Scaling Web Services

Karl O'Brien, karlobrien@gmail.com

1. Architecture of the Web

"REST-based architectures communicate primarily through the transfer of representations of resources" - Fielding Thesis on Restful web-services

I remember learning about Distributed Applications during my university days and at that time it was just another module among many. It says something about where the subject of Distributed Applications stood that it was just one module – nowadays it is one of the most important theories in Computer Science. In that module, we learned about various theories about connecting large numbers of computers together in connected applications. Distributed applications typically used a certain protocol to share information but there was no set format. One company could use one platform for distributed applications very successfully while a different company could be using a rival system. Both systems work very well in isolation but to get one to talk to the other was a significant body of work. To be honest that module made the subject of "Distributed Applications" seem like a subject that was missing something, the current systems were inflexible, difficult to change with new requirements and inefficient. It just seemed really hard to get things done.

A couple of years before I took that distributed applications course, Tim Berners-Lee was busy designing and building the foundations of the World Wide Web [1] while a research fellow at CERN in the early 1990s. The designs that he laid down were an answer to the frustrations that I outlined with the module I took in Distributed Computing. He had also looked at the current way of doing things and decided that they were difficult to link together. He decided to apply some basic architecture that would make it easier to build applications that could talk to one another without needing to employ an army of programmers.

What Berners-Lee and others came up with suddenly made the whole topic of distributed applications more appealing. Instead of grappling with problems on how to integrate two completely different platforms, a whole new platform emerged where it was easy for different services to talk to one another. According to the World Wide Web Consortium, the anarchic architecture of today's Web is the culmination of thousands of simple, small-scale interactions between agents and resources that use the founding technologies of HTTP and the Uniform Resource Indicator (URI)

This new way of looking at loosely coupled systems resulted in an explosion of the web. Now we had a place where large companies, start-ups, charity groups could all speak to each other using a common protocol. None of the previous middleware protocols supplied this functionality. This explosion would bring with it new problems in terms of scalability. With this new platform, suddenly the subject became interesting again. Modules in universities would deal with solving real world problems such as making sure that your web service can be scaled to meet demand. Software and hardware would need to be turned to stand up to the ever changing demands of the web.

2. Scaling Out

The main verbs of the web are GET and POST. GET is used to pull a service or resource back to the user browser while POST is used to push data in the opposite direction. The “GET” verb provides an idea situation in that by its very nature it can generate no side effects on the server side that are caused by a client. It’s these very verbs that provide the solution to the issues I described with previous distributed systems, there is now a common interface which all types of applications can speak. The facts that a client can use GET to request a certain state from a service opens up a lot of possibilities for abusing the service and causing issues. A client that repeatedly requests data can cause issues for the service if it has to answer that request. As a result a general rule can be defined around using GET

GET: Requestor does not want to modify anything. In turn it makes more sense to store the response closer to the client. This will save bandwidth costs and also protect the service.

It was a PHD thesis by Roy Fielding [15] that outlined how useful the web could be in operating as means of representing state transfer across distributed systems. It was this way of thinking that gave birth to a large number of the services we see implemented so successfully today. It gave meaningful parameters as to how resources could be defined and addressed. It described a high level method of linking many technologies together.

The ability to be cache data close to the client is a huge advantage of the web

3. My Site is on Hacker News- What do I do?

Up to now, the paper has focused on a lot of the theory of distributed applications and some http facts. The next section is simple, say you maintain your own blog or web application and a recent release or update draws a high listing on a site such as Hacker News. The question now is not how many queries your web application is going to get but whether you can keep your host machine alive to service all the increased volume of traffic.

The first basic steps to deal with the issue of serving dynamic content are outlined below. More than likely you will be using something like apache to receive HTTP traffic and a language such as PHP/ Python/ Ruby to serve the content.

1. Disable onerous features
2. Serve stale data if needed
3. Serve static files.
4. Serve static files from a different host.
5. Serve dynamic results via App Engine or Amazon S3

The steps above were outlined in the “Managing Software in Production” module that this term paper is concerned with. But are there any other steps you can take to make sure that in the long run the application will stand up to large increases in traffic

3.1 Caching Systems

Caching was mentioned briefly in section 2. With the growth of the web, caching has become a huge area. There are varies different flavours of caching. One can place a local cache close to the individual user which can store their frequently accessed items so they will not have to continually request these resources from the origin server. Nearly all modern browsers [2] have caches build into them which can be set to a configurable setting. They can take advantage of the caching parameters built into the HTTP protocol. The “Expires” and the “Cache-Control” can specify how long a request remains valid and are how the caches are handled.

