;login:

THE MAGAZINE OF USENIX & SAGE
August 2002 volume 27 • number 4

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

an introduction to dependability

Definitions and Examples

by David A. Patterson

David Patterson is Professor of Computer Science at the University of California, Berkeley. He implemented one of the first RISC microprocessors and invented, along with Randy Katz, the Redundant Arrays of Inexpensive Disks (RAID),

patterson@cs.berkeley.edu

To improve dependability of systems, the Recovery-Oriented Computing (ROC) project is creating technology that will let systems recover more quickly from failures [Patterson et al. 2002]. We are especially interested in services accessed over a network, such as Internet sites and enterprise data centers. Since system administrators are the ones called when systems fail, we want to start a conversation about dependability problems in the hopes of developing technology that will really help.

One persistent difficulty with the general topic of making computers systems that can survive component faults has been confusion over terms. Consequently, perfectly good words like reliability and availability have been abused over the years so that their precise meaning is unclear.

Clearly, we need precise definitions to discuss such events intelligently. As a first step in a conversation about dependability, we define the dependability terminology: fault, failure, reliability, availability, mean time to failure (MTTF), and mean time to repair (MTTR). We also show how to calculate MTTF of a system given the MTTF of its components.

This paper is derived from Chapter 7 of Hennessy and Patterson [2002]. It provides a simplified version of definitions used by the IEEE Computer Society Technical Committee on Fault Tolerance and the IFIP working group 10.4. This paper is the first in a series; future papers will talk about issues relevant to the system administration community using these definitions.

Defining Dependability, Reliability, and Availability

The research community picked a new term – dependability – to have a clean slate to work with: computer system dependability is the quality of delivered service such that reliance can justifiably be placed on this service. Each component of that system also has an ideal specified behavior, where a service specification is an agreed description of the expected behavior. A system failure occurs when the actual behavior deviates from the specified behavior. The failure occurs because of a fault, a defect in that component.

We can now explain reliability and availability. Users may see a system alternating between two states of delivered service with respect to the service specification:

1. Service accomplishment, where the service is delivered as specified
2. Service interruption, where the delivered service is different from the specified service

Transitions between these two states are caused by failures (from state 1 to state 2) or restorations (2 to 1). Quantifying these transitions lead to the two main measures of dependability.

Reliability is a measure of the continuous service accomplishment (or, equivalently, of the time to failure) from a reference initial instant. Hence, the mean time to failure of disks is a reliability measure. The reciprocal of MTTF is a rate of failures. Service interruption is measured as mean time to repair. The related term mean time between failures (MTBF) is simply the sum of MTTF and MTTR. Although MTBF is widely used, MTTF is often the more appropriate term, as repair times may be harder to predict.

Availability is a measure of the service accomplishment with respect to the alternation between the two states of accomplishment and interruption. Module availability is statistically quantified as:

Availability = MTTF/(MTTR+MTTF)

Note that availability and reliability are now quantifiable metrics, rather than synonyms for dependability. Availability ranges from 0% to 100%, with 100% being perfect; reliability as measured by MTTF ranges from 0 to infinity, with infinity being perfect.

Calculating MTTF and Availability

If we assume that the age of the component is not important in its probability of failure and that failures are independent, the overall failure rate of a subsystem is just the sum of the failure rates of the modules. Let's do an example. Assume a disk system with the following components and rated MTTF:

- 1 SCSI controller, 500,000 -hour MTTF
- 1 power supply, 200,000-hour MTTF
- 1 fan, 200,000-hour MTTF
- 1 SCSI cable, 1,000,000-hour MTTF
- 5 SCSI disks, each rated at 1,000,000 -hour MTTF;

We can compute the MTTF of the system as a whole by adding the failure rate of each component, which is the inverse of its MTTF.

$$\begin{aligned}\text{Failure rate}_{\text{system}} &= 1/500{,}000 + 1/200{,}000 + 1/200{,}000 + 1/1{,}000{,}000 + (5 * 1/1{,}000{,}000)\\ &= (2 + 5 + 5 + 1 + 5)/(1{,}000{,}000 \text{ hours})\\ &= 18 / (1{,}000{,}000 \text{ hours})\end{aligned}$$

The MTTF of the system is just the inverse of the failure rate of the system:

$$\text{MTTF}_{\text{system}} = (1{,}000{,}000 \text{ hours}) / 18 = 55{,}555 \text{ hours}$$

If the average MTTR is one day for this system, the estimated availability would be

$$\text{Availability}_{\text{system}} = 55{,}555 / (55{,}555 + 24)$$

which is about 99.96%. Marketing departments have shorted availability from the actual percentages to the number of leading 9s in the percentage. Thus, 99.96% can be called "3 nines" of availability. Well-run servers achieve 2 or 3 nines of availability, or 99% to 99.9%.

