"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 2) - System Splitting

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

In Part 1 of this series[1], we talked about the dramatic increase that can be achieved in system availability by making a system redundant so that it is capable of tolerating one or more failures. This, of course, is the basis for NonStop systems, in which availabilities of four 9s are achievable (that is, the system will be up 99.99% of the time).

However, some applications require much more reliability than this. To achieve such high availability, a common technique is to provide a secondary system to back up the primary system. As we showed in Part 1, providing a single level of redundancy doubles the nines. Thus, redundant NonStop systems could potentially achieve availabilities in the order of eight 9s.

In some applications, it may be unnecessary to replicate the entire system. For instance, the system could be split into two independent half systems and provide the same enhanced availability at little additional cost.

This Part 2 of our series on availability considers some of the availability considerations associated with replicating and splitting systems.

The Availability Relation

As a quick review, the availability of a redundant system is given by

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

where it is assumed that the system can be restored to service as soon as a failed subsystem is repaired, and where

A	is the system availability (the portion of time that it is operational).
F	is the probability that the system will be down.
a	is the availability of a subsystem.
s	is the sparing level (i.e., s+1 subsystems must fail in order for the system to fail).
f	is the number of failure modes, or the number of ways that s+1 subsystem failures will cause a system outage.

[1] W. H. Highleyman, "Availability Part 1 – The 9s Game," The Connection, Volume 23, No. 6; November/December 2002.

The approximation in Equation (1) assumes that the subsystem failure probability (1-a) is very much less than one.

The rationale for this relationship is simple. (1-a) is the probability that a subsystem will fail. $(1-a)^{s+1}$ is the probability that (s+1) subsystems will fail. Since this can happen in f ways that will take the system down, then the probability that the system will be down is $f(1-a)^{s+1}$. System availability is one minus the probability that the system will be down.

The number of failure modes is an aspect that can be controlled by proper system design and configuration, as discussed in Part 1. The worst case is that in which any combination of s+1 subsystems will cause a system outage. In this case, f is the number of ways one can select (s+1) subsystems from the total of n subsystems making up the system:

$$f = \binom{n}{s+1} = \frac{n!}{(s+1)!(n-s-1)!}$$

For the NonStop case, s=1, and the number of failure modes f is

$$f = \frac{n(n-1)}{2} \tag{2}$$

where n is the number of processors in the system.

Full Replication

A very common way to protect the availability of a system is to fully replicate it, often in a different geographical region. The systems may be configured as a primary system and a backup system, in which case the primary system handles the full load and maintains a copy of its data base at the backup site via data replication (Figure 1a). This often is called an active/passive architecture. Alternatively, both systems may share the load using a variety of strategies described later. In this case, each will keep its companion's data base synchronized via data replication (Figure 1b). Here, provision is made for the surviving system to assume the entire load in the event of a system failure. This is called an active/active architecture.

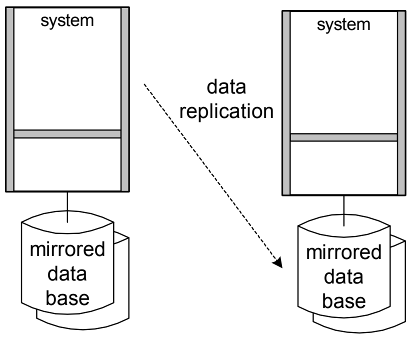
We know intuitively that system replication can dramatically improve reliability. But let us look at this quantitatively.

