Web application Scaling

Pete Stevens

Mythic Beasts Ltd

Basics

A perfectly scalable site has:

The same user experience with 100 users as it does with 10 billion.

Marginal cost of adding a user is a monotonic decreasing function of number of users.

Resource limitations

- CPU cycles
- Disk IO
- RAM
- Money
- Bandwidth
- Development time
- Staff time
- Products
- Change control

CAP theorem

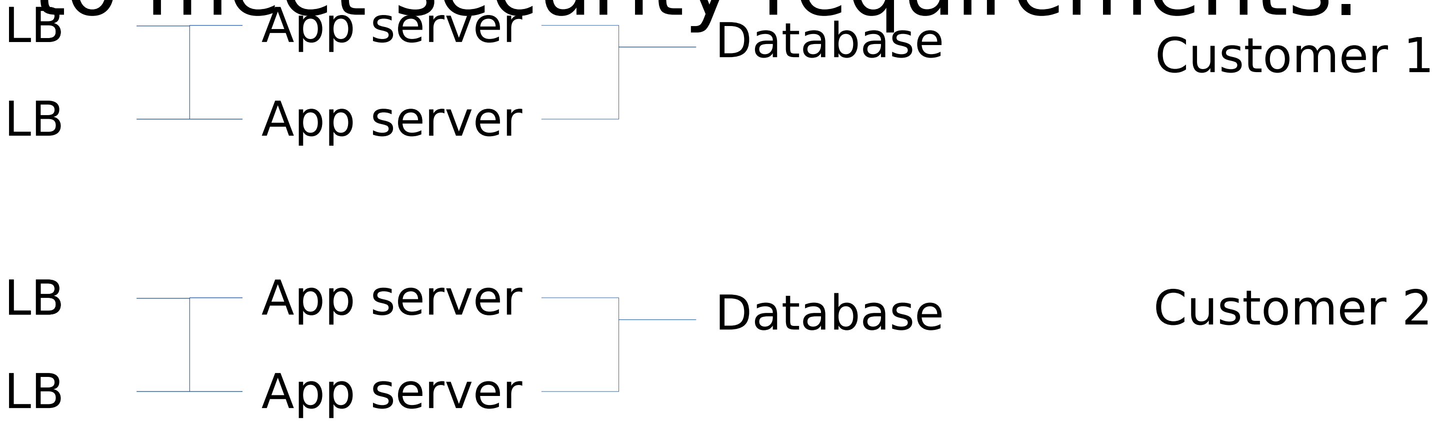
- Consistency / Availability / Partition resistance.
- In a distributed data store you can have at most any two properties.
- In a newly released piece of miracle software you're usually lucky to get one.

Methods of scaling

- Add servers
- Grow servers
- Cache data
- Improve software performance
- Parallelize

Single tenant architecture

- Set of servers per customer.
- Strong security separation between customers.
- Each customer is scaled individually.
- Easy to meet security requirements.



LB

LB

App server

App server

Database

Customer 1

LB

LB

App server

App server

Database

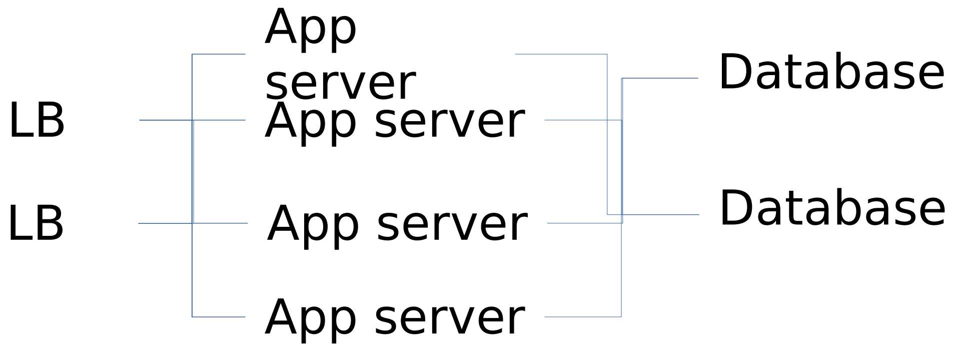
Customer 2

Single Tenant Architecture

- Gets very expensive in servers / virtual machines / containers.
- Automated patch and update management.
- More secure architecture and firewalling.
- Allows on premise and cloud solutions.
- Allows different versions per customer – no forced upgrades.
- Billing is easy – buy fixed sizes.
- Examples, ResourceSpace, Pagely.

