"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 1) – The 9s Game

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

True or False:

- *Adding processors to your NonStop system increases its reliability.*
- *A 16-processor NonStop system has the reliability of a UNIX box.*
- *A NonStop processor is less reliable than a UNIX processor.*

These are indeed provocative questions, and their answers are not straightforward. This paper develops some simple, though not necessarily intuitive, concepts that help answer these and other questions. More importantly, these concepts lead to some straightforward steps that you can take to improve the reliability of your system – steps as simple as being aware of how you allocate processes to processors.

This is the first paper in a five part series. The successive papers are co-authored with Dr. Bruce Holenstein and Paul J. Holenstein and explore in more depth certain aspects presented in this paper:

Availability (Part 2) – System Splitting will point out that splitting a system, a common architecture for disaster recovery, significantly improves reliability at little or no additional cost.

Availability (Part 3) – Synchronous Replication will compare the efficiencies of synchronous replication techniques that may be used to keep database replicates in a split system in exact synchronism, thus avoiding database corruption due to update collisions.

Availability (Part 4) – The Facts of Life will explore what really makes systems fail and what if anything we can do about it. This part draws heavily on actual experience documented by Jim Gray, one of the significant contributors to NonStop computing.

Availability (Part 5) – The Ultimate Architecture will use the concepts explored in the previous parts to suggest a system architecture that can dramatically increase system availability at little additional cost.

Availability (Part 6) – RTO and RPO explains the concepts of "Recovery Time Objective (RTO)", or the measure of how much time it takes to recover from a disaster, and "Recovery Point Objective (RPO)", or the measure of how much data is lost in the event of a disaster, and how the different recovery architectures affect each.

What is Reliability?

The questions above all refer to *reliability.* But before we go much further, we have to agree on how to measure reliability.

There are actually two components that impact the reliability of a system – how long it will work before it fails and then how long it will take to fix it. We call the first the *mean time before failure* (MTBF) and the second *the mean time to repair* (MTR).

To a space satellite designer, reliability is MTBF. Once the satellite fails, it is gone forever. It is not repairable. Clearly, a satellite with a 10-year MTBF is ten times more reliable than a satellite with a one year MTBF.

However, when it comes to life and property protection, reliability is MTR. In a 911 system, an outage of 30 seconds may be simply an aggravating hiatus; but an outage of one hour could mean death by cardiac arrest or a building burned to the ground.

In large transaction processing systems, reliability is often measured as down time. Down time has a cost associated with it – perhaps $1,000 per hour or $100,000 per hour. (Of course, MTR plays a role here as well, as the longer the down time, the higher the cost in many cases – from customer annoyance to lost sales to lost customers.) Down time alternatively can be measured as the proportion of time that a system is up, a measure that we call the *availability* of the system. Since the system is always either up or down, then

$$\text{availability} = A = \frac{MTBF}{MTBF + MTR} \qquad (1a)$$

Note that this also can be written as

$$A = \frac{1}{1 + \frac{MTR}{MTBF}} \approx 1 - \frac{MTR}{MTBF} \qquad (1b)$$

where "≈" means "approximately equal to," and the approximation is valid so long as MTBF is very much greater than MTR (which certainly is true in the cases that we will be considering).

It is availability that this paper is all about. When we speak of reliability, we mean availability. A system with .999 availability is more reliable than a system with .99 availability.

More specifically, we will compare the reliability of systems by comparing their probability of failure. If a system has an availability of .99, then it has a probability of failure of 1-.99, or .01. That is, it will be down 1% of the time. Using Equation (1b),

$$\text{probability of failure} = F = 1 - A \approx \frac{MTR}{MTBF} \qquad (2)$$

Jim Gray has characterized availability and reliability in a very folksy way[1]:

"Availability is doing the right thing within the specified response time. Reliability is not doing the wrong thing."

As outlined above, we apply measures to these:

Availability = Doing the right thing = *A*
Failure = Doing the wrong thing = *F* = (1-*A*)
Reliability = Not doing the wrong thing = 1/(1-*A*)

Consider System A, which is up 99% of the time, and System B, which is up 99.9% of the time. System A has an availability of .99, a failure probability of .01, and a reliability measure of 100. System B has an availability of .999, a failure probability of .001, and a reliability measure of 1000. Thus we say that System B is ten times more reliable than System A.

This is how we will use the term "reliability" throughout this paper.

