HIGH-AVAILABILITY

Application Persistence

Introduction

Highly available solutions remove single points of failure (SPOFs) and this creates an environment in which users can have near continuous availability. Putting aside essential maintenance, like software upgrades, there are a number of possible periods of downtime in the event of a component failure. The extent of the downtime is dependant upon a number of factors but to a large extent the architecture of the application(s) is overriding factor.

This paper outlines the major architectures and gives some examples using popular applications to illustrate the principles.

Server Application Models

Broadly, there are two server application models, those that are state-full and those that are stateless;

State-full Applications

Any server application that accepts a packet and processes it based on prior traffic is maintaining state. For example using a persistent connection, often making use of transmission control protocol (TCP) is state-full. Technically the use of TCP makes an application implicitly state-full, however, it is possible for applications that use TCP to use brief 'sessions' to be considered largely state-less, when they do not track prior 'sessions'.

Some applications re-transmit the state with each request. While the application as a whole is state-full the server component is state-less and therefore we consider this a state-less server application model.

Examples of state-full applications include; telnet & ftp.

The server components of these applications run on a machine and accept connections *in*. For each connection the machine creates a process or thread to track the connection and provide an environment for the user. The 'current working directory' and sub-processes etc. are tracked in this environment, a shell, and should the machine fail then this shell would have to be rebuilt before the user could continue with their work. Some modern ftp clients are intelligent enough to track progress and directories so they can reconnect if the connection is lost. However, there is no way to rebuild the state of a telnet connection.

State-full applications can not usually be failed over in a cluster environment without the user being aware of a interruption of service; this is normally limited to a broken session requiring a re-connection. Some applications permit synchronisation between server, so that sessions can be moved seemlessly.

State-less Applications

Some client applications make requests that include all the necessary data in the request to link back to prior requests, these clients may operate with a state-less server. Some applications are very simple and there is no need to maintain information about prior requests.

Examples of state-less applications include; Apache & Remedy.

Apache is a web server which can provide static content or provide data from a back-end database. It is possible to build Apache servers that don't operate in a state-less fashion but this is generally considered bad practice; an example of such bad practice is to build a shopping cart with a cookie stored on the web server rather than in the database, if the server should fail then the basket will be emptied - it is now common practice to avoid this pitfall and most Apache implementations are 'clean' and state-less.

Remedy's AR System uses a client that maintains user state and provides it to the server with each request. The AR System server uses a back-end database and the server component can be restarted or moved to another machine without causing a user session restart. Further, because the AR System server component authenticates and maintains a session to the database the user does not suffer dropped connections if the database is restarted or moved; the AR System server simply re-connects to the database. Clearly if either the server or the database server are in the process of restarting then the user will suffer and errors may be reported but once the servers are back the user experience will not be affected.

In tests completed by customers using AR System the servers were repeatedly crashed intentionally and the users consulted about their perception of reliability during the testing period. No users reported an adverse perception of the service but if the application servers did not operate in a state-less way then the user experience would have been very different.

Server Application Examples

NFS (Network File System)

NFS provides access to file systems over a network connection. The NFS server is configured to permit or deny access by various users or systems, in addition to the standard file permissions. The client maintains the state of each connection and with each request to a server the current position in the file (inode) is sent from the client to the server.

Fundamentally NFS is designed to be a state-less application at the server, the client must maintain and supply state information with every request. However, the server must maintain lock information as this can not be distributed among the clients.