3.1.1 Typical Problem Environment

The main caching system that this paper wants to look at is the idea of a reverse-proxy cache and how it can be built into a modern web application environment. These are different in that in the past proxy-caches would have been deployed by ISP’s or network administrators to save bandwidth, the reverse-proxy caches are being put in place by the application owner to make their sites more scalable and reliable. There are a couple of different types of reverse-proxy (http accelerator) caches available – Squid [3] and Apache [4]. The http accelerator that we are going to speak of here is known as Varnish [5]. It was built from the start to serve as an HTTP Accelerator.

The diagram below outlines a typical modern web application environment [6]

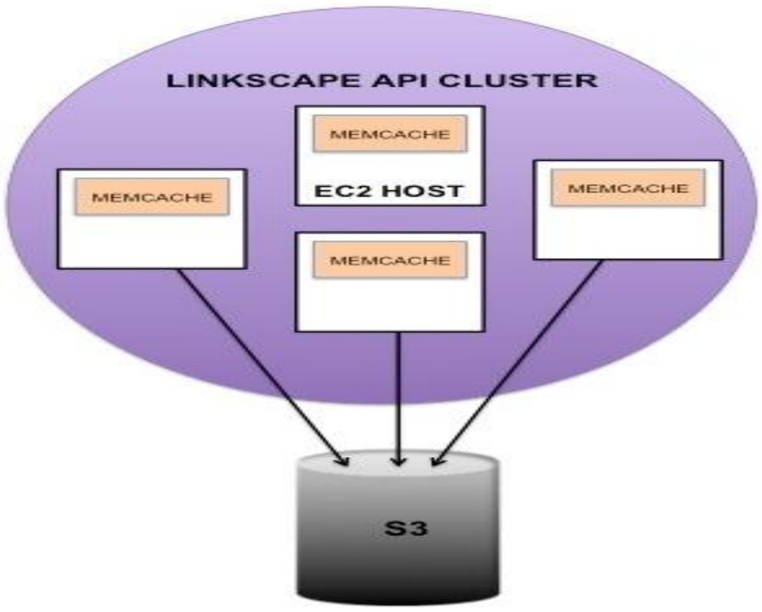

Fig 1.

As you can see from the diagram above, there is some sort of service that is running and making lots of read requests back to an S3 object. If a lot of requests are being put on the S3 object then the default behaviour is to throttle requests. One solution would be to add more S3 buckets to the environment however that means that you now have to pay for ever increasing storage costs from Amazon. The solution to this problem could be a reverse-proxy like varnish

3.1.2 Varnish Internals

- Varnish Configuration Language – VCL [7]
  Varnish does not have standard configuration file syntax, it has a VCL that is converted into C and compiled. This gives it enormous advantages over configurations file that span large files. It is possible to build new features into Varnish through the VCL using conditional statements.

- Logging is carried out in a shared memory segment. This provides huge speed advantages over standard file logging.
- Varnish has load balancing built in.

3.1.3 Elastic Load Balancing

While varnish is the subject of this section, there is also an instance of ELB introduced in the design below. This is defined as *"automatically distributes incoming application traffic across multiple Amazon EC2 instances. It enables you to achieve even greater fault tolerance in your applications, seamlessly providing the amount of load balancing capacity needed in response to incoming application traffic".* [8]

3.1.3 Using Varnish to solve a problem

In section 3.1.1, a typical problem was outlined with a rapidly expanding service. Fig 2 outlines a suggested solution to that environment. As you can see in the new environment, there are now two varnish daemons running on each host in the caching layer. This means that the front end caches can have a small amount of RAM and sit closer to the client to serve local requests. It will also reduce latency from the client as extra steps have been introduced into the design. The larger caches sit behind and can use the same amount of memory as the S3 instances behind them. In this scenario the backend cache instances can serve a lot of the data that you would otherwise have to request and pay for from the S3 instance. If you can image this new design as having a number of different states (storing stale data, expiring data, storing the cached response), you can provide a default implementation in the VCL to deal with each state. This is what makes varnish so powerful.