Some Caveats

There is some algebra used in this paper to develop availability concepts. About the worst relationship looks like

$$A \approx 1 - f(1-a)^{s+1}$$

If you are algebraically challenged, don't despair. Just skip the math and grasp the concept. The concepts are clearly stated and don't depend upon the math for understanding. Besides, charts and tables are provided so that you can use these relationships without ever breaking out a calculator.

Also, we are interested in developing concepts and rules of thumb. This requires that we take a simplistic view of things. You will be tempted to say, "Yes, but my system does this" or "You haven't considered that." True, but we are taking a 50,000 foot view of things in order to develop some general concepts. Moreover, the concepts developed herein will allow those of you who are motivated to drop down to a 5,000 foot view. A 500 foot view, however, is probably obscured by a lack of good data and too many trees in your way.

[1] Gray, J., "Why Do Computers Stop and What Can Be Done About It," Tandem Technical Report 85-7; June, 1985.

The simplistic view presented in this paper is most applicable to repairable systems. Software failures generally are recovered rather than repaired. They are more complex and are considered in Part 4.

9s – The Measure of Availability

When we calculate availability for today's systems, we will get numbers like .99999. Saying this gets cumbersome and can lose the meaning. So we talk about availability in terms of 9s. ".99999" is "five 9s." ".998" is "a little less than three 9s." ".99992" is "a little more than four 9s."

Though we will speak of 9s, this measure can be converted to average down time over any given period, as shown in Table 1.

Nines	% Available	Hours/Year	Minutes/Month
2	99%	87.60	438.
3	99.9%	8.76	43.8
4	99.99%	.88	4.38
5	99.999%	.09	.44
6	99.9999%	.01	.04

Average 24x7 Down Time
Table 1

Of course, an availability of three 9s does not mean that the system will be down 8.76 hours each year. It means that over a sufficiently long period of time, one can expect that the system will be down eight or nine hours per year. This could occur as short 1 minute failures every 17 hours or as a one-day failure every three years.

Specifically, knowing the availability tells us nothing about the MTBF or MTR. But knowing the availability and either MTBF or MTR tells us the other. More to the point, from Equation (1) we can deduce that

$$MTBF \approx MTR/(1-A) \quad (3)$$

$$MTR \approx MTBF(1-A) \quad (4)$$

Thus, if we know that our availability is three 9s and if we have an MTR of 4 hours, then we have an MTBF of 4,000 hours.

Today's Systems

How do today's systems rate so far as availability is concerned? Results compiled by the Gartner Group[2] indicate the following availabilities:

[2] Gartner Group; 2002.

NonStop	.9999
Mainframe	.999
Open VMS	.998
AS400	.998
HPUX	.996
Tru64	.996
Solaris	.995
NT Cluster	.992 - .995

Thus mainframes are four to five times more reliable than UNIX systems, and NonStop systems are ten times more reliable than mainframes.

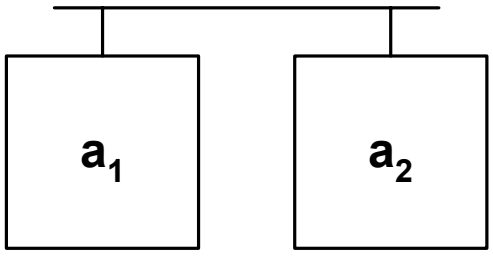
Simple Systems

Let us start our conceptual journey by looking at the two simplest systems and review a little probability theory along the way.

We consider a system made up of subsystems, each with an availability of *a*. The availability of the entire system is *A*.

a) Non-Redundant System

Figure 1 shows a system comprising two non-redundant subsystems. Both must work in order for the system to work.



Non-Redundant System
Figure 1

Let the availability of subystem 1 be a_1 and of subsystem 2 be a_2. Remember that each of these availabilities is the probability that the subsystem will be operational. In order that the system be operational, subsystem 1 *and* subsystem 2 must be operational. The probability of this is the product of the component probabilities:

