"Breaking the Four 9s Barrier" An Informational Series on Enterprise Computing

As Seen in *The Connection*, An ITUG Publication
September 2002 – December 2003

About the Authors:

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein, have a combined experience of over 90 years in the implementation of fault-tolerant, highly available computing systems. This experience ranges from the early days of custom redundant systems to today's fault-tolerant offerings from HP (NonStop) and Stratus.

Series Topics:

***Gravic, Inc*.**
Shadowbase Products Group
17 General Warren Blvd.
Malvern, PA 19355
610-647-6250
www.ShadowbaseSoftware.com

Availability (Part 4) - The Facts of Life

Dr. Bill Highleyman
Paul J. Holenstein
Dr. Bruce D. Holenstein

In the first three parts of this series on availability[1], we focused on some basic concepts and have applied these concepts to the development of architectures that could provide significantly enhanced availability. The model that we used was that of a system that comprised multiple identical subsystems. Of these subsystems, *s* were spares, and the system could tolerate the failure of any *s* subsystems. However, the failure of *s*+1 subsystems might cause the failure of the system. In the event of a system failure caused by *s*+1 subsystem failures, the system was immediately restored to service upon the repair of one of the failed subsystems.

This model is accurate for replicated and split systems. For instance, if a split system comprises two nodes, and should both nodes fail, then service is restored as soon as one of the failed nodes is repaired and brought back into service (assuming that no database recovery is required before the system can be used).

However, within a failed node, things are not so simple. Once the requisite number of subsystems have been repaired and are operational, the node often must be recovered before it can be returned to service. This may require a variety of actions taking several hours.

In this our fourth part on Availability, we take a look at the impact of system recovery which may have to follow system repair. But first, let us review what we have done to date.

A Review of Availability

In our previous parts, we developed the general availability relation

$$A \approx 1 - f(1-a)^{s+1} \approx 1 - f\left(\frac{mtr}{mtbf}\right)^{s+1} \qquad (1)$$

where

[1] Highleyman, W., "Availability Part 1- The 9s Game", The Connection, Volume 23, No. 6; November/December, 2002.
Highleyman, W., Holenstein, B. "Availability Part 2- System Splitting," The Connection, Volume 24, No. 1; January/February, 2003.
Highleyman, W. Holenstein, P. "Availability Part 3- Synchronous Replication," The Connection, Volume 23, No. 2; March/April, 2003.

A	is the availability of a system comprising similar redundant subsystems.
a	is the availability of a subsystem.
s	is the number of spares provided in the system; that is, any *s* subsystems may fail, and the system will continue to operate.
f	is the number of failure modes, or the number of ways that *s*+1 subsystems will fail in such a way that a system outage would result.
mtr	is the mean time to repair a subsystem.
mtbf	is the mean time before failure for a subsystem.

Availability is the probability that the system will be operational and is therefore

$$A = \frac{MTBF}{MTBF + MTR} \qquad (2)$$

where

MTBF	is the mean time before failure for the system.
MTR	is the mean time to repair the system (or, more accurately, the mean time to *restore* the system to service)

We discussed the characterization of availability in terms of 9s; e.g., an availability of .9999 is called four 9s. We showed that adding a backup doubled the nines, which is the basic power of NonStop systems. We discussed the relative efficiencies of methods to keep replicated data bases in exact synchronization. We also showed that splitting a system into several independent nodes using synchronous data replication could provide substantial improvement in availability at little or no additional cost.

However, systems of multiple redundant processors suffer from a plurality of failure modes which could reduce system availability by one or two 9s. We showed that by careful allocation of processes to processors, we could substantially reduce this impact.

We so far have applied these concepts to hardware failures as if hardware failures were the predominant cause of system outages. This has been the mechanism that allowed us to demonstrate the application of these concepts. However, this is not real life. Recent experience indicates that only a very small portion of system outages are caused by dual hardware failures. The rest are caused by complex interactions between hardware failures, software failures, operational errors, and environmental faults.