Multi-tenant architecture

- One instance contains all the customers.
- Security provided by application code.
- Trivial to add new customers.
- No scaling issues for individual customers.
- Scale the whole cluster.



Multi-tenant architecture

- Any bottleneck downs everything.
- Security hard, e.g. Dropbox 2011, Valve 2015.
- Single codebase does the superset of customer demands.
- Maintenance much simpler.
- Much more efficient resource usage.
- All customers must run the latest version.
- Per customer feature requests very difficult.

Vertical scaling

- If service too slow, replace with faster computer.
- Easy to implement, especially with cloud providers.
- Moores law is no longer helping, computers have stopped getting faster, they merely parallelise better.
- Typical for database servers and smaller organisations.
- In 2018 the Meltdown and Spectre security issues meant that computers have become slower.

Horizontal Scaling

- Add more application servers, share the load between them.
- Applications need to be stateless.
- SQL databases don't horizontally scale.
- NoSQL databases don't guarantee consistency.
- Sharding data can help, but cross database write locks still don't scale.

Caching

- Caching stores the results of calculations and reuses them.
- Multiple points to cache:
  - Output (varnish).
  - Intermediate objects (redis/memcache).
  - In database (Mysql Query cache).
  - Object code (e.g. php-apc).
  - Custom (e.g. Wordpress Supercache)

Caching

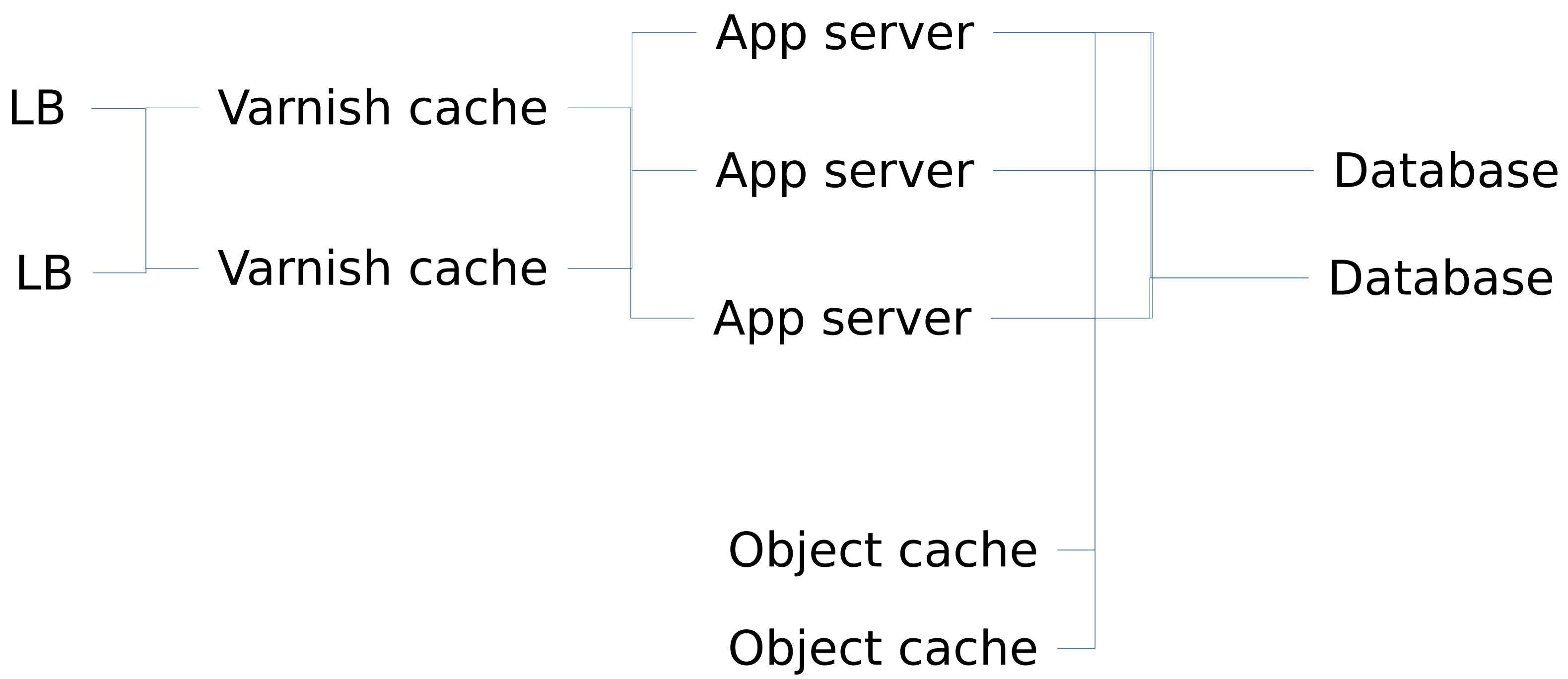
- Cache invalidation is very hard.
- How important is it not to serve stale data?
- e.g. Bank account value,
  - vitally important to update the balance after every transaction.
  - Better to return an error than allow someone to use an incorrect balance.
  - Timeout much less bad than allowing someone to spend money twice.
- e.g. Social media plugins
  - Much more important to deliver a page
  - If a 'number of shares' figure is incorrect, who cares?