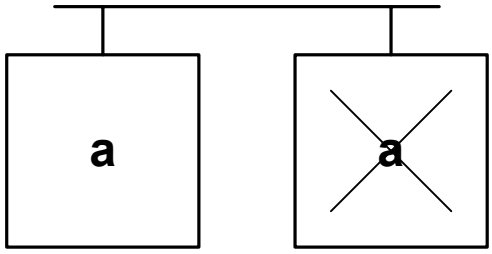
$$A = a_1 a_2 \qquad (5)$$

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

b) Redundant System

Figure 2 shows a redundant system comprising two identical subsystems, but in this case the system is operational if subsystem 1 is operational *or* if subsystem 2 is operational. In order for the system to be down, subsystem 1 *and* subsystem 2 must be down. Since the probability that either subsystem will be down is (1-*a*), then the probability that both will be down is $(1-a)^2$. The system availability is therefore

$$A = 1-(1-a)^2 \qquad (6)$$



Redundant System
Figure 2

Double Your 9s

Let us explore Equation (6) a little further. If the subsystem availability *a* is .99, then the system availability *A* is

A = 1 - .01 x .01 = .9999

Note that we have doubled the 9s from a subsystem availability of two 9s to a system availability of four 9s.

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

This is the basis for the high reliability of NonStop systems and for the even higher reliability that can be achieved by replicating a system using data replication techniques (more about that in Part 2).

The Real NonStop World

Redundant systems are the basis for the high availability (and high scalability as well) of NonStop systems. But they are a bit more complex than the simple systems which we have just considered:

- They contain multiple processors (2-16 per node).
- They comprise multiple redundant subsystems – processors, processes, disks, communications, ServerNet fabric.
- Processes critical to system operation are replicated as process pairs.
- Processes are distributed randomly across processors (usually to satisfy load balancing considerations).

Consistent with our 50,000 foot view, we will consider a NonStop system as a single group of like subsystems. This is a more accurate representation of K-series systems, but for our purposes it will be applicable to S-series systems as well. After all, mirrored disk pairs and communication channels are assigned to processor pairs; and we assume that the reliability of the ServerNet fabric is high enough to be ignored.

Therefore, a subsystem is a processor and its collection of disks and other peripherals. Certainly in a real system each subsystem will be somewhat different since different processors have associated with them different numbers of devices, but our assumption that all subsystems are similar is warranted by the simplifications that allow us to develop some general concepts.

Furthermore, we will assume a subsystem availability of .995. This is close to the K-series subsystem availability of .996 reported to the author by Tandem in the mid-1990s. Today's systems undoubtedly comprise more reliable components and are manufactured using higher quality techniques, but they are also more complex. It is therefore assumed that this number is still in the ballpark.

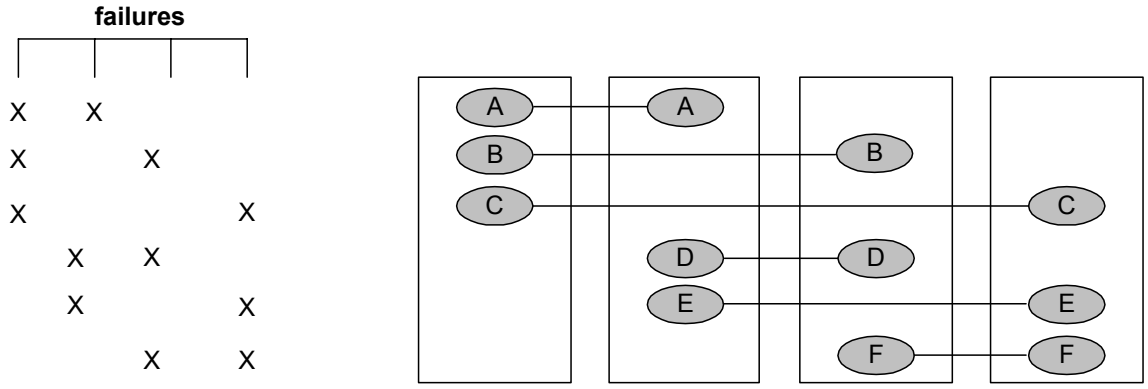
Note that this value for availability includes all sources of failure: hardware, software, maintenance, and operations. More about that in Part 4.

Randomly Distributed Process Pairs

So far as availability is concerned, the heart of a NonStop system is its critical processes. The loss of any one of these processes will cause a system failure, either immediately or after a short period of time due to system degradation. These processes include the disk processes (DP2), terminal control processes (TCP), PATHMON, and a slew of monitors for communication and other subsystems.

Therefore, these processes are provided as process pairs so that they will survive any single processor failure. Coupled with transaction protection that guarantees that no data will be corrupted as a result of a fault, these features provide the high availability for which NonStop systems are known.

