“Achieving Century Uptimes” An Informational Series on Enterprise Computing

As Seen in *The Connection*, A Connect 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 22: Fast Failover with Active/Active Systems (1 of 2)

May/June 2010

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

Continuous availability means that we will never be down. But systems fail. Networks fail. So how can we achieve continuous availability? The secret is to *let it fail, but fix it fast*. If user services are restored following a failure so quickly that no one notices, in effect no failure has occurred. Continuous availability has been achieved.

We relax this criterion a bit by replacing the phrase “no one notices” with the phrase “no one is inconvenienced.” Typically, this means recovery times in the order of a few seconds. If normal transaction response time is two seconds, and if we can recover in one second, we have achieved this goal. A system failure has no worse impact on users than a busy system, and users will probably not perceive the failure as downtime.

Active/Active Networks

Active/active networks[1] allow us to achieve the goal of continuous availability. As shown in Figure 1, an active/active network comprises multiple geographically-distributed processing nodes using geographically-distributed consistent copies of the application database. Data replication keeps the database copies in synchronism.

The key to fast recovery is that users can direct their transactions to any processing node in the application network. Therefore, should there be a node failure or a network failure, all that users need to do is to redirect their traffic from the affected node to a surviving node in the network. If users can accomplish redirection in subseconds or seconds, they have achieved continuous availability.

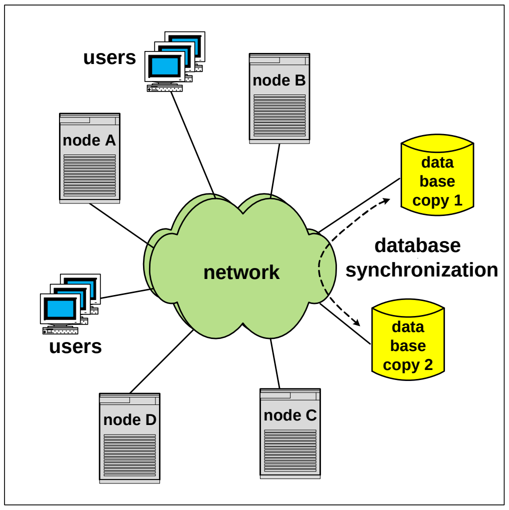

Figure 1: An Active/Active Network

But how can user traffic be rerouted so quickly? The answer to this question is the purpose of this article series.

Fault Recovery

In general, the recovery from a fault requires the following steps:

[1] Achieving Century Uptimes Part 1: Survivable Systems for Enterprise Computing, *The Connection*; November/December 2006.

- *Detect* that a fault has occurred.
- *Determine* the cause of the fault.
- *Decide* the course of action. Is it better to try to recover the failed system or to fail over to a backup system?
- *Approve* the course of action (management approval is often required).
- *Invoke* the recovery action:
    - If recovery:
        - *Start* the applications.
        - *Test* the system.
        - *Restore* system to service
    - If failover:
        - *Rebuild* the database
        - *Start* the applications
        - *Reconfigure* the network.
        - *Test* the system.
        - *Restore* the system to service

For these reasons, failover can often take many hours if a company is using an active/backup configuration for disaster recovery.

If the fault is in an active/active network, the operations staff can bypass most of this lengthy procedure. Of course, the system must first detect the fault. Given that, the course of action is simple – reroute traffic to surviving nodes. There is no need for a lengthy decision process or for management approval – just do it! After all, the application is up and running on the other nodes and it is in a known-working state. And if the system can reroute traffic automatically with no manual intervention required, continuous availability can be achieved.

User Redirection

There are three basic ways that user traffic can be redirected to surviving nodes:

- Client redirection.
- Network redirection.
- Server redirection.

In this article, we discuss client redirection and network redirection. A follow-up article will focus on server redirection. We will find that as the redirection intelligence moves from the client to the network to the server, redirection becomes more complex and, in some cases, slower. However, it is not always possible to add redirection intelligence to clients (such as ATMs and browsers). Therefore, we must be aware of a range of redirection options.

Client Redirection

Client redirection is perhaps the least complex and fastest way to redirect user traffic. However, it depends upon the client being intelligent enough to know that it has a primary node and a backup node and that it understands how to switch between them. This may not always be possible. For instance, standard browsers and ATMs are examples of clients that may not lend themselves to adding such intelligence.

Given the appropriate intelligence, a client will have knowledge of the nodes available to it based on their IP addresses or URLs. In Figure 2, a client is shown as having access to two processing nodes. It sends transactions to its primary node via IP address IP1. If it needs to do so, it can reroute its transactions to its backup node, which is listening on IP address IP2.