Adding to the rules which we derived in our earlier parts, we have

Rule 11 : *Redundant hardware systems have an availability of five to six nines. Software and people reduce this to four nines or less.*

The work to date is no waste of time. The concepts are valid. But in order to apply these concepts to real-life availability, we have to understand better what is going on.

Why Do Computers Stop?

Jim Gray is unquestionably one of the key contributors to fault-tolerant computing. In 1985, he published "Why Do Computers Stop and What Can be Done About It,"[1] a defining paper on the real causes of system outages. This paper formed the basis for his chapter on "Software Fault Tolerance" in his subsequent book "Transaction Processing: Concepts and Techniques,"[2] which sets forth the basic fault-tolerant principles underlying NonStop systems.

His footprints can be found all throughout this paper. Rather than citing each reference, let us suggest that you read his paper, which can be found at http://www.cs.berkeley.edu/~yelick/294-f00/papers/Gray85.txt.

Though Gray's paper was published in 1985, it remains strikingly applicable today. Little has changed except that hardware has become somewhat more reliable as he predicted. We summarize his observations next with some updating comments.

Gray studied 166 unscheduled system outages reported to Tandem over a seven-month period. These outages covered over 2000 systems. Note that this equates to a system availability of .99993 assuming an MTR of four hours (see later), confirming NonStop's claim to availabilities of about four 9s.

About one-third of the reported outages related to "infant mortality" problems, defined as recurring problems which were later fixed. If these were taken out, 107 system outages were reported during this period. These outages were characterized as follows:

	Gray 1985	**Standish 2002**
People	42%	38%
Software	25%	28%
Hardware	18%	17%
Environment (power, a/c, etc.)	14%	17%
Unknown	1%	-

Contributors to NonStop Outages
Table 1

There are some caveats that must be made about these observations. First, they probably don't include many outages caused by application software faults or environmental problems. Customers don't generally report these. Second, they probably don't include all operator errors. Operators don't always report their goofs. Finally, they

[1] Gray, J., "Why Do Computers Stop and What Can Be Done About It?" 5th Symposium on Reliability in Distributed Software and Database Systems; 1986.
[2] Gray, J., "Transaction Processing: Concepts and Techniques," Morgan Kaufmann; 1993.

don’t include scheduled downtime; but HP has an active initiative to eliminate such downtime.

Gray points out in his 1985 paper that “….hardware will be even more reliable due to better design, increased levels of integration, and reduced numbers of connectors.” In fact, this forecast has turned out to be quite correct. Experience now indicates that outages due to dual hardware failures represent less than 5% of all outages.

Gray goes on to say, “….the trend for software and system administration is not so positive. Systems are getting more complex.” A recent study by The Standish Group[2] supports this observation. Not much has changed in these nearly two decades. Citing a NonStop server reliability of .9998, their measurements of the causes of system outages are also shown in Table 1. They are amazingly consistent with Gray’s findings. Interestingly, the Standish study also indicated that network failures happened more often than server system failures; and outages caused by applications swamped server system failures by almost four to one.

Note that about 40% of all outages seemed to have been caused by human error. However, this isn’t as bad as it may look. 45 outages per year over 2000 systems represent one human error per system every 44 years. Don’t we all wish that we were that accurate!

An interesting insight into operator errors has been nicely phrased by Wendy Bartlett of HP. It recognizes the stress factor which accompanies an unexpected failure:

> **Rule 12:** *When things go wrong, people get stupider.*

Today, the numbers may be a little different. In fact, they may be different each year because the sample size is too small. However, all indications are that the story these numbers tell is the same:

> **Rule 13:** *System outages are predominantly caused by human and software errors.*
>
> **Rule 14:** *Seldom does a recovery entail hardware repair. It entails a reload of the system.*

Let us consider further these two important observations.

Some Definitions

Let us first define some terms.

