"Achieving Century Uptimes"
An Informational Series on Enterprise Computing

As Seen in *The Connection*, An ITUG Publication
December 2006 – Present

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.

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

Achieving Century Uptimes

Part 1: Survivable Systems for Enterprise Computing

November/December, 2006

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

Achieving 100% Uptime

It is becoming trite to say that today's global economy and the enterprise's absolute reliance on computing technology mandates that critical IT services are never impaired – ever. However trite it may be, this statement is true. As Anton Lessing, CTO of BankServ said, "I am not interested in 9s. I am interested only in 100% uptime."

How can we achieve 100% uptime? Clearly, we can never know that we have achieved it – something that we did not anticipate can always happen decades down the line. However, we can come arbitrarily close to this goal.

Of course, "arbitrarily close" translates to "arbitrarily expensive." There is a limit to what we are willing to spend for extreme availability. Our willingness to spend is often related to the cost of downtime. Therefore, there is an omnipresent compromise between availability and cost. Actually, to be more accurate, one must compromise between availability, cost, and performance. Increasing availability will almost certainly have a negative impact on either cost or performance, perhaps even on both.

One technique in use today for approaching this goal is the use of active/active architectures. This is what this new series is all about. We will be describing these architectures, their advantages, and the issues associated with them.

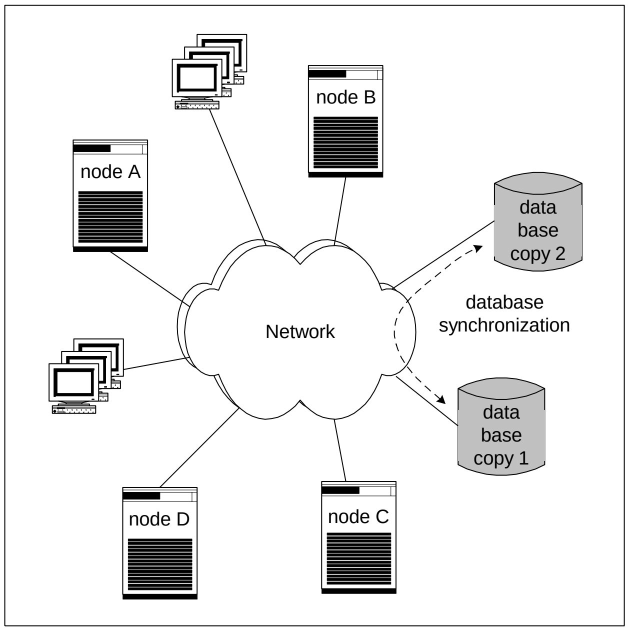
What is Active/Active?

It is a fundamental fact that any system can and will fail at some point. The secret to approaching 100% uptime is to *let it fail, but fix it fast*. This is the premise behind active/active. If a service outage is too short for users to notice, then it will not be perceived as a service outage. In effect, service availability has been maintained.

From a high level perspective, active/active architectures accomplish fast recovery by distributing the user base over multiple independent and geographically distributed processing nodes. Should a node fail, the users of that node are switched to one or more surviving nodes in the application network. This switchover can often be done in subseconds – taking no more time than the resubmission of a failed transaction, an otherwise common occurrence.

Active/Active Architectures

An overview of the architecture of an active/active system is shown in Figure 1. *An active/active system is a network of independent processing nodes, each having access to a common replicated database such that all nodes can participate in a common application.*



Active/Active Architecture
Figure 1

The application network comprises two or more processing nodes connected to a redundant communications network. Any user can be connected to any node over this network. The nodes all have access to two or more copies of the database. The database copies may be connected directly to the network as shown in Figure 1, or they may each be directly attached to one of the nodes.

The database copies are kept in synchronism by ensuring that any change to one copy is immediately propagated to the other copies. There are many techniques for doing this, such as network transactions or data replication. These techniques are described later in this article and in more depth in a future part of this series.

Should a node or its attached database fail, the users connected to that node have lost their services. In this case, they are switched to another node or are distributed across multiple surviving nodes to immediately restore service. Users at other nodes are unaffected. User switching is also discussed in much detail in a future part of this series.

Providing that the nodes and database copies are geographically distributed, active/active systems provide disaster tolerance. Should a disaster take out a node or a database site, there are others in the network to take its place.

Active/Active and Business Continuity Planning

Hardware and software faults and disasters are only some of the things that can take down a system. Though an active/active architecture can effectively remove these failure causes from the availability equation, failures can also be caused by several other factors, such as operator error, network faults, and environmental failures (power, air conditioning, and so forth).

Most failures caused by these faults will only affect one node, and service availability is provided by switching users to surviving nodes. Thus, uptime is not affected.

However, there are worst case scenarios that could take down the entire distributed system. What is the probability that a second node will fail before the first node has been repaired, thus leaving the application network with insufficient capacity to satisfactorily service the users? Are the nodes and database copies distributed widely enough so that the equivalent of the 2004 Northeast blackout in North America will not take down all nodes? Are there operator procedures that have a network-wide affect? What is the chance that a massive network failure could disconnect all users from their nodes? Can a software bug propagate through the network? (Do you remember the software bug that propagated through the ATT network in 1990 and took the entire network down, thus denying telephone service to subscribers for hours? A decade earlier, a similar bug collapsed the ARPAnet.)

Consequently, there must be in place a plan directing the efforts of all concerned so that they can recover from such a disastrous occurrence. Simply having an extremely available system is not enough. One must plan how to recover from a highly unlikely, but not impossible, total system failure.