But a dual processor failure may take down a critical process and result in a system outage. Let us take a look at a four-processor system in which critical processes are randomly distributed across all processors so that any dual processor failure will take down the system (Figure 3).



Any pair of processor failures causes a system failure

Randomly Distributed Processes
Figure 3

Note that there are six possible ways that two out of four processors can fail. We call these *failure modes.* Since any given two processors will fail with a probability of $(1-a)^2$, and since there are six ways that this can happen, then the system will fail with a probability of $6(1-a)^2$. Thus, its availability is[3]

$$A \approx 1 - 6(1-a)^2$$

You can probably figure out that for n processors the number of failure modes is $n(n-1)/2$. For the above example, n=4 and the number of failure modes is 4x3/2=6.

Process/Processor Pairing

Figure 4 shows an alternate strategy for distributing process pairs. Processors are organized into pairs, and process pairs are constrained to run only in processor pairs. For a four-processor system, there are only two failure modes. Either the first pair of

[3] This relation is an approximation since it does not account for failure modes involving more than two subsystems. Since the probability of three or more failures is extremely small, this relation is quite accurate. Besides, it's a lot simpler than the fully accurate relation.

processors must fail or the second pair must fail in order to cause a system failure. Thus, the availability of this configuration is

$$A \approx 1 - 2(1-a)^2$$

This configuration has three times the reliability of the randomly distributed configuration. In general, for *n* subsystems, the number of failure modes is *n/2* for this strategy.

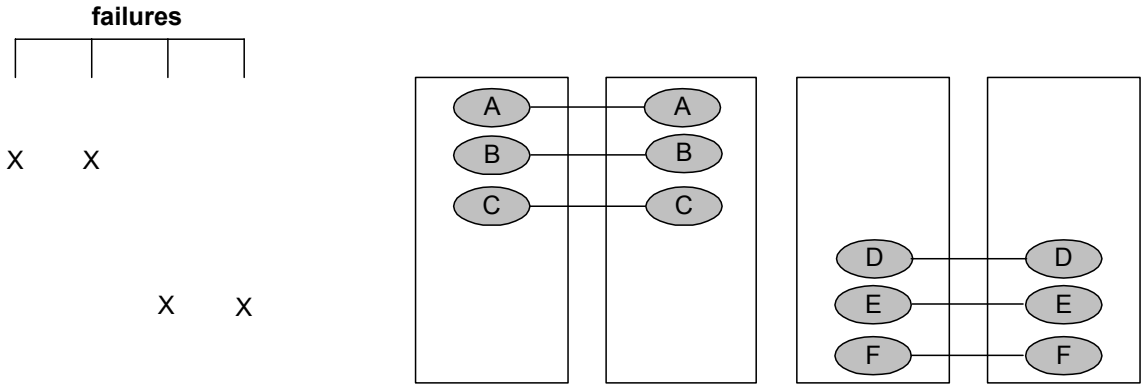
In fact, the advantage of process/processor pairing gets better as the system gets larger. Consider an eight processor system (where f is the number of failure modes):

	n	**a**	**f**	**A**
random	8	.995	28	.9993
paired	8	.995	4	.9999

For an eight processor system, a paired configuration is seven times as reliable as a random configuration. For a sixteen processor system, reliability is improved by a factor of 15!

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.*



Only certain pairs of processor failures cause a system failure

Process Pairing
Figure 4

Availability in General

We have seen above that failure probability is proportional to the number of failure modes. Thus, letting *f* be the number of failure modes, we may write

$$A \approx 1 - f(1-a)^2$$

This relationship assumes that each process is backed up by only one other process (we can consider the backup process as a spare which is put into service if the primary process fails).

But what if we have two spares? Then any given failure occurs only with a probability of $(1-a)^3$. In general, if we have s spares, then the system will fail only if we have $s+1$ subsystem failures. Any particular failure of $s+1$ subsystems will occur with a probability of $(1-a)^{s+1}$, and the system availability becomes

$$A = 1 - F \approx 1 - f(1-a)^{s+1} \qquad (7)$$

This is our general availability equation.[4] Note that it reduces to our simple example represented by Equation (6) for $f=1$ and $s=1$.[5]

There is one assumption that is inherent in our discussion so far, and that is that the system is returned to service as soon as a failed subsystem has been repaired. It needs no further recovery. This assumption is explored further in Part 4.

