chen and David E. Lowell

Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
University of Michigan
{pmchen,dlowell}@eecs.umich.edu
http://www.eecs.umich.edu/Rio

Abstract

Hierarchies of diverse storage levels have been analyzed extensively for their ability to achieve both good performance and low cost. This position paper argues that we should view hierarchies also as a way to achieve both good reliability and low overhead. After discussing the design of a reliability hierarchy, we suggest two metrics to use in evaluating the overall reliability of a reliability hierarchy: mean time to data loss (MTTDL) and data loss rate. These and other metrics allow researchers to evaluate the reliability/performance tradeoffs quantitatively for new storage devices and hierarchies. We use this framework to evaluate how the Rio file cache affects the mean time to data loss and data loss rate of an existing storage hierarchy. Rio improves MTTDL over a standard delayed-write file cache by an order of magnitude and can be used with a long write-back delay without increasing data loss rate. Rio fills in the reliability gap between ordinary memory and disk by providing a storage level with high performance and intermediate reliability.

1. Introduction

Users and system builders are accustomed to thinking of storage system reliability in absolute terms. A common perception is that data is either safe on stable storage, or not safe at all, at least to a first-order approximation. Textbooks often draw a sharp dividing line between non-stable and stable storage [Mullender93], as though one were absolutely unsafe and the other absolutely safe.

In reality, systems are composed of several types of storage, each with different degrees of reliability. For example, a typical user in the Michigan EECS Department stores files in a DRAM file cache, on disk, and on backup tape. There are obvious reasons for using a combination of storage devices with different reliability and performance. For example, DRAM is not typically reliable enough (nor cheap enough) to store all files, and backup tape is not fast enough to serve as the sole on-line storage medium.

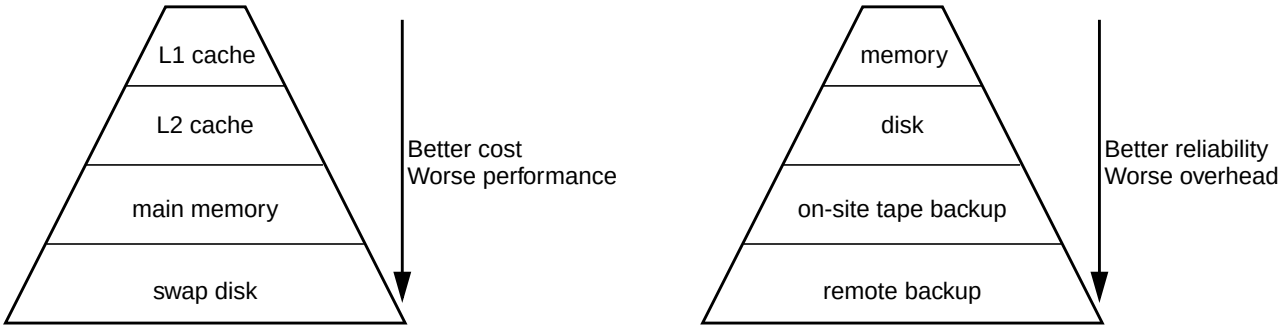
Most past research in storage hierarchies has focused on improving the performance of memory and storage hierarchies [Smith82]. Some have focused on reliability and performance but have concentrated on a single storage level in isolation [Gibson91]. To our knowledge, none have investigated or quantified in a systematic way the overall reliability of a hierarchy of storage levels.

Our position is that we should view a hierarchy of reliability levels in much the same way that we now view hierarchies for performance. We describe different aspects of designing reliability hierarchies, and we propose two simple metrics for quantifying the reliability of a given hierarchy. These and other metrics will help system designers to quantitatively evaluate tradeoffs between performance and reliability. We also show how the Rio file cache [Chen96] could fit into a reliability hierarchy, and we describe how to provide Rio for current PCs (a platform that presents several difficulties for the original approach).

2. Designing and analyzing reliability hierarchies

The basic scheme of memory hierarchies to improve performance is well known. As one proceeds down a per-

This research was supported in part by NSF grant MIP-9521386 and Intel Corporation. Peter Chen was also supported by an NSF CAREER Award (MIP-9624869).