Failures vs. Faults

The difference between faults and failures aren't as obvious as you might think.[1] Here are some examples of the difficulties.

- Is a programming mistake a fault or a failure? Does it matter whether we are talking about when the program was designed, or when it is run? If the running program does not exercise the mistake, is it still a fault or failure?
- Suppose bits on disk in a RAID system change due to a problematic sector in a disk. Did a fault or failure still occur if the error correction codes (ECC) of the sector delivers the corrected value to the processor? Is it a fault or failure if it was an uncorrectable fault according to the disk, but the RAID system corrects it?
- The same difficulties concerning data change, latency, and observability arise with a mistake by a human operator.

NOTE

1. The dependability community makes the subtle distinction between a defect that does not change anything and a defect that does change the state [Gray and Siewiorek 1991, Laprie 1985]. They call the former a fault and the latter an error. An example is an Alpha particle hitting a DRAM cell. That collision is a fault, and it is only an error if it changes the value in the DRAM cell. Although this distinction is more precise, it is often confusing, resulting in debates on whether something is a fault or error. In this paper we concentrate on the differences between defects and service outages, which we call faults and failures.

Initially, a fault is considered latent and becomes effective when it is activated. For example, a programming mistake is a latent fault until that code is invoked by the system. If the fault actually affects the delivered service, a failure occurs. The time between the occurrence of a fault and the resulting failure is the latency. Thus, a failure is the manifestation on the service of a fault. Reviewing the properties of fault:

- A latent fault becomes effective once activated.
- An effective fault often propagates from one component to another, thereby creating new faults.

Thus, an effective fault is either a formerly latent fault in that component or it has propagated from another fault.

Reviewing the fault-failure sequence, the steps are latent faults, then effective faults, and finally, if it disrupts the delivered service, a failure.

Let's go back to our motivating examples above. A programming mistake is a fault; upon activation, the fault becomes effective; when this effective fault produces erroneous data which affect the delivered service, a failure occurs. For the disk example, the flaw in the sector is a fault. If the ECC corrects the fault, the RAID system would not observe it. If the disk could not correct it, and thus has a failure, then RAID system would see a fault. If the RAID system corrected it, the operating system would not see a fault. A mistake by a human operator is a fault; it is latent until activated; and so on as before.

These properties are recursive and apply to any component in the system. That is, a defect is either a fault or a failure depending on your perspective. For example, the specified behavior of a disk is to deliver correct sectors when requested. Thus an uncorrectable read fault is a failure from the disk perspective, but it is a fault from the perspective of the RAID system. Confusion between faults and failures often depends on how you draw the boundaries around the system and hence what is the expected service of that system.

A programming mistake is a latent fault until that code is invoked by the system.

Categorizing Faults and Ways to Handle Them

The purpose of this section is to familiarize you with some terms that you may see when looking at systems that claim greater dependability. There are many ways to categorize faults. We show two ways – by duration or by cause – to give you some intuition about how to talk about faults. Classifying by their duration yields three options:

1. Transient faults exist for a limited time and do not recur.
2. Intermittent faults cause a system to oscillate between faulty and fault-free operation.
3. Permanent faults do not correct themselves with the passing of time; they remain until repaired.

The classification above shows a hierarchical taxonomy of faults based on cause. The first split is whether it is physical or logical, where all software and operator faults are logical. Hardware faults are either due to problems in manufacturing, in operation, or in design. Manufacturing faults are either individual flaws or due to problems in the manufacturing process. Physical operation faults are either the result of wear or of environmental problems, such as power outages, high temperature, fire, flood, earthquake, and so on. Design faults may simply be bugs in hardware or software, or not designing-in sufficient margins in hardware to handle normal variations in, say, volt-

As many as half of disk failures are due to problems in shipping.

age. Finally, logical operation faults can be people breaking into the system or mistakes by operators, although poor design of the user interface and documentation leads to operator mistakes.

Just as there are many ways to categorize faults, there are many ways to categorize systems' handling of them. Laprie [1985] divides improvements into four methods:

1. Fault avoidance: how to prevent, by construction, fault occurrence – that is, preventing the creation of latent faults.
2. Fault tolerance: how to provide, by redundancy, service complying with the service specification in spite of faults having occurred or occurring – that is, preventing faults from becoming failures.
3. Fault removal: how to minimize, by verification, the presence of latent faults.
4. Fault forecasting: how to estimate, by evaluation, the presence, creation, and consequences of faults, and thus take preemptive action to prevent the fault from turning into a failure without necessarily using redundancy.