Caching

- Caching allows you to reduce the load on your servers.
- Vertical scaling lasts longer.
- Fewer servers horizonally scaling.
- But, may become critical for site operation.
- A cache flush might take your site offline.
- Distributed horizontal scaling of your caching.
- Different people may see different things – does this matter?

Base architecture

- Simple architecture that is a good starting point for most applications



Scaling the base architecture

- Load balancers – horizontal
- Varnish cache – horizontal
- App servers – horizontal
- Object cache – horizontal
- Database – vertical

Application performance

- By default a webpage will take between 0.1s and 1s to render.
- If it's quicker than 0.1s the developers will add functionality.
- If it's slower than 1s the developers will be annoyed enough to improve the performance.

Load profiles

- Business application, 9am – 6pm peak.
- Consumer application, lunchtime and evening peaks.
- Product site, peaks for product launches.
- Viral advertising / social media, exponential growth.
- Advertising / press, huge spike, exponential decay.

Example: Netflix

- Netflix have the main website in Amazon AWS.
- Every part was re-architected to horizontally scale – Cassandra NoSQL database.
- A constraint is bandwidth costs, Netflix costs $9.99/month and consumes 7GB of data per hour for UltraHD (15Mbps).
- Amazon pricing, $0.05/GB in volume. 28 hours/month and they lose money assuming everything else is free.

Netflix CDN

- Internet Exchange ports, 10Gbps for ~ $1000.
- 10Gbps is roughly 2000TB/month = $0.0005/GB.
- 100Gbps ports ~ $8000 = $0.0004/GB.
- Deploy CDN boxes directly into ISPs, $10k hardware cost, 4x10Gbps links, over 2 years = $0.0001/GB.

Netflix CDN

- Slower rollout and coordination compared with Cloud providers.
- Bandwidth costs 1/10th to 1/100th.
- Trade staff and capital costs for cheaper bandwidth.
- Entrenches a competitive advantage – cloud based competitors run out of money very quickly.

Example: Blackblaze

- Online backup company.
- Store lots of large files as cheaply as possible.
- $5/month for unlimited data storage.
- Very important to keep the data storage costs as low as possible.

Example: Blackblaze

- Custom storage hardware.
- Storage pod: 45 hard disks, 3x15 drive RAID6.
- Vault : 20 storage pods.
- Each file is Reed-Solomon encoded, 17 data, 3 partity and split across a Vault.
- Each pod runs Apache, files over HTTPs.
- Front-end assembles from the 20 pods.

Example: Blackblaze

- Vault contains 900 hard disks, 663 disks of capacity, 237 of redundancy.
- Survives multiple drive failures and 3 pod failures.
- Cloud storage option, AWS is $0.03/GB. Blackblaze is $0.005/GB.