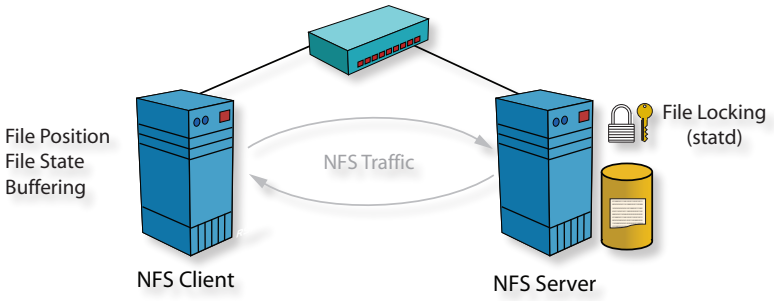

Figure 1.
NFS Client-Server

On most UNIX machines there is a 'statd' (or rpc.statd) process which writes active client file locking operations to disk. If the server is restarted then this information can be used to rebuild the current file locking status. Some implementations, notably Sun Solaris, allows statd to be configured to operate successfully in a clustered environment, by writing lock information to a select-able disk. The Solaris statd options are not provided in the standard user documentation but the full documentation for statd can be found here on our web site.

Oracle

Users are rarely exposed to Oracle itself but instead use applications that rely on Oracle to act as the back-end. There are a number of ways to build a highly available Oracle installation but the options available depend on the way the application is written. For instance an application must be specifically written to be 'RAC' aware to take advantage of the facilities that RAC can provide. Deploying an application in a RAC environment, which has not been written to support RAC is both costly and complex and the end result is a system that is less reliable because of that added complexity.

Well written traditional server based applications make a connection to Oracle and can remake that connection if the connection should fail. In addition the application tracks what transactions are committed, so that if the connection breaks before a transaction is completed then the application can re-submit a change. However, this intelligence must be built into the application by the application vendor.

Transparent Application Failover (TAF) is a feature provided by Oracle that can be used by the application vendor during coding of the application. TAF automates the reconnection procedures and also allows a backup server to be used. TAF is intended to be used with RAC but can be used in traditional standby clustering or with replicated servers. However, TAF has a number of critical limitations; it drops all modifications in-flight and any PL/SQL sessions must be 're-built' as the stored procedures are dropped, further global temporary tables are dropped and Data Manipulation Language (DML) sessions are not restarted.

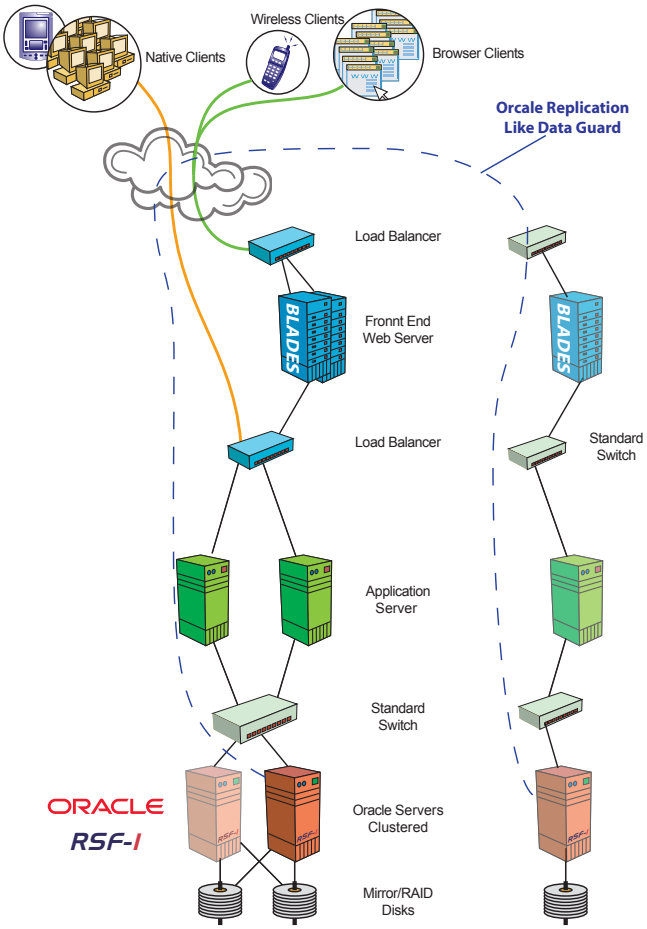

Figure 2.
Oracle Servers

Oracle is not a good example of a server component that is stateless. While applications can be written to minimise the impact of switching servers or rebuilding connections, developers must be aware not only of RAC and TAF but also understand their substantial shortcomings and then build code that can operate in a resilient configuration. Given the hurdles it is not surprising that very few application vendors build solutions that work with RAC.

