What is Fail-over

The goal of fail-over is to allow work that would normally be done by one server to be done by another server should the regular one fail.

For example, Server A responds to all requests unless it has a hardware failure, or someone trips over its network cable, or the data center it is located in burns to the ground. And if Server A cannot respond to requests, then Server B can take over.

Or if you simply need a service to be highly available, fail-over allows you to perform maintenance on individual servers (nodes) without taking your service off-line.

For fail-over server B would ideally be in a separate data center, or if that wasn't possible you would at least want to try and put it on a separate switch than Server A and on a separate power outlet as well. Basically the more physical separation the better.

BIG FAT NOTE: Failover is not the same as Load Balancing

What is Load balancing

Load balancing lets you spread load over multiple servers. You would want to do this if you were maxing out your CPU or disk IO or network capacity on a particular server.

Alternatives to load balancing include 'scaling' vertically. e.g. getting faster or better hardware such as quicker disks, a faster CPU or a fatter network pipe.

Implementing Fail-over

To implement fail-over you typically need to have your data replicated across multiple machines. You could do this via rsync+cron for files/directories. And via something like MySQL replication for databases.

One way to trigger the fail-over is to change the IP address your domain points to. IP address changes can happen within a few minutes of a DNS server update. Though if a client PC is caching an IP then it may take a bit longer for that to notice the change.

There are some services (e.g. http://zoneedit.com) that operate DNS servers that can detect a failure on a particular IP and automatically update the DNS for you.

http://pingability.com (run by the same people that run RimuHosting) also offers a failover server check type if you are using the RimuHosting name servers.

Implementing load balancing

One simple way to implement load balancing is to split services between servers. e.g. running the web server on one server and the database server on another.

This way is easy since there are no data replication issues. e.g. all necessary files are on the web servers, all necessary database data is on the database server.

Another common load balancing option is to have multiple front end servers. To distribute requests to multiple servers you could setup multiple IP addresses for a particular domain. Then clients should get all these addresses and to a random one. Spreading the load around.

Another way to distribute requests is to have a single virtual IP (VIP) that all clients use. And for the computer on that 'virtual' IP to forward the request to the real servers. eg with haproxy.

People can also implement load balancing via http balancers like mod_proxy_balancer in Apache 2.2 and Pound.

Clustering

A cluster is a group of replicated servers. Because there is more than one server, it is possible to load balance requests across them. In addition, a cluster means that the failure of one server does not mean the failure of the system. So, load balancing is just one of the features of a clusteed system, but there are others.

As mentioned in my previous post clustering is defined by a group of servers working together to provide increased scalability and high availability.

Increased scalability is provided by load balancing requests across the 'clustered' servers.

Increased availabity is provided by session replication and component replication across the 'clustered' servers.

Increased scalability can also be provided by a hardware or software Load balancer that lives outside the server cluster.

All of these solutions have trade offs that should be considered when you design your system.

What happens when a node in the cluster goes down?

Scenario:

One cluster with two nodes running a bucket named test. The bucket type is Couchbase. The bucket is set up to have one replica.

I upload 10K documents to the bucket. The cluster now divides the data across the nodes. Node 1: 5K active / 5K replica Node 2: 5K active / 5K replica.
If you remove one node and rebalance the replica of the still active node moves all documents from replica to active. Now the active node has 10K active and 0K replica. This is perfect behavior.

Now lets say I have both my nodes working with node 1 having 5K active / 5K replica and node 2 having 5K active / 5K replica).

Clients are now polling data from the cluster at lets say a rate of 2K gets per second. Then suddenly node 2 goes down. Now when the clients are polling data they will only get a result when the document is part of the 5K active on node 1. I would expect that the clients should get their data but with a little delay since it now needs to also get data from node one's replica documents.

What happens now (at least when using the C# API) is that when one node goes down half of the data is unavailable to the clients until a rebalance is done. If you have lets say 1M documents you will have about 500K documents unavailable to the clients until rebalance is done.