More Sparing

Note that from Equation 7, reliability increases exponentially with the number of spares. If *a* is .99, each additional level of sparing adds another two 9s to the system availability. Single sparing gives a system availability of four 9s, double sparing gives an availability of six 9s, and so on.

> **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.*

How do we increase process sparing in NonStop systems? There are two cases:

- For checkpointed process pairs, allow a process to start a new backup in a surviving processor if the process loses its backup due to a processor

[4] As we said earlier, this is an approximation. However, not only is it conservative in that it gives a lower value for *A* than the actual value, but it is within 5% for the range of values in which we are interested.
[5] This result is an extension of an excellent summary of availability found in Chapter 8, "Reliability Calculations," Burt H. Liebowitz and John H. Carson, "*Multiple Processing Systems for Real-Time Applications*," Prentice-Hall;1985.

failure.

- For persistent processes that are restarted in another processor by a monitor should the process fail because of a processor failure, give the monitor the choice of more than two processors in which to start the process (of course, the monitor must be redundant as well).

How Many Failure Modes?

As we have discussed, the worst availability case is the random distribution of processes. For *n* subsystems and *s* spares, the number of failure modes for this case is the number of ways that *s*+1 subsystems can fail out of *n* systems.[6] These maximum values for *f* are shown in Table 2.

We can see from this table that the maximum number of failure modes for a single-spared 16 processor system is 120. However, we know that if we pair processors and processes, we can reduce the number of failure modes to 8, a 15:1 reduction as we have earlier noted.

	Processors (n)							
	2	4	6	8	10	12	14	16
Spares (s)								
0	2	4	6	8	10	12	14	16
1	1	6	15	28	45	66	91	120
2		4	20	56	120	220	364	560
3		1	15	70	210	495	1001	1820
4			6	56	252	792	2002	4368
5			1	28	210	924	3003	8008
6				8	120	792	3432	11440
7				1	45	495	3003	12870
8					10	220	2002	11440
9					1	66	1001	8008
10						12	364	4368
11						1	91	1820
12							14	560
13							1	120
14								16
15								1

Maximum Failure Modes (f)
Table 2

[6] For Math Nuts: This is n!/(n-s-1)!(s+1)!

Since reliability is proportional to failure modes (Rule 3), we can lose more than a nine from our achievable availability for a 16 processor system if we are not careful with process allocation. More about this later.

The Impact of Repair Time

So far we have talked about availability as the predominant measure of reliability. But as we indicated in the opening to this paper, the system mean time to repair, MTR, is often an equally important parameter. Let us look at system MTR and its relation to subsystem mean time to repair, mtr.

From Equation (1a), we can express subsystem availability *a* in terms of its *mtbf* and *mtr*:

$$a = \frac{\text{mtbf}}{\text{mtbf} + \text{mtr}}$$

where

a	is the subsystem availability.
mtbf	is the subsystem mean time before failure.
mtr	is the subsystem mean time to repair.

Note that we are using upper case MTBF and MTR to represent the system, and lower case mtbf and mtr to represent the subsystem.

A little algebraic manipulation results in

$$1 - a = 1 - \frac{1}{1 + \frac{\text{mtr}}{\text{mtbf}}} \approx \frac{\text{mtr}}{\text{mtbf}} \qquad (8)$$

The approximation depends upon mtbf being much greater than mtr. This is certainly true by orders of magnitude in the systems that we are considering.

Substituting Equation (8) into Equation (7), we have

$$A = 1 - F \approx 1 - f\left(\frac{\text{mtr}}{\text{mtbf}}\right)^{s+1} \qquad (9)$$

We see that reliability is exponentially affected by subsystem mtr. For one spare ($s = 1$), the system failure probability will be cut by a factor of 4 if we can cut subsystem mtr in half.

But how does subsystem mtr affect the overall system MTR and its MTBF? It can be shown[7] that

$$\mathrm{MTR} = \frac{\mathrm{mtr}}{(s+1)} \tag{10}$$

For one spare, system MTR is half the subsystem mtr. Thus, for one spare, if our subsystem mtr is four hours, then our system MTR is two hours.

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

Furthermore, if we reduce mtr by a factor of *k*, we will reduce MTR by a factor of *k*. Since we have seen that the failure probability will be reduced by k^2 (Equation (9)), then from Equation (2) we can conclude that we will increase our system MTBF by a factor of *k*.

