

# 1 Load Balancing

There are various reason why you'd like to load balance your web application. Understanding the reasons will allow you to concentrate your efforts in the proper direction.

The first thing to do is to analyze which part of the application is the bottle-neck.

## 1.1 The Connection

If your site hosts large amounts of images/data, the connection is the first thing you should look at. If you're constantly hitting the full capacity of your link, then it's a pretty clear sign that you should buy more bandwidth.

## 1.2 The Database

If your site handles a lot of detailed info about users, user generated content, is hitting the database a few times per request, then it might be a good idea to look into optimizing the database.

### 1.2.1 Connection Pooling

The first thing to try when it comes to databases is ensure that your connections are pooled (that you're not opening a new database connection on every web-server request). Also, to turn on any database optimization services that might be available.

### 1.2.2 Multiple Read Databases

One simple approach of improving database performance is to have multiple read databases, and one write database (that every read database mirrors off from, every few seconds/minutes). Since usually 90% of the database activity is reads, adding more read databases will improve performance. All writes need to go to 1 database though.

An example of this would be to setup 15 read MySQL instances, that mirror off 1 write MySQL instance.

## 1.3 Multiple Read/Write Databases

There are quite a few possibilities of setting up multiple read/write databases.

### 1.3.1 Oracle Grid

One approach is to setup an Oracle Grid environment. Essentially, it 'looks' like a single instance of oracle, but the load is distributed across multiple servers. Also, the fail-over service allows you to connect to the database even if some of the servers are down.

Each query to the database is executed on a single database instance; so if you have a grid with ten 8-CPU boxes, then you can likely handle 80 concurrent requests before you

begin to see a slowdown (assuming the data disks are connected via a fast link, and boxes have plenty of RAM for caching).

### 1.3.2 Netezza Approach

There's a database appliance named "Netezza". Essentially it's a grid on steroids. Idea is this: Your 'box' (size of a refrigerator) has a lot of 'mini-boxes' (lets say 400). Each of these mini-boxes has a CPU, a hard-drive, a decent amount of RAM, a fast network connection, etc. There's also a controller box (beefy multi-processor contraption).

All data is evenly (hopefully) distributed across all the mini-boxes. So if you store a 1,000,000 records in table, and your appliance has 400 mini-boxes, each box will get 2,500 records. Now, processing 2,500 records can be -much- quicker than processing 1,000,000 records, right?

Every query that you issue, is split across all the processors. The data is read 400 times faster than any single database, each database works on a relatively small number of records, etc., and the end result is your database is 400 times faster than... well... anything. ie: it can be 400x faster than Oracle for *some* things<sup>1</sup> (and obviously slower for some other things).

### 1.3.3 Budget Databases

You can achieve incredible performance if you take effort in organizing your data. One simple approach is to simply let the data tell you where it is stored.

Lets say you have  $N$  read/write databases. When a user 'bob' logs in, which database has the user data? You can find that out by doing something like `crc("bob")%N`, ie: treat the database has a hash table—hash the data to particular database instances.

This has the benefit of not crippling the whole system if a few boxes go down. For example, if some box goes down, then only a certain percentage of your users won't be able to login (and access their information), but most will be able to login just fine.

## 1.4 Web Server

If your site is getting so much traffic that you need yet another web server, the you're doing great :-)

Most folks need a second web server as a backup, in case they need to do some maintenance on the primary one (or for some other reason).

## 1.5 Complex Web Servers

If your web server isn't just serving static HTML pages, but is doing some processing (like running Java servlets, etc), then it might be a good idea to have a few. There are several ways of accomplishing that:

---

<sup>1</sup>It's also that much more expensive than Oracle.

### 1.5.1 One Entry Point, Proxy to different boxes

You can have one primary web-server, hosting HTML content, and have a sub-directory where the primary web-server will proxy off JSP/ASP/PHP, etc, requests off to different servers. Everyone sees the main site URL, even though processing is done elsewhere.

Your main server needs to have a huge amount of bandwidth for this to work right.

### 1.5.2 One Entry Point, forward to pool

Another approach is something what Yahoo apparently does. Have a common entry point, with basic HTML. Whenever the user does anything of consequence, you forward them to a dedicated host. For example, user comes in on `www.domain.com`, and is instantly forwarded to `abc123.domain.com`. You can have dozens (or hundreds? thousands?) of these hosts.

Subsequent user requests go to just 1 host.

The initial forwarding script needs to know which servers are in the pool, and forward requests based on some algorithm, either round-robin, or picking most free host, or picking the host that's physically closer to the client.

## 1.6 DNS Addresses

The DNS system supports basic load balancing. It is possible to specify multiple address lines for each domain name. When someone requests to resolve a domain name to an IP address, the addresses are return in shuffled order—usually the client (like a web-client) just picks the first.

For example, when you ping `google.com` from work, it will ping a different IP address than if you ping it again a few minutes later (from a different location hopefully).

The DNS system is a great way to load-balance your web-application, since users still see your domain name in their web-browser, and there's no discontinuity of their perception that they're still using your site.

Note, some clients cache DNS responses, so just because you remove a server from the DNS list, it doesn't mean that nobody can access it.

### 1.6.1 IP Failover

IP failover is setup for critical services, such as web-services or databases. Usually a hot 2nd copy of the database (or web-server) is kept online at all times. It has a script running that pings the primary box every few seconds. If the ping times out (there's a timeout period), the secondary box changes its IP address to that of the primary. All the clients that use the service don't even need to be aware that anything happened.