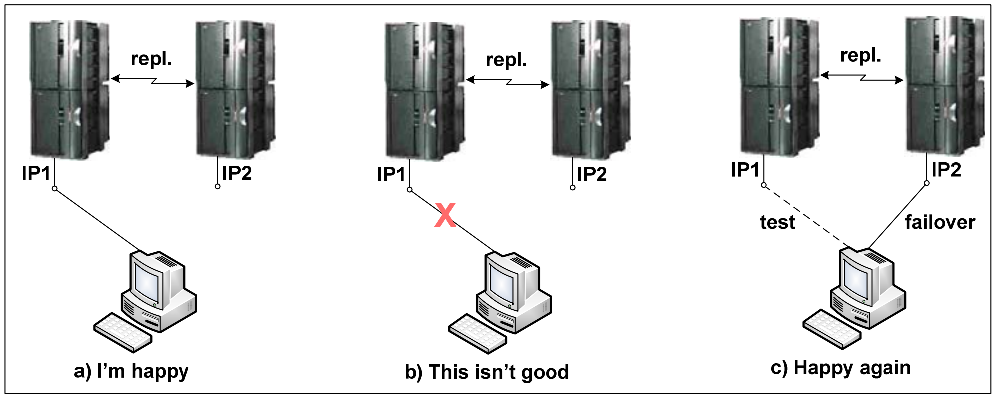

Figure 2: Client Redirection

The client normally sends its transactions to its primary node (IP1 - Figure 2a). However, should it not receive a response from this node (Figure 2b), it resends the failed transaction and all further transactions to its backup node (IP2 - Figure 2c). While the client is using the backup node, it periodically sends test transactions to its primary node to determine when that node is back in service. When the client begins to receive responses to its test messages, it can switch back to its primary node and resume sending transactions to it.

In this way, failover takes the same amount of time that any resubmission of an aborted transaction would take. Failover time is primarily the transaction timeout. Furthermore, the user is unaware of the restoration of the failed node. Restoration simply entails the submission of the next transaction to the restored primary node instead of to the backup node.

The client can retry a transaction a specified number of times before failing over, or it can do so on the first transaction failure. If the failure is transient in nature, the client can quickly determine this condition and can switch back to its primary node.

If the active/active network comprises multiple nodes, the client can be given an ordered list of nodes to which it can fail over. In this way, multiple node failures are easily handled providing the surviving nodes can handle the increased transaction load.

An alternate strategy is for the client to round-robin its transactions to each node, one node after the other. If a node fails to respond, the client removes it from the round-robin list and instead sends the nonresponsive node periodic test transactions. When the node once again becomes responsive, the client returns it to the round-robin list.

The client should also be able to respond to an external command to change its current processing node and perhaps its failover list. This capability allows all clients to move from a node that is about to be taken down for maintenance. It also allows the operations staff to easily redistribute workload to accommodate short-term peaks or long-term trends.

Network Redirection

With network redirection, the responsibility for redirecting traffic from a failed node is the responsibility of the network. Figure 3a shows a typical configuration for an active/active network. It comprises two nodes, Node A and Node B, which are geographically separated. A replication engine keeps the nodal databases in synchronism by sending changes made to each database to the other database over a replication communication link.

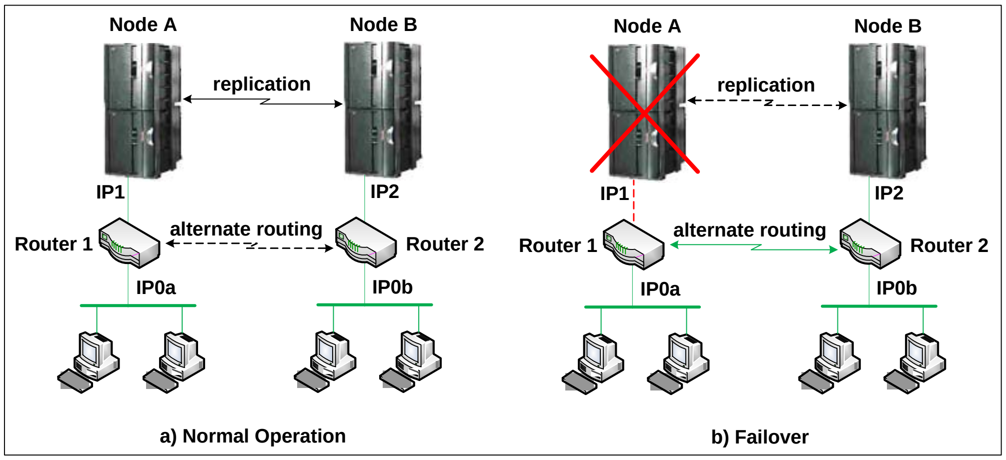

Figure 3: Network Redirection

In this example configuration, each node services its own community of users accessing the node over a local LAN. However, rather than being directly connected, the LAN traffic flows through a router that normally routes all traffic to the local node. As we shall see, it is the router that allows the network to redirect traffic.