A final topic is repair. Some systems or modules are repair tolerant, in that you can safely repair them while the system continues to operate. For example, many systems allow disks to be hot swapped without shutting down the computer. Some systems and modules are repair intolerant. For example, you often must shut down the system before replacing the motherboard.

Drawbacks to MTTF and the Definition of Failure

One drawback of MTTF calculations is that they imply a comfort zone that is not merited in practice. First, although MTTF is just the inverse of the failure rate, it is not intuitive. For example, a million-hour MTTF means the mean time to failure is over 100 years. Does this mean the average disk lasts 100 years? No. Since manufacturers calculate disk lifetime as five years, it means that if you bought many disks and copied the data to a new drive every five years, on average you could do it 20 times before you saw a failure. Annual failure rates are a better match to human intuition. For example, if we make common assumptions about the distribution of independent failures, about 1% of components would fail in each of the first few years if each component was rated as a 100-year MTTF.

The second drawback is that MTTF assumes failures are independent and that they are based on MTTF numbers supplied by the manufacturer. The manufacturer supplies MTTF rates assuming the products were not damaged in shipping and that they were operating in nominal conditions of temperature and voltage. As many as half of disk failures are due to problems in shipping. High operating temperature due to fan failure, something blocking the air flow, or air-conditioning failure can severely shorten lifetimes. Such environmental problems can also violate the independent-failure assumption.

The purpose of MTTF calculations is to show relative reliability of different designs rather than to predict what you will see in practice. For example, the calculation above shows that MTTF is limited by the weakest link in the chain. To significantly improve the reliability of this subsystem, we would need a more reliable power supply and fan. If those two components were unchanged, even if we had perfect controllers, cables and disks, the MTTF would be capped at a tenth of the reliability of one disk.

As it is either expensive or impossible to replace components with more reliable versions, the primary way of coping with failures is redundancy. For example, one of the long-standing guidelines in design is to have no single point of failure. We will talk about calculating the reliability of redundant systems in a future paper.

A final note about the definition of failure itself. The terminology takes the simplified view that the system is either accomplishing service or there is a service interruption. A more nuanced view sees a third state, service degradation, whereby the service is not interrupted but is performing poorly enough to be a problem. It also assumes a single service, although Internet sites like eBay and Yahoo offer a collection of services. We will tackle a more nuanced definition of failure in future papers.

Conclusion

The goal of this paper is to begin to pierce through the fog of dependability terminology. We distinguished a fault from a failure, showing the difference can simply be a matter of perspective. We also gave quantitative definitions of reliability and availability, and provided an example of how to estimate MTTF and availability. Finally, we warned to not be too comfortable with high MTTF, since the numbers can be misleading.

Future papers will talk about redesigning systems so that there are no single points of failure, statistics collected on why systems fail, the cost of downtime, and the goals of the Recovery-Oriented Computing project.

We hope to initiate a series of conversations about why current systems fail and how researchers can help create a new foundation for systems that are easier to operate.

Acknowledgements

I would like to thank Aaron Brown, George Candea, David Oppenheimer, and Mike Patterson for comments on earlier drafts of this paper. This work is supported by the National Science Foundation, grant no. CCR-0085899, the California State MICRO Program, Allocity, Hewlett Packard, IBM, and Microsoft.

REFERENCES

Gray, J., and D. P. Siewiorek. 1991. High-availability computer systems. *Computer* 24:9 (Sept.): 39–48.

Gray, J., and A. Reuter. 1993. Fault tolerance. Chapter 3 of *Transaction processing: concepts and techniques.* San Francisco: Morgan Kaufmann Publishers.

Hennessy, J. L., and D. A. Patterson. 2002. *Computer architecture: A quantitative approach*. 3d ed. San Francisco: Morgan Kaufmann Publishers.

Laprie, J.-C. 1985. Dependable computing and fault tolerance: Concepts and terminology. Fifteenth Annual International Symposium on Fault-Tolerant Computing FTCS 15. *Digest of Papers*. Ann Arbor, MI, USA (June 19–21), 2–11.

Patterson, D., A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, N. Treuhaft. 2002. Recovery-oriented computing (ROC): Motivation, definition, techniques, and case studies. *U.C. Berkeley Computer Science Technical Report* UCB//CSD-02-1175, March 15, 2002 (see *http://roc.cs.berkeley.edu*).