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.

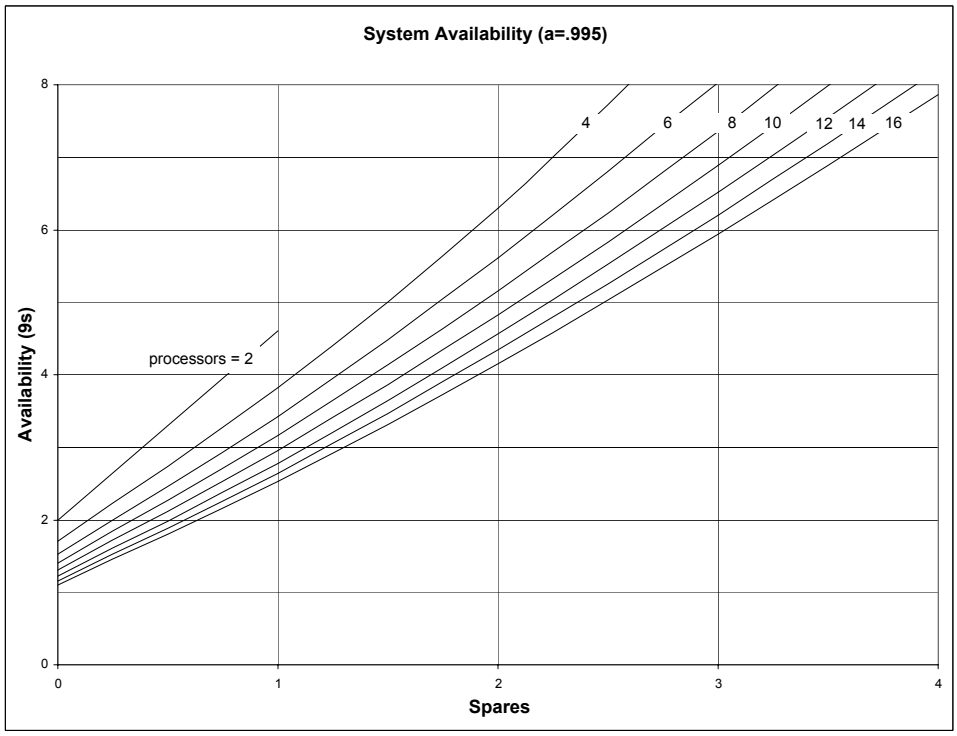
For instance, let us say that our system has an MTBF of five years and an mtr of 4 hours, leading to an MTR of two hours. If we can cut mtr in half to two hours, our system MTR will be reduced to one hour; and our system MTBF will be increased to ten years. Our reliability has increased by a factor of four as indicated above.

Some Helpful Charts

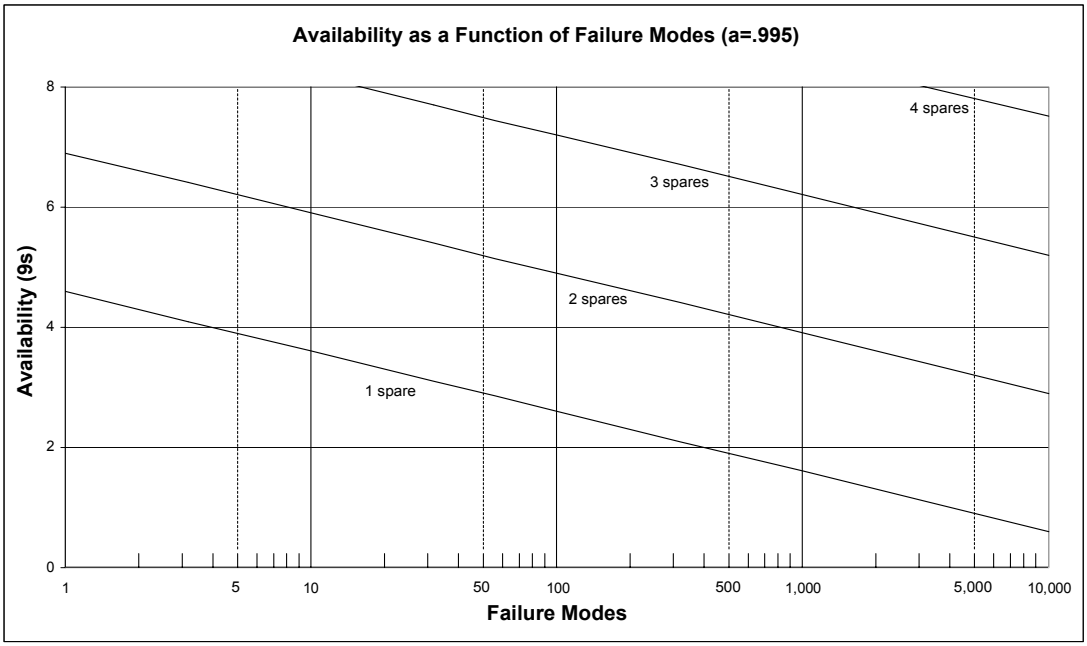
Figure 4 shows availability as a function of the number of processors and the number of spares for random distribution of processes. Note that no matter the number of processors, each additional spare adds about two 9s to the system availability.