Facebook: Photos

- First implementation: Big network file system and a mysql database to hold the meta-data.
- Vertical scaling, lasted three months before they needed caching layers for small and common images.
- Eventually a complete rewrite to store binary chunks indexed by database.
- NFS doesn't work with 80bn files!
- They would do exactly the same thing today.

Facebook: Photos

- The NFS solution was quick to implement and allowed them to work out the functionality.
- Optimisation was driven by the access patterns.
  - Small images much more commonly accessed than large ones.
  - Cache those preferentially.
  - Use a CDN (in-house, see Netflix for bandwidth costs) to offload.

Facebook: PHP

- The Facebook stack is fairly simple:
  - Database (slow, persistent).
  - Application (PHP).
  - Memcached – 28TB of RAM to allieviate PHP cycles and database access.
  - Load balancers.
  - Services (varied).

Facebook: PHP

- Memcached reduces the PHP processing load enormously.
- But still >10,000 servers processing PHP.
- HipHop VM, virtual machine to speed PHP execution.
- Roughly 3-5x performance increase.
- Huge cost savings.

Example: Music Streaming Site

- 100 machine cluster, ~ 400 cores, large application.
- Memcache used to store all the intermediate calculations.
- Nobody noticed that with the cache off the front page had increased from 0.1s of CPU time to 100s per request as memcache hit it.
- One day we flushed the cache in production – 10 minute outage while it rebuilt all the intermediate objects.

Raspberry Pi

- The original aim was to produce 10k tiny computers.
- Run linux, designed to be programmable by computer science students, the first wave of which are sat in this lecture theatre.
- Severely budget & time constrained.

Raspberry Pi v1

- Launch of the original Raspberry Pi.
- Volunteer project, minimal funding – hosting budget £4/month.
- Aim: direct 10,000 or more people to RS components and Farnell who were selling Raspberry Pi.
- Wordpress based dynamic site, blog + comments + forums.

Raspberry Pi v1

- Replaced site with a static webpage
- Performance improved from ~5 pages/second to 1000+
- Sent >100,000 visitors to the vendors
  - Who fell over – linked to their search pages which were dynamic and not cacheable.
  - 1s+ of CPU time per visitor on their sites.
  - 'Bulletproof' major e-commerce websites overwhelmed.
  - Bottleneck was belief.

Raspberry Pi v1

- Human resource limits:
  - Not enough staff (1 part time) to answer queries.
  - Not enough stock to make the products.
- Post launch the website traffic never returned low enough to turn the existing site back on.
  - Vertical scale – move from shared hosting account to big dedicated server (4 cores / 96GB RAM) (2012 – this was a huge machine).

Raspberry Pi. DDOS

- Flooded with tcp connection opens.
- Syncookies means CPU but no RAM until connection open.
- Ran out of CPU to process syn packets.
- Ran out of network capacity to receive syn packets (500k/sec, > 1Gbps).
- Front with 4x1Gbps dual core machines just for MD5 sum calculations.

Raspberry Pi: Downloads

- Donated a downloads server with a 100Mbps unlimited use network port.
- Images are ~ 1GB in size, 100Mbps means 1m40 to deliver an image ~ 800 per day.
- Filled immediately, built a mirror network from donated servers and bandwidth.

Raspberry Pi Downloads

- 1Gbps means we can serve an image download every 8s per 1Gbps network port.
- Scale horizontally, add little servers that deliver 1Gbps each – handily on the DDOS servers.
- 8 servers allows an image per second which is enough.
- Raspberry Pi uses more bandwidth than the University of Cambridge.

Wordpress

- Wordpress is a PHP application.
- Naive configuration under apache/nginx runs a php interpreter for every page request and compiles and executes 400k lines of PHP on every request.
- Dynamic language – interpreted on every request.
- Many CPU cycles spent compiling PHP on every request.

PHP caching

- Advanced PHP Cache, compile once, store intermediate object code.
- Only recompile when the source files change.
- Improves Wordpress performance by up to 50%.

Wordpress Caching

- Wordpress generates semi-static pages :
  - Only change when user content is added.
  - Supercache stores the generated HTML and outputs that unless the content has changed.
  - To avoid recompiling the whole of wordpress on every request, if a cached page exist it just compiles a shim to output a static file.
  - Dynamically changes the code to compile based on the data in the request.
  - If APC caches the object code...