This plan is, of course, the Business Continuity Plan; and it is the subject of much literature today. Without a good Business Continuity Plan, one cannot say that he has approached 100% uptime because otherwise, the one-time failure of the system could have disastrous consequences.

Though needed for an active/active system, Business Continuity Plans are generally associated with "highly available" systems in which occasional outages are expected. This leads to an important distinction between these plans and active/active systems. *A Business Continuity Plan is used to <u>recover</u> from the effects of an outage. An active/active architecture is used to <u>avoid</u> the effects of an outage.*

Synchronization

There are many ways to keep two or more database copies in synchronization. These include network transactions, asynchronous data replication, and synchronous replication.

Whatever the technique, one result is mandatory. The database copies must always maintain referential integrity so that each can be used actively by any application copy. This typically means that transactions initiated by a node be committed in the same order at any of the database copies. In some cases, this requirement may extend to the updates within a transaction. This is known as preserving the “natural flow” of all updates.

Network Transactions

Using network transactions, the scope of a transaction includes all database copies. This results in each copy of a data item across the network being locked before any copy is updated. As a result, all database copies are kept in exact synchronism.

One problem with this technique is that each lock request and each update must individually flow across the network and a completion response received. In widely dispersed systems, such a round trip could take tens of milliseconds, and application performance can be seriously affected. This is a clear example of the compromise between availability and performance.

Asynchronous Replication

An asynchronous replication engine extracts changes made to its source database from some sort of a Change Queue (such as an audit trail) and sends these changes to a target database. This replication is done “under the covers” with respect to the application, which is therefore unaware of the replication activity. Therefore, there is no performance impact to the application.

However, there is a delay between the time that a change is applied to the source database and the time that it is subsequently applied to the target database. This delay is known as *replication latency* and typically ranges from hundreds of milliseconds to a few seconds. As a consequence, should a node fail, there will likely be transactions that, having been applied to the source database, are still in the replication pipeline at the time of the failure. These transactions never make it to the target database and in effect are lost.

Another issue with asynchronous replication is data collisions. If a data item is updated on each of two database copies within the replication latency time, each will be replicated to the other database copy and will overwrite the original update to that copy. The two database copies are now different, and both are wrong. Techniques for avoiding data collisions or for detecting and resolving them are discussed in a later part in this series.

Synchronous Replication

With synchronous replication, changes are replicated to the target database via asynchronous replication but are held there and used only to lock the affected data items. When the source node is ready to commit the transaction, it checks with all database copies to ensure that they have been able to obtain locks on all of the affected data items. If this is the case, the source node instructs all database copies to apply the updates and to release the locks. Otherwise, all copies are instructed to abort the transaction.

Synchronous replication, like network transactions, guarantees that all database copies will be in exact synchronism (as opposed to asynchronous replication, which keeps the database copies in near-synchronism because of replication latency). Thus, no transactions are lost as the result of a failure; and data collisions cannot occur.

In this case, the application is delayed as it waits for the commit to complete across the network. This is called *application latency*. However, this delay compares to network transaction delays which must wait for each update as well as the commit to complete across the network. As a consequence, synchronous replication is generally more efficient if database copies are widely distributed or if transactions are large. Network transactions may be more efficient for collocated database copies and short transactions.

Other Advantages of Active/Active

There are many other advantages that an active/active architecture brings:

- A node can be upgraded by simply switching its users to other nodes. The node then can be brought down and its hardware, operating system, database, or application upgraded and tested. At this point, the node can be returned to service by returning its users to it. This technique effectively eliminates planned downtime.
- As compared to an active/backup configuration in which the backup system is not processing transactions, an active/active configuration can be distributed to provide data locality. Users can be connected to their nearest respective node, thus improving performance.
- All purchased capacity is available for use. There is no idle backup system sitting around. Therefore, in a multinode active/active architecture, less capacity may need to be purchased. For instance, in a five-node configuration, if each node can carry 25% of the load, full capacity is provided even in the event of a node failure. However, only 125% of required capacity must be purchased, rather than 200% for an active/backup system.
- Capacity easily can be added by installing a new node and then switching some users to the new node.

- The load across the application network can be rebalanced by moving users from a heavily loaded node to a lightly loaded node.
- Failover testing can be risk-free and not require any user downtime. In an active/backup system, it may take hours to fail over, during which time the users are denied service. The same downtime impact occurs when users are switched back to the primary node following failover testing. Furthermore, what if the backup node turns out to be non-operational? In an active/active system, it is known that the other nodes are working; and failover takes seconds at most.

Active/Active Issues

There are several important active/active system issues that must be understood and resolved. Many of these are dealt with in later parts of this series. They include:

- How will user switching be handled? Can it truly be done automatically?
- Is there a chance of data collisions? If so, how will they be handled?
- Are lost transactions following a node failure acceptable? To what extent? Can they be recovered?
- Can the applications be run in an active/active environment? Do they require modification?
- What performance impact can be tolerated when implementing an active/active architecture?
- What additional cost can be tolerated? This may include hardware, software licenses, networking, people, and sites.

What's Next?

In the coming parts of this series, we will talk about many of the issues raised in this introduction. They include:

- active/active topologies.
- reliable networks and user switching.
- distributed databases.
- recovering from node failures.
- application issues.
- data collision avoidance versus detection and resolution.
- upgrading with zero application downtime.
- total cost of ownership.
- other benefits of multinode active/active architectures.
- case studies.