In the case of full replication, there is an overall system failure only if both systems fail. If the probability of failure of a single system is F_1, then the probability that both systems will be down, F, is

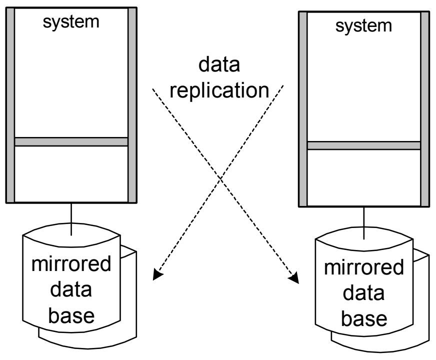
$$F = F_1^2 \tag{3}$$

The overall system availability *A* is the probability that both systems will not be down:

$$A = 1 - F_1^2 \tag{4}$$



a) Primary/Backup (active/passive)



b) Load Sharing (active/active)

Full Replication
Figure 1

An important implication of this relationship is shown in the following table, which calculates overall availability *A* as a function of single system availability A_1:

Single System Availability A_1	Single System Failure Probability $F_1 = 1-A_1$	Replicated System Availability $A = 1 - F_1^2$
.9	.1	.99
.99	.01	.9999
.999	.001	.999999

We see here that *full replication doubles the nines* (see Rule 2 in Part 1 of this series).

An insight into this power of replication can be gained by looking at our approximation to availability given by Equation (1). According to this relationship, the probability of failure for a single system, F_1, is approximately

$$F_1 \approx f(1-a)^{s+1}$$

According to Equation (3), the probability of failure for the replicated system, *F*, is then

$$F = F_1^2 \approx f^2(1-a)^{2(s+1)}$$

In effect, the impact of sparing has been doubled due to the doubling of the exponent for the (1-*a*) term. If it took two failures to cause a system failure in one of the systems, it will take *four* failures to cause a replicated system failure. True, the number of failure modes has increased (from *f* to f^2); but this is generally small compared to the effect of the extra sparing.

System Splitting

Simple Splitting

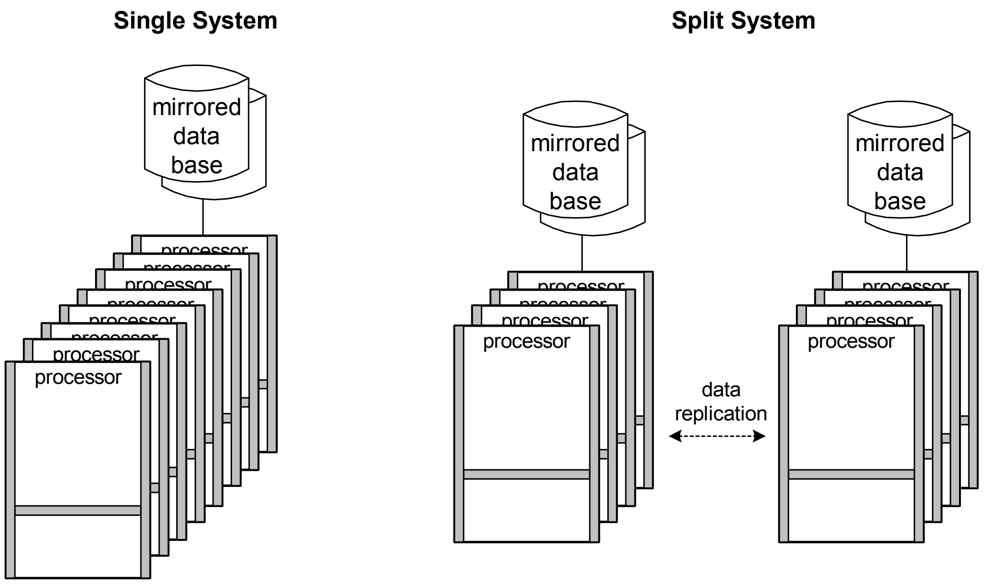
Rather than fully replicating a system at twice the cost of a single system, let us consider splitting a system into two equal half pieces, or nodes, perhaps geographically separated, as shown in Figure 2. A fully replicated system could either be an active/active or an active/passive system, as described in the previous section. However, the split system is necessarily an active/active system. Each node carries its share of the load during normal operation and keeps its companion's data bases in synchronism via data replication. However, should one node fail, the other will assume the full load. There are, of course, many considerations involved in implementing this sort of solution, such as load shedding in the event of reduced capacity and the switching of users to the surviving system. Assuming that these problems are solvable, let us look at the availability implications of this solution.

