

Scalable web applications using OSS

Neil Blakey-Milner
Independent Online

nbm@mithrandr.moria.org
<http://mithrandr.moria.org/>

<http://mithrandr.moria.org/writings/scalable-web-applications-using-oss/>

What is scalability?

- It is not performance.
 - Performance is the speed at which a certain activity is performed.
- Scalability is *how easily an activity can support increased usage.*
- Or, *how well an application can make use of additional resources to improve its capacity to perform work.*

Huh?

- 20 junior programmers
- One expert programmer.
- One junior programmer asking the expert programmer for help will be good performance relative to going alone.
- Having all 20 ask for help every time they have a problem shows how this doesn't scale.

What do I know about it?

- Lead developer, Independent Online
- 20Mbit/s outgoing at peak, averaging 8Mbit/s
- Over a million unique visitors a month
- Multiple page views per visitor
- Many people visit twice a day
- Peak time is 8am to 9am, weekdays.

General scalability

- Ensure your need for computing resources doesn't outstrip what you have.
- Resources are finite: instructions per second, megabytes of memory/storage, and throughput.
- Main resources are CPU, RAM, and disk I/O throughput.
- Network I/O throughput might be a consideration in some environments.

The downward spiral I

- Premises:
 - The faster you respond, the more requests per second you can handle.
 - The less resources you use, the more concurrent requests you can handle.
 - The slower you respond, the more concurrent requests you need to handle.
 - If you run out of a resource, you will respond slower.

The downward spiral II

- Conclusion:
 - Too many concurrent requests will cause you to run out of a resource, and you will respond slower.
 - The less requests per second you can handle, the more concurrent requests you need to handle, which means you respond slower...
- Cap your concurrent capacity to prevent an out-of-control spiral...

Web Application Architectures

- CGI: new external process per connection
 - High-overhead (process, application, request startup)
- Embedded: Web server is the application
 - Medium overhead (application/request startup)
- Application Server: Separate process(es) from web server
 - Low overhead (request startup)

Which way do you want to scale?

- Vertical scalability
 - Being able to make use of additional resources in a single computer
 - Lots of CPUs, RAM, and disks
 - Increasingly expensive as you approach the top-of-the-line hardware
- Horizontal scalability
 - Being able to make use of additional resources by “adding more machines”
 - Lots of lower-spec machines
 - Can be kept relatively inexpensive

LAMP: The ingredients

- Linux
 - The powerful operating system
- Apache
 - The ubiquitous web server
- MySQL
 - The speedy database platform
- PHP
 - The popular programming environment

LAMP: characteristics

- Embedded application architecture
- Database-driven
- Usually database-bound
- Suited to rapid development
- Aimed at ease of deployment
- Scales both horizontally and vertically
- Memory is primary resource on web servers
- Disk I/O and memory are primary resources on database server.

LAMP: advantages

- Open Source(!)
- Can be had for free, if necessary
- Well documented
- Vibrant, supportive, community
- Commercial support available

LAMP: disadvantages

- Like everything else, it requires thought to scale.
- Like everything else, popularity increases the number of people who think they know what they're doing.
- Like everything else, you'll find detractors convincing staff that “it can't scale”.
- Still less mature from a programming point of view than some platforms.

Linux scalability

- Kernel 2.6 has a number of scalability improvements
- Understand resource limits for your OS
 - file descriptors per process
 - file descriptors per user
 - memory usage per process
- Keep database and web servers on separate machines – tuned differently
- Consider FreeBSD (well, it's what we're using...)