A *fault* is a lurking incorrectness waiting to strike. It may be a hardware or software design error, a hardware component failure, a software coding error, or

[2] The Standish Group, VirtualBEACON, Issue 244; September, 2002.

even a bit of human ignorance (such as an operator's confusion over the effects of a given command).

A *failure* is the exercise of a fault. Failures in themselves do not cause outages. NonStop systems are designed to survive any single failure.

An *outage* is a denial of some service to part or all of the user community. Outages can range from unacceptable response times to total system unavailability.

A *trigger* is an initial failure that begins an event sequence that leads to a subsequent failure from which the system cannot recover. The result is a system outage.

A *repair* is the return to service of a component which has experienced a failure.

A *recovery* is the return to service of a system which has experienced an outage.

Note that the *repair* of a failed subsystem does not necessarily result in the *recovery* of its system.

Triggered Outages

Our availability analysis so far has assumed (in today's NonStop environment) that an outage is caused by two failures *and* that these two failures are independent (i.e., one failure does not induce the other). Though simplistic, this view of things has led us to several important concepts in availability.

But we now see that dual failures which cause outages are hardly independent at all. Rather, one random failure leads to a directly related second failure, which causes the outage. For example,

- a disk unit fails, and the good disk is erroneously pulled for replacement.

- a critical process aborts, and a bug in its backup checkpointing procedures causes the backup to fail as well.

In general, a failure which must be corrected occurs but otherwise does not seriously affect system operation. But then that failure triggers another failure in the recovery process. We call this a *failover fault*, and the failover fault is fatal.

This leads to another interesting observation. You may have noted that new systems seem to be less reliable than established systems. A system seems to "burn in" with time. How can this be?

New systems are subject to continuous change. Functional errors are corrected, bugs are worked out, enhancements are made. Each change carries with it the potential for further errors which may act as outage triggers. As the system matures, changes become less frequent and system reliability improves.

Rule 15: *Change causes outages.*

The Impact of Failover Faults

In Part 1 of this series, we showed that the probability of an outage for a system configured with one spare and with randomly distributed processes is

$$F \approx \frac{n(n-1)}{2}(1-a)^2 \qquad (3)$$

where

F	is the probability of a system outage.
n	is the number of subsystems in the system.
a	is the probability of a subsystem failure.

That is, a subsystem will fail with a probability of (1-*a*). A pair of subsystems will fail with a probability of $(1-a)^2$, and there are *n*(*n*-1)/2 ways in which two subsystems might fail.

This relation assumes that the outage is caused by the independent failures of two subsystems, whether those failures are caused by hardware or by software. Implicit in Parts 1 and 2 was the assumption that critical process pairs were trusted and would not fail. A system outage was caused by a dual hardware failure which took down a critical process pair or which denied a critical process pair access to data which it needed. We now relax that assumption and allow a subsystem to fail due either to a hardware failure or to a software failure. Furthermore, we realize that the failover mechanism which should have recovered from that failure may fail itself, creating a failover fault.

Thus we now know that not all outages are caused by dual independent failures. With a probability of *p*, a *single* subsystem failure, whether it be due to hardware or software, will experience a failover fault, leading to a system outage. Only (1-*p*) of all outages are caused by dual subsystem failures:

p is the probability that a failover attempt will fail (a failover fault).

We now have two failure modes for a system:

- two subsystems have failed.
- one subsystem has failed, and the failover has failed.

Furthermore, there are two components to system restoration – subsystem repair and system recovery. Let

r	be the mean time to repair a subsystem.
R	be the mean time to recover a system.

Also, remember that the average time to return one subsystem to service when there are two failed subsystems is *r*/2 (see Rule 6 in Part 1).

Let us consider each of the two failure modes listed above.

(1) Two subsystems have failed.