(a) Performance hierarchy

(b) Reliability hierarchy

Figure 1: Hierarchies for Performance and Reliability. Combining multiple storage levels into a hierarchy can be used to improve performance or reliability. Performance hierarchies trade off cost for performance; reliability hierarchies trade off overhead for reliability. Overhead in a reliability hierarchy can refer to many things, such as performance, cost, or power consumption.

formance hierarchy, *cost per bit* improves and *overhead* (time to access data) worsens (Figure 1a). One can depict a hierarchy of reliability levels in much the same way (Figure 1b). As one proceeds down a reliability hierarchy, *reliability* improves and *overhead* worsens. While overhead in a performance hierarchy always refers to performance (e.g. time to access data), the definition of overhead in a reliability hierarchy depends on the application domain. For example, overhead may refer to:

- **performance**: e.g. backup tape is slower than magnetic disk.
- **cost**: e.g. mirrored disks (RAID Level 1) are twice as expensive as non-redundant disks (RAID Level 0).
- **power**: e.g. it takes more power to write data to magnetic disks than to memory.

Many aspects of performance hierarchies have analogous aspects in reliability hierarchies. For example, one metric must improve monotonically as one proceeds down the hierarchy (cost/bit or reliability), and the other metric must worsen monotonically (overhead). In either hierarchy, some applications may choose to bypass certain levels. For example, applications that exhibit poor locality may bypass an upper level in a performance hierarchy. Similarly, applications that demand a minimum reliability may bypass an upper level in a reliability hierarchy. Both types of hierarchy have similar design dimensions: which levels to include, the size of each level, the policy of when to transfer data between levels (write-back vs. write-through, replacement algorithms, prefetching, etc.).

Transfer policy is particularly important for reliability hierarchies, because it directly affects the reliability of the overall hierarchy. A straightforward policy is to write through to the lowest, most reliable level. While this yields the good reliability of the lowest level, it also suffers from the high overhead of the lowest level. Always writing through to the lowest level essentially reduces the reliability hierarchy to a single level. Most systems use a variety of levels because a single level does not suit the needs of varying applications [Lampson83]. Not all applications require the highest reliability, so they should not be forced to pay the high overhead associated with that high reliability.

Instead, most applications choose to sacrifice some reliability to achieve lower overhead. One common transfer policy is based on time. For example, transfer from a DRAM file cache to disk after 15 seconds, and transfer from disk to tape backup after 1 day. Let $delay_L$ be the age of the data before it leaves level L and is written to level L+1. Note that $delay_L$ is monotonically increasing with L. Of course, other transfer policies are possible, such as transferring after creating a certain amount of data. To simplify the discussion, we will assume a policy based on time for the remainder of this paper; other policies can often be mapped to time if the workload is known.

Transfer policy represents a tradeoff between overhead and reliability. Longer delays lead to less data being written to the lower level, because locality of reference allows some data to be written multiple times to one level before being propagated to the next level. Longer delays also allow more data to be transferred at a time, which increases transfer efficiency. However, longer delays reduce reliability, because data lives longer on a less reliable level.

Evaluating these types of tradeoffs requires metrics that quantify the overall reliability of a hierarchy. One straightforward metric is MTTDL (mean time to data loss

for the overall hierarchy). MTTDL for a hierarchy of n levels can be computed as follows:

$$MTTDL = \frac{1}{\sum_{L=1}^{n} \frac{1}{MTTF_L}}$$

where $MTTF_L$ is the mean time to failure of level L. Note that MTTDL is limited by the reliability of the least reliable level.

MTTDL is the relevant metric when losing any amount of data is equally significant. However, MTTDL is too simplistic for applications that distinguish between different amounts of lost data. For example, MTTDL does not distinguish between a large loss of data due to a disk crash and a small loss of data due to a memory error.