Figure 5 shows availability as a function of spares and failure modes. Let's look at process pairing and random distribution for 16 processors. For process pairing (f=8), system availability is almost four 9s. For random distribution (f=120), system availability is about two and a half nines. My! That's about the availability of a UNIX box.

[7] Highleyman, W. H., "The Impact of Mean Time to Repair on System Availability," ITI, Inc. white paper, August 19, 2002.



Availability as a Function of Processors and Spares
Figure 5



Availability as a Function of Spares, Failure Modes
Figure 6

Answers

Let's return to our initial provocative statements.

- *Adding processors to your NonStop system increases reliability.*

Well, it all depends. If sparing remains the same, then the number of failure modes increases; and reliability decreases. However, if the extra processors are used to increase sparing, then reliability can dramatically increase. However, since we don't generally change the design of the system when we add processors, this statement is typically false. Adding processors reduces reliability.

- *A 16-processor NonStop system has the reliability of a UNIX box.*

Again, it all depends. If you are distributing processes randomly, then this is true. However, if you are intelligent in the way you distribute processes, your system reliability will beat that of a UNIX system by an order of magnitude or more. So hopefully, your answer to this is false.

- *A NonStop processor is less reliable than a UNIX processor.*

The answer to this one is probably true, but so what? In order to build a truly fault-tolerant system, all components, both hardware and software, must be fail-fast so that corrupted data does not get propagated. To achieve this, NonStop processor boards use a pair of on-board lock-step processors which continually compare results. If there is a mismatch, the processor board shuts down immediately. Thus, the processor board can fail if either on-board processor fails, giving two failure modes instead of one for the UNIX processor. Given comparable component and manufacturing quality and comparable component count per processor, we would expect the NonStop processor board to fail twice as often as a UNIX processor. But this is a trivial price to pay for the ultimate fault tolerance provided. NonStop's fault tolerant architecture beats non-fault tolerant architecture by two 9s or so (100 times more reliable).

Here's a bonus question:

- *To double your capacity, should you upgrade your 4-processor S74K to an 8-processor S74K or to a 4-processor S86K?* (Assumes an S86K processor is twice as fast as an S74K processor, which is almost true).

A 4-processor S86K will not only be more reliable, but its response time also will be almost twice as fast.

Summary

We summarize the concepts presented above by considering how we might improve reliability. Our general availability relationships of Equations (1), (7), (9), and (10) state that[8]

$$A = \frac{MTBF}{MTBF + MTR} \approx 1 - \frac{MTR}{MTBF} \qquad (1)$$

$$A \approx 1 - f(1-a)^{s+1} \approx 1 - f\left(\frac{mtr}{mtbf}\right)^{s+1} \qquad (7), (9)$$

$$MTR = \frac{mtr}{s+1} \qquad (10)$$

From these equations, we can also determine that

$$MTBF \approx \frac{mtbf}{f(s+1)}\left(\frac{mtbf}{mtr}\right)^{s} \qquad (11)$$

These expressions relate system availability *A*, system mean time to repair, MTR, and system mean time to failure, MTBF, to four parameters on which we can get our hands – *f, s, mtbf,* and *mtr*:

[8] These equations are restatements of the Einhorn relationships. See Einhorn, S. J., "Reliability Prediction for Repairable Redundant Systems," Proceedings of the IEEE; February, 1963.