The probability that two subsystems will fail is given by Equation (3) above. Of all outages, this will occur (1-*p*) of the time. Furthermore, the effective system repair time is the time to repair one of the subsystems, *r*/2, plus the time to recover the system, *R*, giving a total repair time of *r*/2+*R* rather than just *r*/2. Since downtime is proportional to repair time, the failure probability for this mode is increased by a factor of (*r*/2+*R*)/*r*/2:

$$\text{outage probability due to dual failures} \approx \frac{r/2+R}{r/2}(1-p)\frac{n(n-1)}{2}(1-a)^2 \qquad (4a)$$

(2) One subsystem has failed, and failover has failed

The probability that a particular subsystem will fail is (1-*a*). There are *n* ways in which a system can experience a single subsystem failure. Thus the probability that one subsystem in the system will fail is *n*(1-*a*). Of all outages, *p* are caused by single subsystem failures followed by a failover fault.

Should an outage be caused by a failover fault, there is no need for a subsystem repair since no more than one subsystem is down. Thus, subsystem repair time, *r*, has been replaced by system recovery time, *R.* Therefore, the probability of failure for this mode is modified by a factor of *R*/*r*:

$$\text{outage probability due to failover fault} \approx \frac{R}{r}pn(1-a) \qquad (4b)$$

Recognizing that the system failure probability *F* is the sum of the above two probabilities, and assuming that *p* is very much less than 1, then

$$F \approx \frac{r/2+R}{r/2}\frac{n(n-1)}{2}(1-a)^2 + \frac{R}{r}pn(1-a) \qquad (4c)$$

The following observations are made about this relationship:

Observation 1: A Better Value for Subsystem Availability *a*

So far, we have assumed that system availability *A* is about four 9s, given a subsystem availability *a* of .995. Now we can deduce a better value for *a* from Equation (4c).

Let us consider an 8-processor system (n = 8) with an availability *A* of four 9s (*F* = .0001). We assume a repair time *r* of 24 hours, a recovery time *R* of 4 hours, and a failover fault probability of 1%. Solving Equation (4c) for *a*, we find that a better value for subsystem availability *a* is .9986, more than three times better than we had previously assumed. This is because we now recognize that some outages are caused by failover faults and not by dual subsystem failures.

Observation 2: Effect of Failover Faults on System Availability

We can calculate from Equations (4) that a 1% failover fault rate causes 20% of all system downtime time under the previous example. Note from Equation (4b) that this is directly affected by system recovery time *R*. To the extent that we can reduce recovery time, we can minimize the effects of failover faults.

Observation 3: Effect of Failover Faults on Effective Subsystem Availability

A further insight into the impact of failover faults on system availability is gained as follows.

We can rewrite Equation (4c) as

$$F \approx \frac{r/2 + R}{r/2} \frac{n(n-1)}{2} (1-a)(1-a') \qquad (5)$$

where

$$a' = a - \frac{R}{r/2 + R} \frac{p}{n-1} \qquad (6)$$

Note that *a'* is less than a, being reduced by a subtractive term. Thus, comparing Equation (5) to Equation (4a) (and ignoring the 1-*p* term as being very close to one), we can make an interesting interpretation. The failure of the first subsystem will occur with a probability of (1-*a*) as expected. **However, once one subsystem has failed, the system then behaves as if it comprises *n*-1 remaining subsystems with decreased availability *a'*.**

A simple example will serve to illustrate this. Let us consider a 4-processor node (n = 4) comprising subsystems with an availability, *a*, of .9986 (as calculated above). Subsystem repair time, *r*, is 24 hours and system recovery time, *R*, is four hours. If the probability of a failover fault, *p*, is 1%, then the effective subsystem availability, *a'*, following a single subsystem failure, is reduced from .9986 to .9978. Failure probability has increased from .0014 to .0022. Under the above parameter assumptions, a 1% chance of a failover fault makes the system 60% less reliable following a single subsystem failure!

This leads to the following rule:

Rule 16: *Following the failure of one subsystem, failover faults cause the system to behave as if it comprises n-1 remaining subsystems with decreased availability.*

Note also that, as recovery time *R* decreases, Equation (6) shows that the reduced subsystem availability *a'* improves and approaches the subsystem availability *a.* Thus, reducing recovery time directly reduces the impact of failover faults on system availability.