We propose a second metric for evaluating reliability hierarchies: *data loss rate*. Data loss rate is the fraction of data lost over time due to failures in the storage hierarchy. Data loss rate is analogous to the average access time used to evaluate performance hierarchies. Average access time for a performance hierarchy of n levels is computed as a weighted average of access times:

$$\sum_{L=1}^{n} hitRate_L \times accessTime_L$$

Similarly, data loss rate for a reliability hierarchy is computed as a weighted sum of the duration of data lost when a level fails:

$$data\ loss\ rate = \sum_{L=1}^{n} \frac{delay_L}{MTTF_L}$$

$delay_n$ is a special case, as there is no lower level to which to transfer data. One way to handle this is to assume the lowest level has an MTTF of infinity, then use a storage technology that approaches this ideal (relative to the reliability of other levels). Another way is to set $delay_n$ to the amount of useful data that would be lost if level n failed. For example, perhaps only the last year of data is valuable in a closet of backup tapes.

Both these metrics fall in the general class:

$$\sum_{L=1}^{n} \frac{f_L}{MTTF_L}$$

where f_L denotes the *value* of the data that would be lost when level L fails. Other definitions of f_L are possible. For example, f_L could be set to a sub-linear function of $delay_L$, so that new data would be weighted more heavily than old data.

The above analysis does not account for correlated failures. If a failure (such as a flood) affects multiple levels, the formulas must be adjusted to factor in a fault type exactly once. Otherwise, a single fault would lower the MTTF of multiple levels and affect the MTTDL and data loss rate multiple times. One way to correct for correlated failures is to compute metrics by summing over fault types rather than reliability levels:

$$MTTDL = \frac{1}{\sum_{\text{all fault types F}} \frac{1}{MTTF_F}} \qquad \text{(EQ 1)}$$

$$data\ loss\ rate = \sum_{\text{all fault types F}} \frac{damage_F}{MTTF_F} \qquad \text{(EQ 2)}$$

where $MTTF_F$ is the mean time to occurrence of fault type F and $damage_F$ is the duration of data lost when fault type F occurs.

3. Examples

In this section, we describe examples of the types of faults that occur in computer systems and storage devices commonly found in a reliability hierarchy. Table 1 summarizes the storage levels and faults and shows an example MTTF for each fault type.

3.1. Fault categories

We consider the following fault categories: operating system, file system, power, motherboard, media, and catastrophe.

Operating system: The frequency of operating system crashes varies widely depending on the operating system. Mature operating systems can have an MTTF measured in months, while newer operating systems may crash every few days. Unfortunately, the most common operating system in use today is reputed to fail every few days [Carreira97].

File system: A small percentage of operating system crashes corrupt permanent file system data [Chen96, Ng99]. We use a separate category for these crashes because they can affect file system data on all on-line storage devices.

Power: North American power may fail once every few months. An uninterruptible power supply (UPS) can increase the MTTF to 5-50 years [APC97].

Motherboard: This category includes hardware failures in the processor, memory, and motherboard systems. Field data and common experience indicate that hardware faults are much rarer than software faults [Gray91]. In particular, design faults in hardware are rare and can cause quite a stir when discovered (e.g. Intel's FDIV bug). Most hardware faults are due to physical phenomena such as wear-out and environmental conditions (overheating). These types of faults are relatively easy to guard against by using redundant hardware. In contrast, design faults in

Fault Category	Example MTTF	Storage Devices Affected by Fault				
		processor/ memory	disk	redundant disk array	on-site tape backup	remote-site backup
operating system	2 months	✔				
file system	5 years	✔	✔	✔		
power	2 months (no UPS) 10 years (UPS)	✔				
motherboard	5 years	✔				
media	5 years		✔			
catastrophe	50 years	✔	✔	✔	✔	

Table 1: Examples of Fault Categories and Storage Devices.

large software packages are the expected norm, and are difficult to guard against with simple replication.

Media: Disks are quite reliable, with most carrying warranties of 5 years. MTTF for disks are probably significantly longer, though most manufacturers have stopped publishing MTTF data.

Catastrophe: This category includes rare events such as fire, floods, and war.

There are also other important types of errors, such as user errors (e.g. accidentally deleting a file) and application bugs. The analysis below considers only system-level errors. Surviving user and application errors can be done with techniques such as saving recently deleted files and automatic checkpointing.

3.2. Reliability levels

Storage hierarchies may include a wide variety of levels, such as processor/memory, magnetic disk, redundant disk arrays, on-site tape backup, and remote backup.