Raspberry Pi. Performance

- Types of request:
  - Dynamically generated page (0.5s CPU, 8/second) ~ 10%.
  - Wordpress cached page (0.01s CPU, 400/second) ~ 90%.
  - Error page served from the load balancers (0.0001s, 40k/second) ~0%.

Raspberry Pi 2

- Launch of Raspberry Pi 2.
- Simultaneous press launch at 9am, radio, TV, multiple internet sites.
- Expecting 1m + visitors ~ 20 per second.
- Expecting a high cache hit ratio.
- In the event the site is overwhelmed we set the error page to the announcement – repeat of Pi 1 launch.

Raspberry Pi 2: Mistakes

- Super top secret – I didn't know about the launch.
- Instead of monitoring and managing the site, I was without internet access on this beach

Raspberry Pi 2

- Any logged in user doesn't get a cached page
  - Because of our comments plugin this includes anyone who's ever posted a comment in the past.
  - Cache hit rate muchlower than expected.
- On posting of a comment the cache is invalidated.

Raspberry Pi 2: Cache invalidation

- Wordpress / Apache don't queue requests, they process them all in parallel.
- Lack of a cached page means that every process starts generating replacement pages at the same time.
- If number of requests > number of cores, each additional request slows down the currently processing ones.
- If the request rate is high enough, a page generation will never complete.

Raspberry Pi 2: Cache invalidation

- Eventually the load balancers decide the site has failed and serve the error page.
- Load subsides and the cached page is created.
- Traffic is directed back and the site works again.
- Until the next comment is posted.
- We run into performance issues at 2000 simultaneous visitors, peak about 4500 ~ 10 million visitors on launch day.
- Not a disaster, but not really a success either.
- Exercise: write some code to simulate just how bad this is, asymptotic to a maximum performance level.

PiZero

- Site architecture change
  - Lots of VMs for each part of the site.
  - Key parts all load balanced over multiple VMs.
  - Easy to add VMs – scale all the PHP processing horizontally.
  - Split the databases into separate VMs – shard the data stores.
  - Scale the database servers vertically and offload with caching.

PiZero

- Multiple Webserver VMs should solve the comment posting problem
  - While one VM is under heavy CPU load the load will be transferred to the other VMs.
  - Providing they don't all invalidate simultaneously.
  - Not prepared to put this to chance.
  - We know, that our test setup isn't good enough to catch the comment posting issues we had last time.

PiZero

- Additional static caching
  - Pre-render the most popular pages on the site and always serve static HTML.
  - Might be up to 60s out of date, but always fast.
  - Could achieve with Varnish (see earlier slides), possibly in a more clever way.
  - Simple configuration reduces the chance of something going wrong.

PiZero

- Over 10000 simultaneous viewers at peak.
- ~ 75 million page views in a day.
- We probably could have handled 20k-50k, network limitations in the host (4Gbps) was the next obvious bottleneck.
  - We don't know where the next non-obvious bottleneck is!

PiZero: Magpi

- PiZero was launched on the front of the MagPi magazine
  - MagPi website sees a huge influx of traffic.
  - Designed by the 'techie genius' of the design agency.
  - Delivered late – night before – no significant testing.
  - Page generation time ~ 0.2s => 10,000 visitors/second => 2000 cores.
  - Live patch the configuration/code to reduce CPU load.

PiZero: Magpi

- Wordpress cache, set to maximum caching.
- Deploy static page generation for the most popular pages.
- Called Google+ 8 times per page request to get number of shares – deleted functionality.
- Dynamic elements of pages and menus – all forced static on a 60s refresh.
- Ideally done in Varnish but Keep It Simple Stupid (KISS) applies – minimal changes to production.

Raspberry Pi 3

- Stuck with the configuration we used for PiZero.
- Drank coffee.
- Smooth launch, not quite as busy as PiZero.

Raspberry Pi 4

Scaling: The End.

- Questions?
- Further questions, pete@ex-parrot.com
- https://twitter.com/Mythic_Beasts
- https://twitter.com/Raspberry_Pi