<u>Observation 4: Effect of Failover Faults on System Splitting</u>

In Part 2, we showed that splitting a system into *k* nodes improved system reliability by at least a factor of *k*:

$$\text{Split System Reliability Improvement} = k\frac{n-1}{n-k} > k \qquad (7)$$

where

k is the number of nodes into which the system is split.
n is the number of processors in the system.

Thus, Equation (7) predicts that the reliability improvement achieved by system splitting is always greater than *k*. However, this gain becomes compromised when we have the possibility of failover faults. This relation then can be shown to become

$$\text{Split System Reliability Improvement} = \frac{(n-1)+x}{\frac{1}{k}(n-k)+x} \qquad (8)$$

where we can think of *x* as an exasperation factor. If *x* is very small, then Equation (8) approaches Equation (7); and we have the split system advantage we are seeking. However, if *x* is very large, then Equation (8) approaches one; and the reliability advantage of system splitting disappears.

x is given by

$$x = \frac{R}{r/2+R}\frac{2p}{1-a} \qquad (9)$$

Note that as recovery time *R* becomes smaller, *x* becomes less significant; and we recover our availability advantage provided by system splitting. Again, minimizing recovery time is of the utmost importance.

As an example, from Equation (7) we would expect that splitting a 16-processor system into four 4-processor nodes would give us a reliability advantage of a factor of 5. But suppose that we have a failover fault probability, *p,* of 1%. Furthermore, assume that subsystem availability, *a,* is .9986, that subsystem repair time, *r,* is 24 hours, and that system recovery time, *R,* is four hours. In this case, from Equations (8) and (9), the availability advantage of system splitting has decreased from a factor of 5 to a factor of 2.8.

The Golden Rule – Reduce Recovery Time

We have seen the importance of minimizing recovery time to improve system availability. A further insight can be gained by noting that about 20% of CPU halts are caused by hardware failures, and about 80% are caused by software faults or human errors. Thus, only about 4% of system outages (20% x 20%) are caused by dual hardware failures.

The remaining 96% of outages are caused by no more than one hardware failure combined with a software fault or a human error. These outages do not require a repair to return them to service. They only require a recovery:

> **Rule 14 (restated):** *A system outage usually does not require a repair of any kind. Rather, it entails a recovery of the system.*

We now have seen that recovery time is a predominant factor in system availability:

- The time to recover from most outages is recovery time rather than repair time. Therefore, any reduction in recovery time is *directly* reflected in system availability.
- Minimizing recovery time helps negate the effect of failover faults on system failure rate.
- Minimizing recovery time helps maximize the availability advantages of system splitting.

This leads to what might be considered the golden rule of availability:

> **Rule 17:** *Design your systems for fast recovery.*

What can we do to reduce recovery time? The recovery process is quite complex. It may entail

- realizing that a problem has occurred.
- diagnosing the cause of the problem.
- deciding what to do.
- obtaining permission to follow a course of recovery action.
- collecting diagnostic data (such as a processor memory dump).
- cold-loading the system.
- restarting software subsystems (TM/MP, Spooler, SQL, etc.)
- restarting the applications.
- restarting the network.
- perhaps recovering the data base.

A typical recovery procedure takes anywhere from an hour to several hours. Four hours is a pretty good guess at an average recovery time.

As can be seen from the above list, recovery time isn't minimized by simply building our applications for quick recovery, though this, of course, is very important. Efficient recovery also requires

- good operator training.
- efficient decision making by the entire management team.
- well-documented recovery procedures for
  - the system.
  - the application.
  - the data base.
  - the network.

Rule 18: *Rapid recovery of a system outage is not simply a matter of command line entries. It is an entire business process.*

The Importance of Restore Time

Let us define *restore time* as the time required to return a system to service. Restore time may entail a hardware repair and probably requires a system recovery. The *MTR* term in Equation (2) should really stand for *mean time to restore*.