Processor/memory forms the top level of most reliability hierarchy and includes processor registers, CPU caches, and memory. These devices are often not considered part of the hierarchy for permanent storage. However, all modern operating systems include a file or buffer cache, which caches permanent file data in this level. This level typically loses data during operating system crashes, file system crashes, power outages, motherboard failures, and catastrophes.

Magnetic disk is the mainstay of current reliability hierarchies. Disks survive most operating system crashes, power outages, and motherboard failures. However, they do not survive file system crashes, media failures (such as head crashes), or catastrophes.

Redundant disk arrays enhance reliability using a variety of error correction codes. To a first-order approximation, disk arrays survive all faults except file system crashes and catastrophes.

On-site tape backup represent another form of redundancy. Backup tapes survive most types of faults except site-wide catastrophes. Note that tapes themselves are not very reliable; however they are needed only in the infrequent case of disk crashes (or user errors).

Remote backups are used to store long-term, critical data such as financial records. These off-site vaults of storage survive essentially all faults (and if they don't, nobody will be around to debate the point).

3.3. Example hierarchy

A typical file server in the Michigan EECS Department has a UPS and a 3-level reliability hierarchy: memory, magnetic disk, and on-site tape backup.

Table 2 shows the values for our reliability metrics for this hierarchy. We compute the MTTF for a level by taking the harmonic sum of the MTTF for all faults that affect that level (applying Equation 1 for a single-level hierarchy). We compute the MTTDL for the overall hierarchy using Equation 1, because several of the fault types affect multiple levels. The overall MTTDL of the hierarchy is 1.8 months and is dominated by the frequency of operating system crashes.

To compute data loss rate, we assume a transfer policy that moves data from memory to disk 15 seconds after the

$MTTF_{memory}$	1.8 months
$MTTF_{disk}$	2.4 years
$MTTF_{tape\ backup}$	50 years
overall MTTDL	1.8 months
data loss rate	0.0011

Table 2: Reliability of an Example Hierarchy.

$MTTF_{memory}$	1.9 years
$MTTF_{disk}$	2.4 years
$MTTF_{tape\ backup}$	50 years
overall MTTDL	1.4 years
data loss rate	0.0011

Table 3: Reliability of a Hierarchy with Rio.

data is created (a 15-second delayed-write file cache), then moves data to tape backup after 1 day. We approximate the MTTF of tape as infinity. Applying Equation 2, the resulting data loss rate is 0.0011, or about 10 hours of data per year. Whereas MTTDL is dominated by the frequency of operating system crashes, data loss rate is dominated by the infrequency of backing data up to tape. This transfer policy between disk and tape is necessitated by the slow speed and high management overhead of tape backup.

4. Rio's place in the reliability hierarchy

4.1. Effect on reliability

The Rio file cache enables data in memory to survive operating system crashes as well as disks do [Chen96]. As shown in Table 3, this improves the MTTF of memory by 12x (1.9 years). Rio removes the dominant component of MTTDL in the example hierarchy in Section 3.3, improving MTTDL from 1.8 months to 1.4 years (a factor of 9x). Data loss rate in the example is dominated by the disk-tape boundary, so Rio does not improve this significantly.

While Rio does not improve data loss rate significantly, it does allow the system to use a much lazier policy for transferring data from memory to disk without increasing data loss rate. With Rio, data can stay in memory for 5 minutes and still maintain the same data loss rate as a non-Rio system would have with a 15-second delay. Remember that changing transfer policies does not affect MTTDL for the hierarchy as a whole.

Rio provides a new storage level with a unique set of characteristics. Rio fills in the "reliability gap" between ordinary memory and disk by providing a storage level with high performance and intermediate reliability (much more reliable than memory, though not as reliable against all faults as disk). We have found this new storage level to be very useful in our work on lightweight transactions, checkpointing, and distributed recovery.