mtbf	Subsystem mean time to failure is out of our control. Not much we can do about that.
mtr	Reductions in subsystem repair time have an exponential impact on availability. In a one-spare system, cutting subsystem repair time in half provides a four-fold improvement in reliability. Cutting it by a factor of ten provides 100 times more reliability – two 9s on the availability scale. Consider a tighter service contract or, for larger users, on-site spares and on-site maintenance.
s	There's not much that we as users can do to increase sparing of NSK critical processes. HP would have to decide that the significant software development effort to do this is worthwhile. Until they take this step, there is not much sense in critical applications processes being written with sparing in excess of one.
f	Ah! We can control the number of failure modes. As we've shown, intelligent distribution of critical processes can reduce failure rates significantly, picking up one or two nines if we work at it.

How Far Should We Go?

From Figure 6, we see that we can achieve an availability of four 9s with our NonStop systems if we can keep the failure modes to five or less. Isn't this enough?

The Standish Group[9] has defined the following application categories and their required availability:

Class	9s
Non-critical	2
Task critical	3
Business critical	4
Mission critical	5
Safety critical	6

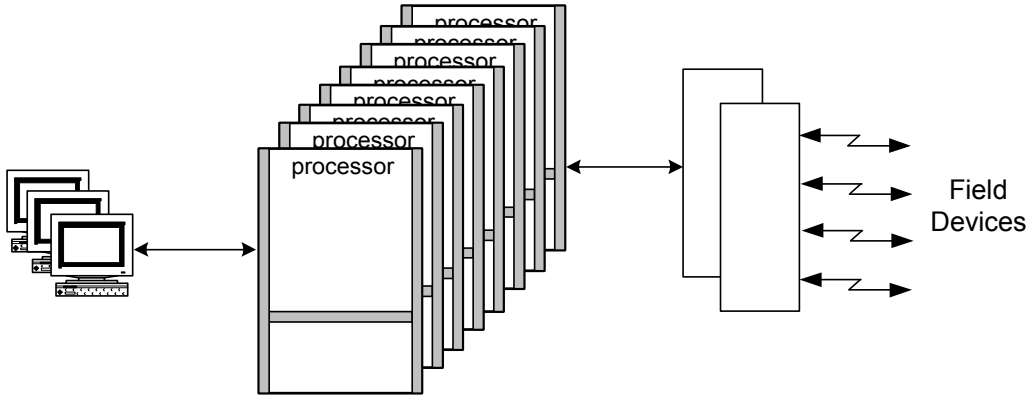
Availability Requirements
(The Standish Group)
Table 3

So if you have a need that you would characterize as mission critical or safety critical, you'd better mind your 9s.

[9] Standish Group; 2002.

A Case Study

Amtrak provided a real-life case study of these concepts with their real-time train control system for the busy Northeast corridor. Shown at a very high level in Figure 7, this system uses a NonStop system to monitor and control trains via a duplexed communication link to track-side devices (signals, switches, occupancy detectors). A redundant console system is used by the train dispatchers to monitor and direct train traffic.



System Availability = .999984 x .99952 x .9999992 = .99950
Failure Rate is .0005/.0002 = 2.5 times worse than required.
Solution: Reduce failure modes from 28 to 12 thrugh software configuration.

Amtrak Train Control System
Figure 7

A detailed analysis of the system availability (whose results are shown in Figure 7) predicted a system availability of .9995. Not bad! Unfortunately, the specifications for the system required an availability of .9998. The design had missed the reliability mark by a factor of 2.5. To make matters worse, this was a fixed price contract; and the system was not going to be accepted unless the availability requirement was met.

It was clear that the culprit was the 8-processor K-series system that had an availability of .99952 based on the worst case of 28 failure modes. However, a little algebra showed that if we reduced the failure modes to 12, we met the specification. Process allocation guidelines were put in place, and the system was accepted. By the way, it went into service in July, 1999, and hasn't failed yet.

This was an inexpensive fix to a potentially very expensive problem using the concepts discussed in this paper.

What's Next

Bear in mind that all we have really talked about so far are general availability concepts as applied to redundant hardware systems. The consideration of faults caused by software is a much more complex subject which is considered in Part 4 of this series of articles.

In Parts 2 and 3, we explore in more depth the availability considerations and advantages of replicating and splitting systems. In Part 4, we delve deeper into the impact of software and operational errors as well as environmental faults. In Part 5, we put all of this together to suggest a system architecture that can dramatically increase availability at little additional cost.