Specifying data availability in multi-device file systems

John Wilkes and Raymie Stata

Concurrent Computing Department
Hewlett-Packard Laboratories
Palo Alto, CA

Technical report HPL–CSP–90–6

1 April 1990

Position pape r for the 4th ACM SIGOPS European Workshop, Bologna (3–5 September 1990), whose topic was "Fault tolerance support in distributed systems".

This paper was subsequently published in Operating Systems Review 25(1):56–59, Jan. 1991.

HEWLETT PACKARD

Specifying data availability in multi-device file systems

John Wilkes and Raymie Stata

Concurrent Systems Project
Hewlett-Packard Laboratories
Palo Alto, CA 94304

1 April 1990

1 Problem statement

Computers are increasing in processing power much faster than disks are reducing their access times or improving their bandwidth. This gap in dynamic performance will continue to widen over the next several years. There seem to be three ways to address this problem:

- improving the way in which existing devices are used
- using multiple devices in parallel to reduce latencies and increase bandwidth
- making the devices "smarter" by coupling processing power closely to the storage device.

The DataMesh research project at Hewlett-Packard Laboratories is investigating all three approaches in the context of a *large, fast, highly functional storage server* design for the needs of cooperating workgroups of computers (workstations, compute servers, etc) in the mid-1990s:

- *large* means in the range of 0.1–10 Terabytes of stored data
- *fast* means that getting information from the DataMesh will be faster than retrieving it from a similar number of storage devices distributed amongst the clients
- *highly functional* means that a DataMesh storage server will (eventually):
  - tolerate processor, storage device, and network failures
  - provide 10:1 scalability in capacity with smooth incremental growth
  - support many different types of structured data simultaneously (not just flat files)
  - take advantage of a storage hierarchy of multiple device types
  - offload data-intensive work from clients (e.g. image compression, tuple filtering)
  - let clients specify the performance and availability properties of their data, rather than how to store them

As a basis for our work, we have defined a *research model* of a multi-device DataMesh, with each storage device associated with a moderate-performance computing element and local RAM (perhaps 20 MIPS and 8–16 Mbytes by 1995).

An image of a DataMesh used to go here, but has suffered bit-rot over time, and no longer prints. Sorry.

Our current research emphasis is on the performance aspects of such distributed storage server designs. However, there are some factors that caused us to explicitly add consideration of high-availability ("fault tolerance") into our early deliberations:

- In the absence of a major breakthrough in storage devices, achieving high performance will require the use of massively-parallel arrays of disks. Since the failure rate of such arrays is proportional to the number of disks they contain, the mean time between failure (MTBF) is usually approximated as:

  $MTBF_{array} = MTBF_{disk} / n_{disks}$

  With individual device MTBF times of 150,000 hours [HP6000], this would lead to a device failure about once a week for a 1000-disk array.
- Centralizing storage can improve cost-performance, but it has the opposite effect on data availability: a failure can affect many more people than would an outage local to just one workstation.
- The increase in value of information to an organization, together with the growing trend toward sharing of communal data by cooperating workgroups, means that failures can have significant economic impact.

2 User-specified data properties

Everybody agrees that having data be "highly available" would be nice, but there is typically little emphasis on expressing just what this means. In fact, we assert that the following position is taken by almost all proponents of high availability systems:

> *We will provide a set of fault-tolerance mechanisms, and then educate/cajole/force users into listing the mechanisms that should be used to achieve their goals.*

For example, for a user to take advantage of a mirrored disk drive on a typical system, they have to put their files on the portion of the file system that corresponds to the redundant hardware.

Our position is rather different: we believe that the user should specify their needs in some technology-independent manner (e.g. the financial cost of failing to provide a certain data availability), and then *get the storage system itself to determine how to meet these goals*. In the mirroring example, the user would specify an availability constraint on the files of interest—and the system would be free to decide how best to satisfy it.

For lack of a better term, we have christened our approach ***user-specified data properties***. (The technique can as easily—and fruitfully—be applied to performance requirements as to availability needs, but this will not be described further here.)