For purposes of illustration, let us assume that a system or a node is configured with one spare element (*s*=1) and that any dual failure will bring that system or node down. Then, for a system with *n* elements, the number of failure modes *f* is, from Equation (2),

$$f = \frac{n(n-1)}{2}$$

and the failure probability for that system is

$$F \approx \frac{n(n-1)}{2}(1-a)^2 \tag{5}$$



Splitting a System
Figure 2

Referring to Figure 2, if we were to split this system into two equal nodes of *n*/2 processors each, then each node would have a failure rate F_n of

$$F_n \approx \frac{\frac{n}{2}\left(\frac{n}{2}-1\right)}{2}(1-a)^2 \tag{6}$$

where F_n is the failure probability of a single node in the split system. The split system would maintain full capacity so long as both of its nodes were operating. The probability that either node is not operating, assuming that this probability is small, is

$$F_{100} \approx 2F_n \approx \frac{\frac{n}{2}(n-2)}{2}(1-a)^2 \tag{7}$$

where F_{100} denotes the probability of the split system failing to provide 100% capacity.

Let us define a reliability ratio that is the ratio of the failure rate, *F*, of the full system (Equation (5)) to that of the split system, F_{100} (Equation (7)), where failure means the failure to provide 100% capacity. Then

$$\frac{F}{F_{100}} \approx \frac{\frac{n(n-1)}{2}}{\frac{\frac{n}{2}(n-2)}{2}} = 2\frac{n-1}{n-2} > 2 \qquad (8)$$

Since this reliability ratio always is greater than 2, then we can conclude that the split system will provide full 100% capacity with a reliability at least twice as great as a single system providing the same capacity.

This is a direct result of the reduction in failure modes. For instance, an 8-processor system has 28 failure modes for a single spare configuration (8x7/2 from Equation (2)). If we split this 8-processor system into two 4-processor nodes, each of these nodes will have six failure modes (4x3/2). Since the failure of either node constitutes a failure to provide 100% capacity, then the total failure modes for the split system is 2x6, or 12. This is less than half of the 28 failure modes for the full system.

In addition, the split system will provide dramatically better availability if 50% capacity is acceptable. In this case, the availability of the system will be double the 9s of one of the nodes (which is already significantly better than the full unreplicated system).[2]

By way of example, consider an 8-processor system that is split into two 4-processor nodes. All system elements have an availability of .995, and each node is configured to have one spare. Their relative availabilities are as follows:

Configuration	**Availability of 100% Capacity**	**Availability of 50% Capacity**
Single System	.9993	---
Split System	.9997	.99999991

As we can see,

- the split system provides 100% capacity more reliably than the single system (by a failure improvement of more than a factor of 2).

- the split system provides some capacity with an availability dramatically better than that of the single system.

[2] Note: There are patents pending on the various methods for achieving the advantages of splitting.

Multiple Splitting

The concept of splitting a system into two nodes can be extended to splitting it into *k* nodes. For instance, one could consider splitting a 16-processor system into four geographically distributed 4-processor nodes.

Assuming that any dual processor failure will take a node down, then node failure rate F_n (given for $k=2$ in Equation (6)) is

$$F_n \approx \frac{\frac{n}{k}\left(\frac{n}{k}-1\right)}{2}(1-a)^2 \tag{9}$$

Since partial capacity can be lost in *k* ways (any one of the *k* nodes failing), then the probability that the system will not provide 100% capacity, F_{100}, is

$$F_{100} \approx kF_n = \frac{\frac{n}{k}(n-k)}{2}(1-a)^2 \tag{10}$$

The reliability ratio is then

$$\frac{F}{F_{100}} \approx \frac{\frac{n(n-1)}{2}}{\frac{\frac{n}{k}(n-k)}{2}} = k\frac{n-1}{n-k} > k \tag{11}$$

Thus,

- The split system will provide 100% capacity at least *k* times more reliably than the single system (in the sense that its failure probability is reduced by a factor of more than *k* compared to the single system).