The clients are not aware, nor do they care, to which node they are connected. They are only aware of a virtual IP address assigned to them for the submission of transactions. Clients at Node A are assigned virtual address IP0a, and clients at Node B are assigned virtual address IP0b. Router 1 normally routes all client traffic originating from IP address IP0a to Node A's physical IP address, which is IP1. Likewise, Router 2 normally routes all IP0b traffic to Node B at IP address IP2.

In order for network redirection to work in this configuration, the routers must have the intelligence to be able to route around an outage – a capability inherent in most routers today. These routers maintain routing tables that designate alternate routes to use if a primary route fails. In the example of Figure 3, a communication link connecting Routers 1 and 2 provides the alternate route for each router.

Figure 3b illustrates the traffic flow should Node A fail. Router 1 discovers that it can no longer send traffic to Node A. Consulting its routing table, it determines that the alternate route is the routing communication link to Router 2. It therefore begins sending further traffic for the IP0a clients to Router 2, which will forward that traffic to Node B. Node B is now responding to all transactions from the entire user community. The router has accomplished failover in the time that it takes it to determine that one of its links is down.

When Node A is ready to be returned to service, a command must be sent to Router 1, directing it to once again route IP0a traffic to Node A.

Not shown in Figure 3 is the network redundancy that should exist for truly continuous availability. Every path in the network should be redundant to ensure that the network can reroute traffic around any network fault. Network redundancy includes duplexed replication and alternate routing links, duplexed routers, duplexed LANs, and duplexed nodal LAN interfaces. Of course, in actual practice, a company may decide that it is willing to accept the unlikely failure of a highly-reliable component in order to save costs or to avoid technical complexities. For instance, a single LAN may be used at each site if the clients are incapable of dual-LAN operation.

Connection and Session Loss

A common problem with any strategy for user redirection, whether redirection is the responsibility of the client, the network, or the server (server redirection is discussed in our next article), is that the node to which the user traffic is being redirected knows nothing about the original session that the client had established with its server. The client loses both its session with the application and its connection to the server node that it had been using.

If no other provision is made, the network will notify the user of the connection and session losses, and the user will have to reestablish them. If this is the case, continuous availability has been compromised because the manual reestablishment time has "inconvenienced" the user. However, if the client has the intelligence to automatically detect this condition and can automatically reestablish the connection and session, or if it can maintain dual sessions with its primary and backup nodes, then the user will be unaware of the problem (except perhaps for a short time delay if automatic connection/session reestablishment is needed). In this case, continuous availability has been achieved.

Nodal Capacity

This article has inherently assumed that the processing nodes and routers in an active/active network have been sized to properly handle the increased load in the event of a node or router failure (or multiple failures, if so desired). Otherwise, a node or router failure can overload other nodes or routers, taking them down and causing a cascade of failures that takes down the entire network. The press is rife with stories of just this happening.[2]

What's Next?

Fast and reliable recovery from a node or network failure in an active/active network is conceptually simple. All that has to be done is to redirect traffic generated by affected users to surviving nodes in the network. Failover is reliable since it is known that the surviving nodes are

[2] Google Troubles – A Case Study in Cloud Computing, *Availability Digest*; October 2009.

fully operational. After all, they are currently handling transactions for the same applications being used by the affected users. Failover can also be fast – subseconds to seconds – so fast that users may not even notice that there has been an outage. However, the devil is in the details.

We have discussed in this article how to rapidly recover from a node failure in an active/active network using client redirection and network redirection. Failover using these techniques can be automatic and very rapid. Transaction timeouts for client redirection and router timeouts for network redirection are the determinants of failover times. Note that these techniques do not affect the applications. They are implemented in the active/active infrastructure itself.

In our next article, we look at user redirection under the control of the processing nodes in the network, what we call *server redirection*. Though in common use, we will see that server redirection can be more complex and may take longer, even involving some manual intervention in some cases.

One problem not addressed here that will be discussed in our next article is the loss of the replication network. In this case, unless something is done, each node will continue to process transactions independently; and the database copies will begin to diverge. If this is unacceptable, provisions must be in place to take one of the nodes offline until replication can be restored. It is for this reason that the replication network should be redundant with automatic failover.