Applications can be written to use Oracle in a way that means Oracle will operate as a stateless server.

Firewall

Firewalls are not really server applications in the traditional sense but there are some important issues to consider when designing a network solution where servers are protected by firewalls.

The basic function of a firewall product is to block unwanted traffic. The normal premiss is that all traffic will be blocked from entering a secured network, while most if not all traffic from the secured network is permitted to pass through the firewall. However, when a secured machine makes a request to a machine outside the secured network the firewall must allow the response back through the firewall. The firewall may be intelligent enough to fully understand the flow of traffic or more likely allow valid responses for a period of time. An example traffic flow is shown here;

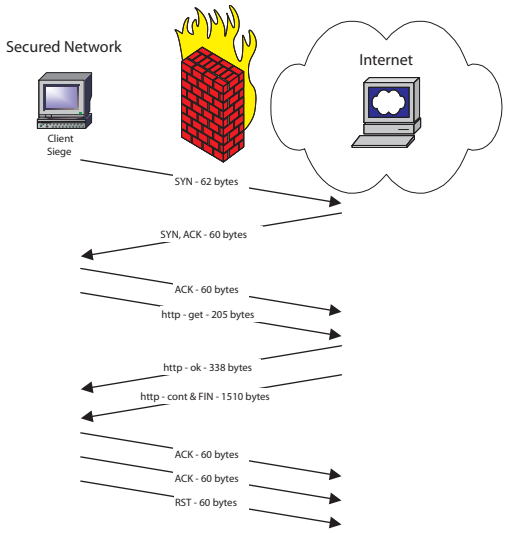

Figure 3.
Traffic flow to and from a secured network

To improve resilience and possibly performance more than one firewall is often installed. However, the path taken out of the secured network is not necessarily re-traced by the reply and hence a response may be received at a firewall which has not itself seen an outgoing request. For this type of multi-firewall architecture a system to synchronise the firewalls is required. Check Point's FireWall-1 product is a popular, if quirky, firewall product that includes 'state table synchronisation' to provide the type of synchronisation required for a multi-firewall architecture. An example configuration is shown here;

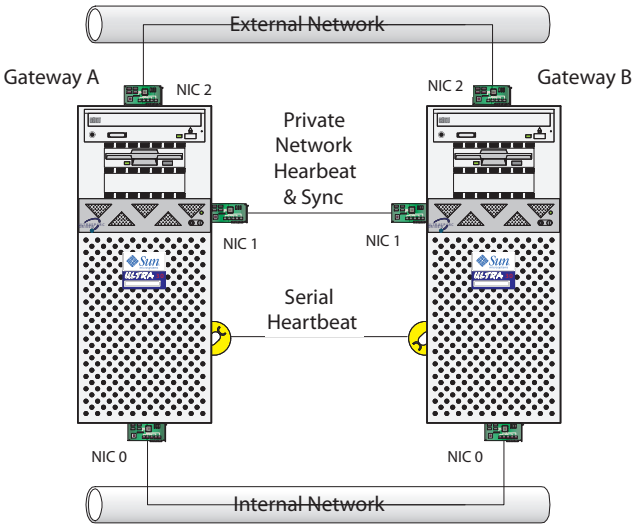

Figure 4.
Example firewall setup

The state table synchronisation operates of the 'Private' network in this example, which is the recommended deployment method. The updates are sent periodically and the frequency is configurable. There are constraints with the total number of connections that can be supported. The frequency of updates dictates the minimum response time that 'external' servers can respond in without error.

In the event of a node failing then the state is maintained by the other nodes and can indeed be re-established once the failed node is rebooted. The state information is fairly simple and even if a node were unable to rebuild the state information, because there was only one node for instance, then a brief outage would be unlikely to cause substantial disruption; most applications will retry but this of course cannot be guaranteed.