The importance of restore time (which can also be called downtime) so far has only been considered as a factor in availability. To the extent that one can reduce restore time, one can increase system availability.

However, restore time has a far greater importance in its own right. It may be that the average cost per downtime hour is a user's measure of the value of reliability. But this cost is also a function of the length of a downtime interval. As downtime grows longer, the outage cost may increase. Customer annoyance may give way to customer anger, then lost sales, then lost customers.

Downtime is even more profound for safety critical functions. For instance, 911 operators will probably feel that a five-second outage is an annoyance. But a five-minute outage can mean a death due to cardiac arrest or a building being burned to the ground.

NonStop systems are full of outages that are seen not as outages at all but as very brief periods of perhaps degraded response times. These are the normal outages caused by subsystem failures that are recovered in seconds by a process-pair backup or by resubmitting a Pathway transaction to a surviving server. It is the severe outages requiring a system recovery that currently limit NonStop systems to four 9s.

Since downtime is predominantly recovery time for NonStop systems, we could perhaps add a nine to the current NonStop availability if we paid more attention to designing systems and business processes for fast recovery. An excellent example of this is a major corporation which runs a variety of applications on several NonStop systems. Every one of their critical applications is backed up by another system that either normally runs other applications or is a dedicated backup. Recovery time for many of their replicated applications is about two minutes. A typical NonStop system has an availability of four 9s and an average recovery time of four hours. By cutting the recovery time from four hours to two minutes – over a 100:1 improvement in restore time – they have added two 9s to their availability and have created an availability in excess of six 9s.

Summary

In the first four parts of this series, we have explored basic availability concepts and their application to software configuration, system replication and splitting, repair time, and recovery time. We summarize by listing the various rules we have developed for availability.

Rule 1: *If all subsystems must be up, then the availability of the system is the product of the availabilities of the subsystems.*

Rule 2: *Providing a backup doubles the 9s.*

Rule 3: *System reliability is inversely proportional to the number of failure modes.*

Rule 4: *Organize processors into pairs, and allocate each process pair only to a processor pair.*

Rule 5: *System availability increases dramatically with increased sparing. Whatever the availability of a subsystem is, each additional level of sparing adds that many 9s to the overall system availability.*

Rule 6: *For a single spare system, the system MTR is one-half the subsystem mtr.*

Rule 7: *For the case of a single spare, cutting subsystem mtr by a factor of k will reduce system MTR by a factor of k and increase the system MTBF by a factor of k, thus increasing system reliability by a factor of* k^2.

Rule 8: *If a system is split into k parts, the resulting system network will be more than k times as reliable as the original system and still will deliver (k-1)/k of the system capacity in the event of an outage.*

Rule 9: *If a system is split into k parts, the chance of losing more than 1/k of its capacity is many, many times less than the chance that the single system will lose all of its capacity.*

Rule 10: *For synchronous replication, coordinated commits using data replication become more efficient relative to dual writes under a transaction manager as transactions become larger or as communication channel propagation time increases.*

Rule 11 : *Redundant hardware systems have an availability of five to six nines. Software and people reduce this to four nines or less.*

Rule 12: *When things go wrong, people get stupider.*

Rule 13: *System outages are predominantly caused by human and software errors.*

Rule 14: *A system outage usually does not require a repair of any kind. Rather, it entails a recovery of the system.*

Rule 15: *Change causes outages.*

Rule 16: *Following the failure of one subsystem, failover faults cause the system to behave as if it comprises n-1 remaining subsystems with decreased availability.*

Rule 17: *Design your systems for fast recovery.*

Rule 18: *Rapid recovery of a system outage is not simply a matter of command line entries. It is an entire business process.*

What's Next?

We have developed in the last four parts of this series several concepts regarding availability and the related parameters of mean time before failure and mean time to repair. We have discussed the impact of system recovery on these parameters. In "Availability (Part 5) – The Ultimate Architecture," we put these concepts to work to suggest a system architecture that will dramatically improve all of these parameters at little additional cost.