Here is a sample availability specification for a file:

maximum downtime:	30 sec
maximum repair time:	1 hour
minimum failure performance:	40% of normal
maximum data loss:	0 bytes
minimum disaster interval:	100 years

The example specification given above might lead to selecting a parity disk scheme, as follows:

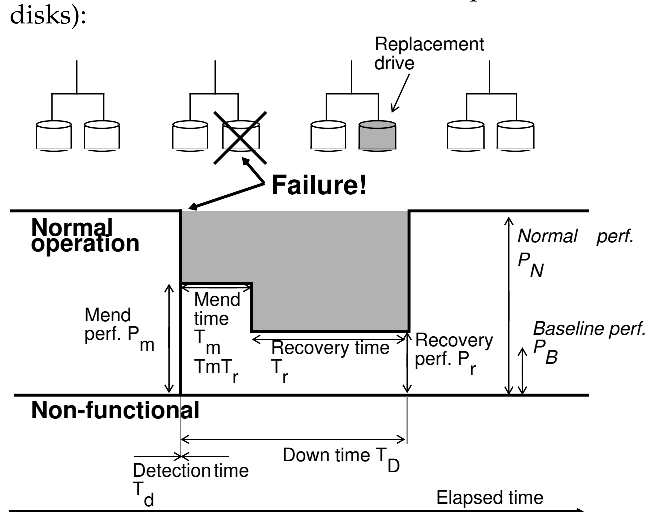
- the no-data-loss constraint would preclude use of occasional checkpointing of a copy in volatile storage that might be lost on a power failure
- the maximum repair time would preclude log-based secondary schemes (e.g. magnetic tape)
- the requirement for 40% of the failure-free access rate during repair could be satisfied with two-way mirrored disks or a parity-disk scheme
- the 100 year interval between violations of the specification would be met by a parity disk scheme with one parity disk for every m data disks, where, from [Patterson88]:

  $$MTBF_{data} = \frac{(MTBF_{disk})^2}{m(m+1) \times MTTR_{disk}}$$

  With the values given above, the maximum value of m would be about 30, assuming no account is taken of non-disk failures (e.g. power supplies, controllers, cables, etc [Schulze88]).

Exactly which arrangement is used will depend on other parameters, such as the performance constraints (which might force the selection of mirroring), or cost assumptions (such as "choose the solution that uses the least disk capacity").

Underlying our choice of specification parameters is a model of failure behaviour that encompasses the effects of the initial failure, recovery, and subsequent repair. The following graphic outlines the effects of a single, isolated failure (in this case, to one of a pair of mirrored disks):



Under normal operation, the specification calls for a performance behaviour P_N (e.g. via throughput or response time constraints) and a mean time between violations of T_N.

On a failure, the system proceeds through a number of stages, with characteristic durations as follows:

- *detection time T_d*: time to notice that a fault has manifested itself; it is usual to assume that this time is vanishingly short
- *mend time T_m*: physical repair time, such as replacing a faulty unit, or simply switching to a spare device. This can range from milliseconds (with an online spare) to days or even weeks (if a unit has to be sent away to be repaired).
- *recovery time T_r*: the time to bring the system back to normal operation after the mechanical repair has been accomplished. This usually involves regenerating redundant information, and sometimes allows continued operation as the recovery is performed (e.g. with a mirrored disk). Recovery times can range from minutes to hours, depending on how much information has to be restored, and the relative priorities of ongoing operations and recovery.
- *failure time $T_F = T_d + T_m + T_r$*: time before normal operation is restored
- *downtime T_D*: time before the system meets minimal user specifications (in the case of the mirrored disk example above, $T_D = T_d$, since operations can continue immediately on the second drive, but in some examples T_D might occur at the end of T_m, or even part-way through T_r).

In addition to the time parameters in the model, there are a number of performance specifications of potential interest:

- *normal performance P_N*: the performance specification for normal (failure-free) operation
- *mend performance P_m*: the performance specification during the physical repair period
- *recovery performance P_r*: the performance during the period when redundant data is being restored
- *failure performance $P_F = min(P_m, P_r)$*
- *baseline performance P_B*: the minimally-acceptable performance that can meet user "degraded mode" operation requirements. Continuous operation is possible if P_F P_B

We expect that users will get most of what they need from the model from specifying limits on T_D and T_F, and minimum values for P_N and P_B.

Three other sorts of constraints are also important:

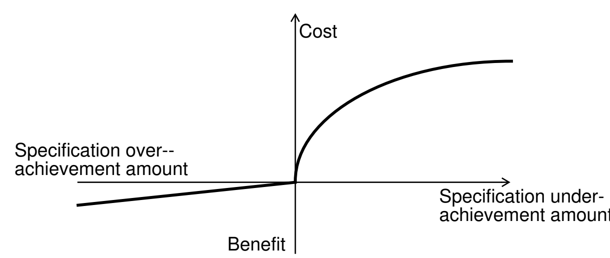
- *maximum data loss:* amount of "already-written" data that may be sacrificed during the recovery process; specified as one of:
  - percentage of total data
  - number of I/O operations
  - bytes of data
- *disaster interval:* minimum period before violating the availability specifications
- *"fuzziness" parameters*: we have presented the resource needs so far as if they were absolutes. Given that failures are themselves probabilistic in nature, it may be more appropriate to provide probabilistic constraints in the specifications, such as "meet these availability goals with 90% confidence". (There are many examples of this in the civil engineering world, such as specifications for "hundred year flood barriers".)