- If it does suffer an outage, a split system loses only 1/*k* of its capacity (for instance, 25% for a four-way split).

Rules 1 through 7 were given in Part 1 of this series. Here are some additional ones:

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/k-1 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.*

Impact on Mean Time Before Failure

In Part 1, we showed that mean time before failure, MTBF, mean time to repair, MTR, and the system failure probability, F, were approximately related as follows:

$$\text{MTBF} \approx \frac{\text{MTR}}{\text{F}}$$

That is, MTBF is inversely proportional to the system failure rate. If we cut the failure rate in half, we double the MTBF, all other things remaining equal.

Let us consider separating a 16-processor system into four 4-processor nodes. From Equation (11), we see that this increases our reliability (decreases our failure probability) by a factor of 5 (i.e., 4 x 15/12). Thus, if the single system had a mean time before failure (MTBF) of 10 years, we have just extended that to 50 years. Furthermore, in the event of an outage, only 25% of the system capacity is lost. The average time before losing more than 25% capacity is measured in centuries.

In the extreme, splitting a 16-processor system into eight 2-processor nodes will provide a 15-fold availability advantage. Our 10-year system MTBF has now been extended to 150 years, and an outage costs us just 12.5% of capacity.

Elimination of Planned Downtime

In addition to the additional fault tolerance achieved by splitting a system into independent nodes, one now has the ability to make hardware and software upgrades one node at a time, thus eliminating planned downtime. This is a cornerstone of HP's NonStop Indestructible Scalable Computing initiative[3].

Replication of Data

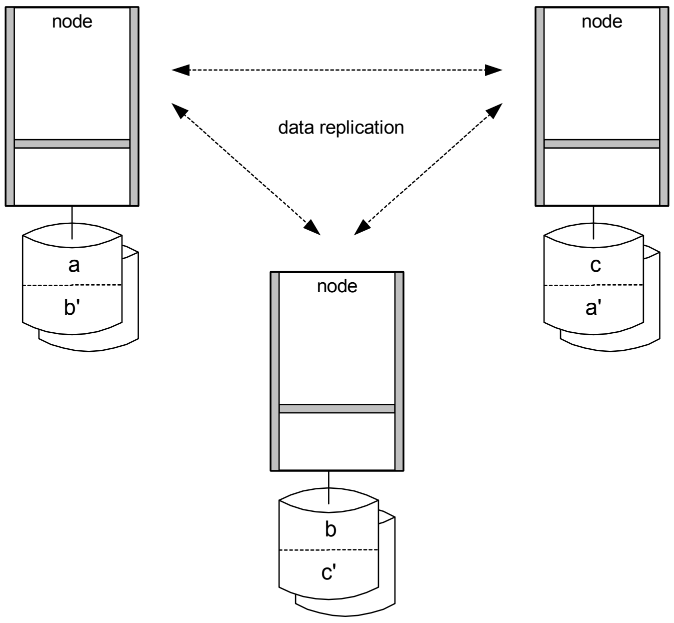
When we split a system into two or more nodes, we need to replicate the critical components of the data base if we are to achieve the benefits of system splitting. Should we split by a factor greater than two – say by a factor of *k* – we do not have to replicate the data base *k* times. We do have to make sure, though, that it is replicated at least once so that the entire data base is available even in the presence of a node failure. For instance, the data base could be partitioned so that each node in the network has 2/*k* of the data base connected to it, as shown in Figure 3. Each of these replicated database partitions must be kept in synchronization via some sort of data replication.

[3] Wendy Bartlett, "Indestructible Scalable Computing," ITUG Summit presentations.