4.2. Implementing Rio on PCs

The original Rio prototype was implemented on Digital Alpha workstations running Digital Unix. This implementation used *warm reboot*, which wrote file cache data to disk during reboot. Unfortunately, several aspects of PC hardware require a different approach than warm reboot. First, some PC console (BIOS) firmware clears memory during the initial phase of boot. Second, the PC reset button (if it exists) often erases memory. Third, the CPU cache uses a write-back policy and is also reset at the beginning of boot.

We have re-implemented Rio on PCs using the FreeBSD 2.2.7 operating system [Ng99][1]. The key new technique is called *safe sync*. Whereas warm reboot writes the file cache to disk during reboot, safe sync writes the file cache to disk during the last phase of the crash. Because this takes place before the system resets, this avoids the problems mentioned above. Unix kernels already try to sync data when they panic, but this fails 40% of the time on FreeBSD [Ng99]. Rio's safe sync uses several techniques to improve the probability of success. First, we minimize the use of kernel state and machine resources during safe sync. For example, we keep a separate registry that tracks the contents of the file cache, and we run safe sync on a pre-allocated stack in physical memory. Second, we modify the low-level interrupt handler to trigger a panic if it sees a special key sequence. This allows the operator to initiate safe sync even if the system deadlocks. Our interrupt handler operates below the software interrupt mask used by FreeBSD, so safe sync can be triggered even while interrupts are masked. This keyboard interrupt serves a similar purpose to the reset button on Alphas, which injects a software halt instruction into the Alpha pipeline.

After implementing these techniques, FreeBSD-Rio achieves a corruption rate of 1.9%. FreeBSD-Rio loses

1. FreeBSD-Rio is available at http://www.eecs.umich.edu/Rio.

data during operating system crashes less frequently than even a disk-based, write-through file system (3.1%).

5. Conclusions

Hierarchies of diverse storage levels have been analyzed extensively for their ability to achieve both good performance and low cost. This position paper argues that we should view hierarchies also as a way to achieve both good reliability and low overhead. We have suggested two metrics to use in evaluating the overall reliability of a reliability hierarchy: mean time to data loss and data loss rate. These and other metrics allow researchers to evaluate the reliability/performance tradeoffs quantitatively for new storage devices and hierarchies.

We use the Rio file cache to illustrate how a new storage device affects the mean time to data loss and data loss rate of an existing storage hierarchy. Rio improves MTTDL over a standard delayed-write file cache and can be used with a long delay interval without increasing data loss rate. Rio fills in the reliability gap between ordinary memory and disk by providing a storage level with high performance and intermediate reliability.

6. Acknowledgments

The ideas in this paper were developed in discussions with Subhachandra Chandra, George Dunlap, Wee Teck Ng, and Brian Noble.

7. References

[APC97] Measuring High Availability Power Protection Systems - The Power Availability Index (PA Index). Technical report, American Power Conversion, 1997. http://www.apcc.com/english/prods/dcent/symme/.

[Carreira97] Joao Carreira, Diamantino Costa, and Joao Gabriel Silva. WinFT: Software Implemented Fault Tolerance for Win32 Applications. *Byte*, February 1997.

[Chen96] Peter M. Chen, Wee Teck Ng, Subhachandra Chandra, Christopher M. Aycock, Gurushankar Rajamani, and David Lowell. The Rio File Cache: Surviving Operating System Crashes. In *Proceedings of the 1996 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 74–83, October 1996.

[Gibson91] Garth Alan Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. PhD thesis, University of California at Berkeley, December 1991. also available from MIT Press, 1992.

[Gray91] Jim Gray and Daniel P. Siewiorek. High-Availability Computer Systems. *IEEE Computer*, 24(9):39–48, September 1991.

[Lampson83] Butler W. Lampson. Hints for Computer System Design. In *Proceedings of the 1983 Symposium on Operating System Principles*, pages 33–48, 1983.

[Mullender93] Sape Mullender, editor. *Distributed Systems*. Addison-Wesley, 1993. Chapter 6.

[Ng99] Wee Teck Ng and Peter M. Chen. The Systematic Improvement of Fault Tolerance in the Rio File Cache. In *Proceedings of the 1999 Symposium on Fault-Tolerant Computing (FTCS)*, June 1999.

[Smith82] Alan J. Smith. Cache Memories. *Computing Surveys*, 14(3), September 1982.