Finally, the issue of the *cost of under- or over-compliance* needs addressing. Since availability is important, presumably there is some cost associated with failure to comply with the specification; by the same token, there may be some benefit from being more aggressive than required. To allow the system to decide which goals are the most important, a cost/benefit model is needed for deviations from the specifications.

There are two obvious kinds of cost that have to be modelled:

- *direct, or resource costs*: the cost (in megabytes, disks, dollars, etc) of providing the storage needed to meet the users' constraints.

  It is easy to give the system the raw data to calculate this. In the absence of other information, it would be reasonable to set a policy of choosing the lowest-direct-cost solution for a given set of constraints (e.g. don't use triple redundancy if simple mirroring will do).
- *indirect, or corollary costs*: economic consequences external to the system of violating the specifications (e.g. losing the payroll file). Such costs are likely to be non-linear with the degree of violation. For example:



We should make it clear that these are our ideals; pragmatic considerations in the execution of the constraint models needed will doubtless limit the search space that can be explored. (For example, a simple linear—or even static—approximation to the cost model may be appropriate.)

3 Benefits

The benefits of this approach are many-fold:

- *ease of use*: the user can achieve their availability (and performance) objectives without needing to know about how the storage is configured, what the characteristics of the individual devices are, or what set of mechanisms are the best ones to use
- *better resource utilization*: since the system has information about the exact mix of components available, it can do a better job of allocating resources to meet needs than could a human
- *dynamic adjustment*: the system can dynamically adapt to changing user needs and available storage capacity
- *take advantage of new technology*: as new storage devices become available, the system can make full use of their characteristics (e.g. there is a steady trend toward more reliable disk drives: this scheme can adapt to a mixture of old and new drives without dropping back to the lowest-common-denominator)

4 Research issues

The promise of this approach seems clear, but there are a number of research issues that still need to be addressed. For example:

- Is the set of parameters we have described necessary and sufficient?
- Is this approach (and our parameters) sufficiently simple for users to understand?
- What set of cost metrics are appropriate to prevent users demanding 100% availability with zero downtime for everything?
- What are good heuristics to solve the resource allocation problems involved? (The equivalent performance-related assignment problems have been shown to be NP-hard.)
- How many (and which) simplifications can be made in the cost modelling and specification to achieve the twin goals of adequate match to needs and minimum-overhead allocations?
- What are appropriate ways to handle the availability implications of client caching?
- What is the set of mechanisms that should be considered?
- How can users express their needs in a convenient way?

5 Related work

IBM's *System-managed storage* model [Gelb89] is in some ways similar. The emphasis of the IBM approach is on allowing a system manager to choose a small suite of mechanisms, and then having the system operate within the constraints these imply (e.g. cylinder and disk placement is left to the system, but the choice of mirroring over parity disks is made by hand). We go further, and suggest that the system should itself proactively select the mechanisms to meet performance and availability constraints. Our scheme also allows the selection of mechanisms to vary over time (e.g. as a new more cost-effective device becomes available, or as new usage patterns develop).

Work on the RAID project at UC Berkeley is addressing the availability characteristics of disk arrays and checksum/parity disk schemes (e.g. [Schulze88, Chen88]).

6 Conclusions

We believe that specification of *user-oriented* measures for reliable systems is the key to ease of use, efficient resource utilization, and performance optimization. Initial investigation suggests that a relatively small number of parameters can be used to define availability goals. It also seems that interactions between availability and performance specifications can be described in a fairly clean way. Although the approach shows promise, there are still several hard research questions to be answered, and doubtless many interesting things to be learned.

References

[Chen88] Peter Chen, Garth Gibson, Randy H. Katz, David A. Patterson and Martin Schulze, *Two papers on RAIDs.* Technical report UCB/CSD 88/479, Department of Electrical Engineering and Computer Sciences, UC Berkeley, December 1988.

[Gelb89] J. P. Gelb, "System managed storage". *IBM Systems Journal* **28**(1): 77–103, 1989.

[HP6000] *HP Series 6000 Models 330S and 660S Mass Storage Systems: Technical data*. Part number 5952–0631. Hewlett-Packard Company*, 1989.*

[Patterson88] David A. Patterson, Garth Gibson and Randy H. Katz, "A case for redundant arrays of inexpensive disks (RAID)". *Proceedings of the ACM SIGMOD Conference* (Chicago, Illinois) 1–3 June 1988.

[Schulze88] Martin E. Schulze, *Considerations in the design of a RAID prototype.* Technical report UCB/ CSD 88/448, Department of Electrical Engineering and Computer Sciences, UC Berkeley, August 1988.