There are several options as well as some serious issues involved with the replication of data across systems.

- The simplest architecture is the fully replicated primary/backup (or active/passive) system described earlier (see Figure 1a). In this case, the backup data base is kept synchronized with that of the primary; but there is no update activity to the data base by the backup system (it may, however, be used for read only operations such as query, reporting, etc.). Here, the only issue is replication latency. Should the primary go down, any updates that are in the replication pipeline will be lost.

- Perhaps the data base can be partitioned so that each partition is "owned" by a node, as shown in Figure 3. Only that node may make updates to its partition. These updates then are replicated to other nodes for information purposes only. Thus, even in the event of a node failure, all current data is available to all users. If ownership of a partition moves to a surviving node, then full functionality is maintained even in the event of a node failure. Again, the primary data replication concern is lost updates due to replication latency.



Database Partitioning Across Nodes
Figure 3

- In the general case, any user at any node can update any data item. This is the active/active configuration shown in Figure 1b. There are several important data replication issues with which to contend when using this approach:

a) There must be a mechanism to guard against *ping-ponging* or the replication of an item back to the source of that item.[4]

b) To the extent that there is latency in the data replication process, there is a chance that two different updates will be made to the same data item at the same time and that these conflicting updates will be replicated across all systems. The database contamination caused by these updates must first be detected, and then either be corrected manually or by automatic conflict resolution via business rules.

c) If the distributed data bases are tightly synchronized to eliminate latency so as to ensure that there will be no data conflicts, then system performance may suffer significantly.[5] Synchronous replication performance will be explored further in Part 3 of this series.

In addition, the data replication facility must be aware of the current network configuration so that it can replicate around node and network faults. Furthermore, provision must be made to switch users from a failed node to an operational node.

Must We Replicate the Data Base?

We have talked about splitting a system into two or more parts at little additional cost except for a replicate of the data base. However, in many large systems, the cost of the disk subsystems can be 70% to 90% of the system cost. It doesn't seem that splitting a system and replicating the data base is the cost effective solution that we suggested.

The solution to this quandary is to note that, up until now, we have assumed that the database replicates are mirrored (see Figures 1 through 3). Is this really necessary? If we are replicating data across the nodes anyway, why does it need to be replicated once again by mirroring the disks at each node?

There is a strong argument that suggests that the database replicates in a split system do not need to be mirrored. It goes as follows:

In 1985, Jim Gray[6] pointed out that the MTBF of disk drives was about 10,000 hours. By 1988, this had grown to 75,000 hours. 1992 brought 175,000 hour MTBFs, and by 1996 disk drive MTBFs were at 500,000 hours[7]. Surprisingly, not only disk speed and capacity but also disk availability have been following Moore's Law.

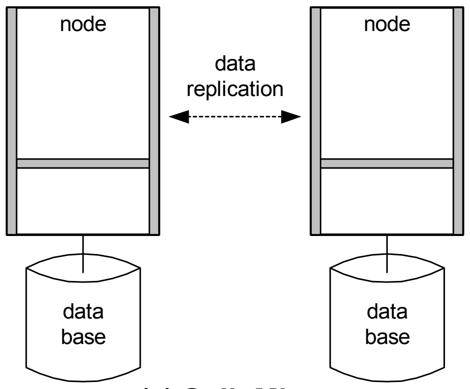
[4] See United States Patent 6,122,630, "Bi-directional Database Replication Scheme for Controlling Ping-Ponging," Strickler et al; Sept. 19, 2000.
[5] There are patents pending on various methods for conflict identification, resolution and avoidance.
[6] Gray, J., "Why do Computers Fail and What Can Be Done About It," Tandem Technical Report 85-7; June, 1985.
[7] Wong, B. L., "Configuration and Capacity Planning for Solaris Servers," Prentice-Hall.

