

Writing Scalable Web Applications

(And why no-one really bothers..)

Adrian Chadd <adrian@creative.net.au>

(NOTE: Somewhat technical in places!)

Overview

- I'm going to do a **very** high level overview of how and why web applications typically don't scale..
- .. and then I'm going to talk about why people generally don't bother ..
- .. finally, I'll briefly cover some things that people do to work around said issues
- .. if people ask, I'll answer some questions on the whiteboard.

Why don't things scale?

- Generally running out of some kind of resource:
 - CPU
 - Memory
 - Disk
 - Database
 - Networking

What do people do?

- Buy more hardware
- Buy more bandwidth
- .. people are more expensive than computers?

When won't it work?

- When one particular component is just not scalable by adding hardware
 - Eg databases
- Scaling isn't always linear
 - Doubling the hardware doesn't always mean doubling the performance
- So at what point does one have to care?
 - Do you ever have to care?

What about “the cloud”

- Something people keep talking about
- But what is the “cloud” ?
 - Is it a way of having a bunch of physical resources which can be shared and moved around?
 - Is it a way of distributing processing and content delivery around a large network?
 - Is it both and more?

So how to scale it?

- A few conceptual ideas can go a long way..
 - How the internet/networking works
 - System clue - profiling, behaviour, etc
 - Live and historical monitoring
- (But no, I won't be covering it all in detail - I'm just planting the seeds of clue..)

Processing Time?

- Web code can be small or large
- Web frameworks aren't always written to be scalable; they're often written to be usable or pretty
- Testing individual script execution doesn't tell you how well it will scale with lots of users

Disk?

- Static content (eg images)
- What about logging requests to disk?
- What does your application do when you've maxed out the available disk IO?

Databases?

- A lot of web applications grab data from various sources!
 - .. sometimes SQL ..
 - .. sometimes with lots of layers between the code and SQL ..
 - .. sometimes other data sources ..
 - .. which may be SQL ..
 - .. which may be other data sources ..

Resource usage?

- Performance on one user doesn't reflect multiple users
 - .. all at once ..
 - .. which may or may not be executing quickly
- So all of the resources used during a request may be tied up (especially RAM) until the request finishes

Networking?

- A hint - “LAN” is not “WAN”
- Another hint - applications behave differently when on the WAN
- .. and how users actually perceive how well the application works
- (And if you care about it; how TCP/UDP behaves under load, latency and packet loss)
- HTTP Caching - not just for saving bandwidth!

What else?

- What about HTTP?
 - Eg Caching, content negotiation, etc
- What about end-user experience?
 - Page loading times, rendering speed, local resource usage
- .. basically - stop pigeon holing yourself into one area and learn a little bit about everything.

Cloud?

- How about scaling things onto a cloud?
 - Databases aren't SQL-like - bigtable, etc
 - Processing isn't normal - map/reduce, etc
 - Content delivery doesn't happen from one spot - eg CDN
 - Interdependencies between software modules

Cloud (ctd)

- Eg Amazon S3 ?
 - Run virtual *NIX machines under Xen
 - Amazon “storage” and “database” to distribute content and information
- Eg Google Apps ?
 - Python framework; map/reduce, bigtable, etc
- Eg Sun, VMWare, etc, etc ?

Some suggestions?

- Look at decoupling a whole lot of your application
 - Database fetching
 - Processing
 - Page fragment assembly
 - Page assembly
 - Content serving to users

More suggestions?

- Learn about “internet”, HTTP and TCP
 - Even if superficially!
 - Especially HTTP caching! (seriously!)
- Look at modelling the behaviour of your application(s) and note interdependencies which may impact performance
- Doing this will almost certainly show a whole lot of places where information is re-used over and over and over and over..

More suggestions?

- HTTP caching isn't just for saving bandwidth
 - Page fragments can be cached
 - Database accesses can be cached
 - HTTP caching layer in between rendering layers, application layers and fetching layers
- Don't discount system monitoring - profiling, historical information, etc

More suggestions?

- Investigate other methods of distributing data between servers and nodes
 - caching - memcached, http caching, etc
 - distribute frequently used data instead of backing it into SQL
 - Investigate bigtable, Hadoop, etc, and understand what the implications are
- If in doubt - find someone who knows and bribe them with toys!

Time to draw..

- (At this point I'll be drawing stuff on the whiteboard instead of spending time doing up graphics here.)

Questions?

Thanks!

- Adrian Chadd <adrian@creative.net.au>