Apache scalability I

- Avoid DNS lookups
 - *HostNameLookups Off*
 - *Avoid Allow/Deny*
- Avoid unnecessary filesystem checks
 - *Enable FollowSymLinks*
 - *Avoid SymLinksIfOwnersMatch*
 - *Avoid AllowOverride*
- Don't load unnecessary modules
- Use static (built-in) modules

Apache scalability II

- Pre-forked architecture (Apache 1.3)
- Configuration
 - *StartServers*: Your average load excluding totally off-peak times
 - *MaxSpareServers*: Handle spikes in usage by having extra servers ready for action
 - *MaxClients*: Maximum processes at a time, prevents resource starvation spiral
- Logging
 - Consider logging over the network
 - Else, log to a single file if possible

Apache scalability III

- Keepalive keeps connections open in case there are additional requests
 - Primarily useful for static content
 - Wastes a process for web applications
 - Very impractical when more users than processes available
- lingerd
 - Takes over the job of timing out a connection from an Apache process, freeing it to do real work instead of wait around

Apache scalability IV

- Reduce network write latency
 - Use a reverse proxy
 - Separate static content to another web server
 - Use `mod_gzip` or similar compression
 - Tune *SendBufferSize*

MySQL scalability I

- Hardware
 - Don't skimp on memory
 - Fast, not necessarily big, disks
 - Use RAID striping (1, 0+1) for increased disk throughput

MySQL scalability II

- Configuration
 - Default settings suited to small, shared, servers
 - Don't bump up per-connection settings too much, or you'll run out of memory under load
 - Focus on global buffers/caches like *key_buffer_size*, *innodb_buffer_pool_size*, *table_cache*, *thread_cache_size*

MySQL scalability III

- Query cache
- Optimise queries
 - Use indexes
 - Only return the necessary data
 - Remember: Only uses one index per table per query(!)
- Replication (spread the load)
 - A server can only have one master...
 - All writes happen on all servers...

PHP scalability I

- “shared nothing” architecture
 - stateless, just like HTTP
 - No sharing between instances
 - Does the same work over and over
 - BUT it's well-suited to horizontal scaling
- Use caches (database I/O, CPU)
 - Page cache – cache whole pages
 - Still do the same work over and over inside...
 - Function caches – cache static data
 - Need to understand the volatility of each type of data in your application

PHP scalability II

- Use an “accelerator” (I/O, CPU, mem)
 - By default, PHP interprets each source file on every request
 - The accelerator caches the result of the interpreter for reuse.
- Build options
 - Disable unused extensions (mem)
 - Investigate compiler optimisations (CPU)
 - Compile statically into Apache (CPU, mem)
 - Strip PHP files (mem)

PHP scalability III

- Configuration
 - Disable *register_globals*, *magic_quotes_**, *register_argc_argv* (if possible)

Network Architecture I

- One machine, running web server and database
- Pro
 - Less moving parts
 - Very cheap
- Cons
 - One huge SPOF
 - Hard to tune for both uses
 - Vertical scaling expensive

Network Architecture II

- One web server machine
- One database machine
- Pros
 - Cheap
 - Easier to tune
 - Easy to administer
 - Can start horizontal scaling easier
- Cons
 - Two big SPOFs

Network Architecture III

- Multiple web servers (load balanced)
- One database
- Probably the most common scenario

- Pros
 - Web servers are easily horizontally scalable
 - No SPOF on web servers
- Cons
 - SPOF on database
 - Database may become bottleneck

Network architecture IV

- Multiple web servers (load balanced)
- Multiple databases (load balanced, replicated)
- Pros
 - No SPOFs
 - Less bottlenecks
- Cons
 - May start becoming expensive
 - High maintenance overhead

The IOL architecture

- Front-end load-balancer
- Multiple (four atm) public web servers
- One admin web server
- One master database for admins
- One slave database for web servers
- One slave database for text searches

- Missing
 - Separate image/static content server
- Hosted at hosting center

The IOL architecture II

- Actually, the database also contains the archive for all group newspaper articles over the past five years...
- Don't want to compete with web users
- Archive web server and slave database
- Backup machine
- Hosted at offices for extra access speed

Conclusion

- Focus on your resources and where you are or will be reaching their capacity.
- Trim down on resource usage by caching, recompiling, and disabling unused functionality.
- Plan for horizontal scalability – your web servers especially.
- LAMP can and does scale.
- But, like any other option, you need to think about it, and know what you're doing.