Let us conservatively use a disk MTBF of 100,000 hours. Assuming a leisurely 24-hour repair time, a mirrored disk pair will have an availability of .99999994, or over seven 9s. In fact, since a dual disk failure has a mean time to repair of one-half the disk unit mean time to repair, or 12 hours (see "Part 1 – The 9s Game"), this means that the MTBF for a mirrored disk pair is almost 240 centuries!

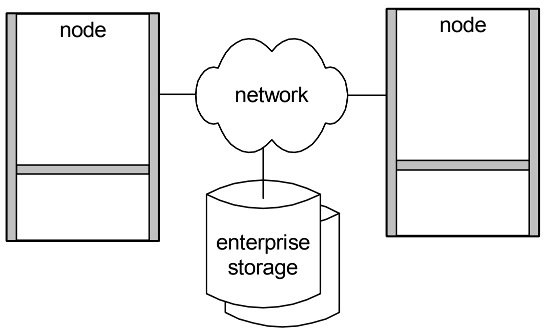
Since the MTBF of a NonStop system is in the order of 10 years, it seems reasonable that database replicates need not be mirrored. It is important only to make sure that there are at least two copies of the data independently accessible somewhere in the network.

Of course, if we have *d* mirrored disk volumes, then the average time between failures is 240/*d* centuries. However, unless *d* is very large, the replicated disk farm will still have an MTBF much longer than a processing node.

This leads to two strategies for providing replicated data to all nodes in a split system. One strategy is to split the mirrors (Figure 4a). In essence, the data base on each node is not mirrored. Mirroring is provided by having a single copy of the data base on each of two nodes and by keeping them synchronized via data replication.



(a) Split Mirrors



(b) Enterprise Storage

Split System Networked Data Base
Figure 4

The other is to provide mirrored enterprise storage accessible to all nodes (Figure 4b). Note that split mirrors satisfy the needs for disaster recovery since the database replicates can be geographically separate. Enterprise storage does not satisfy this requirement unless the mirrors can be geographically separate and accessible via different network paths.

Either of these solutions leads us to the desired result – increased availability via system splitting with little if any additional cost.

By the way, you might ask how a mechanical dual disk system can be so much more reliable than dual processors? The answer is simple – no software and no humans. More about this in Part 4.

Summary

No advantage comes for free. The significant availability advantages brought about by system splitting have their potential costs. In principle, we can split a system into two or more nodes with little additional hardware cost, can add data replication to keep them in sync, and can gain dramatically improved reliability. As we have seen,

- Replicating a system doubles its 9s.
- Splitting a system into *k* nodes reduces the chance of not providing 100% capacity by at least a factor of *k*.
- Splitting a system into two or more nodes dramatically increases the availability of at least some processing capacity. In fact, if a system is split into *k* nodes, 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.
- Splitting a system allows us to eliminate planned outages.

However:

- Data replication latency could cause lost updates in the event of a failure.
- Data replication latency also could cause database corruption via conflicting updates if all nodes are allowed to actively process inputs independently without concern for update activity at other nodes.
- Tight database integration to eliminate latency and data collisions may have a negative performance impact.
- Software licensing costs may increase.
- We will see in "Part 4 – The Facts of Life" that the chance of failover faults has the potential for eroding the availability advantages provided by system splitting.

The good news is that solutions are emerging for these problems, as evidenced by patent activity, published papers, and new product enhancements. Replicating full or partial systems using data replication as the synchronizing mechanism is a powerful method for significantly improving system availability in the presence of system faults, natural disasters, and human errors.

What's Next

In this Part 2 of our series on availability, we have talked about replicating and splitting systems. For active/active configurations, we are left with the severe problem of data collisions. In Part 3 we look at synchronous replication, a method for avoiding these collisions.

In Part 4, we question the assumption that a system can be returned to service as soon as a failed subsystem is repaired; and we explore the impact of failover faults and system recovery on availability. Finally, in Part 5, we put all of this together to suggest an ideal system from an availability viewpoint.