Shouldn't the cluster always return the same result as long as the data exist in active or replica?

Load balancing options at RimuHosting

We do not recommend you load balance on VPS'. VPS owners share their physical hardware with other users. If you are maxing out the CPU or disk IO or network then you are probably using too much of the host server's resource which would not be fair to other users.

If you are load balancing then you would probably need to have your own hardware (i.e. be on a dedicated server).

If you have multiple dedicated servers in one data center then we can setup one of them with LVS (see http://www.linuxvirtualserver.org/) to distribute requests. Or you could setup round robin dns. Another solution we have tested is to use pacemaker+corosync as described at http://www.clusterlabs.org

Failover options at RimuHosting

For failover we can help setup cron jobs to run rsync to replicate your file systems. And we can setup mysql replication to replicate your database. And you could use an automated service like http://zoneedit.com or http://pingability.com to 'fail over' the IP. Or do it manually yourself in the event of failure.

If you want to implement failover we recommend the failover server be in a separate data center. e.g. have your primary server as a dedicated server or VPS in Dallas and the failover server be a VPS or dedicated server in, say, London.

One issue with fail-over is falling back to the primary server. Say your main server fails. You fail over to the fail-over server. Your customers use that server. The files and database on that server are updated. When the main server comes back you would need to ensure that those changes are reflected on the main server. e.g. by rsyncing the files and by exporting/importing the database. You could do replication (of files and databases) in both directions to automate this. But that approach may lead to conflicts. e.g. what if two different people using two different servers tried to change the same thing? Which person's update would take affect? So some thought has to go into

recovery options before implementing to insure that will work the way you expect.

What about IP takeover/heartbeat?

A popular failover technique is IP failover. This is where a 'heartbeat' process runs on your servers. And in the event one server fails to see the heartbeat of another server it takes over an IP on that server (i.e. makes that IP route to itself, rather than to the other server that is not sending out heartbeats).

Typically IP takeover is implemented when two servers are connected on the same switch and are running on the same subnet.

RimuHosting does support IP Takeover, but there are some considerations you need to keep in mind...
- IP takeover makes it a bit tricker for us to troubleshoot problems and assist customers (e.g. which physical server are we actually going to check on?)
- For some servers (e.g. in the non-Dallas data centers we use) we do not control the switch. And the ip takeover would either not work. Or if it worked the data center staff may at some point may disable functionality needed for that to work.
- DNS failover may be just as good a solution.

What about 'shared storage'?

(Below is old documentation, which we are working on updating)

We have had a few people ask about clustered or shared file systems.

These can be tricky.

In an ideal world you would have a file system that any server could read/write to and where that filesystem was located on disks spread across multiple servers and where any of those servers or disks could fail and that would not affect the file system's availability.

In the real world to do this you need a clustered file system. That is a file system that knows it needs to co-ordinate any disk access between other servers in the cluster. To do that you need to have monitoring software (to check when a device goes down) as well as locking software (e.g. DLMs) that ensure that no two servers are writing to the same place, or that one server is not reading something that another server is in the middle of writing.

In the real world there are setups like the RedHat clustered file system. e.g. see http://www.linuxtopia.org/online_books/rhel5/rhel5_clustering_guide/rhel5_cluster_s1-ha-components-CSO.html These systems enable you to run clustered file systems. Although the setup can be 'somewhat' (i.e. really) complex. And it can also require specialized hardware to work. And often the system ends up being deployed on a single shared file system (e.g. a SAN) with no failover capability in the event the SAN fails (which SAN vendors will tell you will never happen, but which can actually happen, e.g. due to electricity providers or meteorities)

Often for many applications the 'simple' solution is for one server to export an NFS share. Other

servers can use that. And you would setup frequent rsync backups between that server and another. And in the event of a failure export the NFS share from the other server instead.