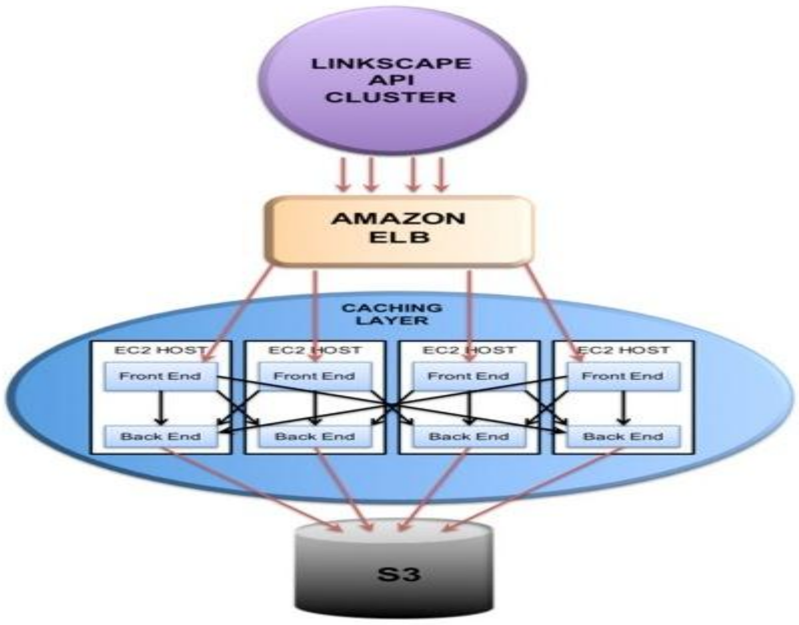

Fig 2.

Now the web service has additional capacity built into the design. If the caching layer comes under pressure, new EC2 hosts can be added to the layer without affecting the end user and without hammering the Amazon S3 instance.

3.2 Queuing Systems

3.2.1 Queuing in a Web Application

When it comes to scaling web applications, one place which you can see bottlenecks and very often make rapid improvements is in the area of asynchronous queues. With the complexity of today's web applications, users often expect them to perform in the same manner as a web application. With the amount of interaction on sites such as Gmail, Facebook and Twitter, there is nothing that will drive customers away from your site as quickly as a non-responsive web application. This is where the concept of asynchronous queuing comes in, the response does not need to happen instantaneously - it just needs to make the user think that it does. Very often when you hit that tweet button in an app, the request is taken and placed in a queue. In fact twitter has had many very public outages [9] and has carried out a lot of work on its queuing systems using Kestrel [10].

3.2.2 An Example Application

Working in the high frequency trading financial industry, there are huge volumes of financial data. There is the ingoing problem of designing systems to deal with fluctuating markets incorporating peaks and dips in volume. While a web service would not scale very well in terms of latency against trading applications that make use of items such as FPGA cards and infiniband [16]. There are instances where latency is not a concern and this is where something like a web service would come in useful.

Say for example there are 3 markets that we want to look at data from.

Stock Exchange	Description	Stock
London	Vodafone Stock	VOD LN
Germany	BMW Stock	BMW GY
New York	Ford Stock	FORD US

Table 1

There are a couple of ways to approach this problem.

1) Sequential updates – We subscribe for each stock and wait for a response. The main issue with this method is that if one update hangs or comes under pressure then all the updates will be affected. This is not the approach you take in a volatile environment.
2) Using desktop applications the obvious design is to go with high end machines and distribute the application across several CPU instances. This provides scaling and also separates our environment out so that high traffic on one CPU should not affect the others. There is still the issue of dealing with physical hardware. When all the physical cores are being used the only option with this design is to purchase and maintain more hardware. The method of scaling will become very expensive very fast.
3) Another way of looking at this problem is to use a service to handle the scaling for you. In this way you can concentrate on doing the parts of the business that you are an expert at (trading stock) and letting the service complete the heavy lifting. In the example outlined above it is important to treat each subscription to an exchange as a unit of work.

3.2.2 Amazon Simple Queue Service (Amazon SQS)

Instead of building out the infrastructure for implementing your own queuing system, Amazon provides this as a service. Fig 3 outlines a typical scaling system that can be put in place using a combination of amazon technologies. In this case the web server passes each unit of work into a queue provided by Amazon. When the queue is ready this is passed to the Market Tick Server which can perform some transformations on the data (maybe convert it to a normalised form). The important point in this case is that the web server keeps receiving the data and passing it to the queue, the possibility of a bottleneck is moved from the Web Server to the queuing system. It can also be possible to input the raw data directly into an S3 instance. If the Market Tick Server becomes unavailable, the Web Server will be unaffected. If the Market Tick Server remains down over a period of time, then it is possible to add another server to the backend without affecting the frontend.

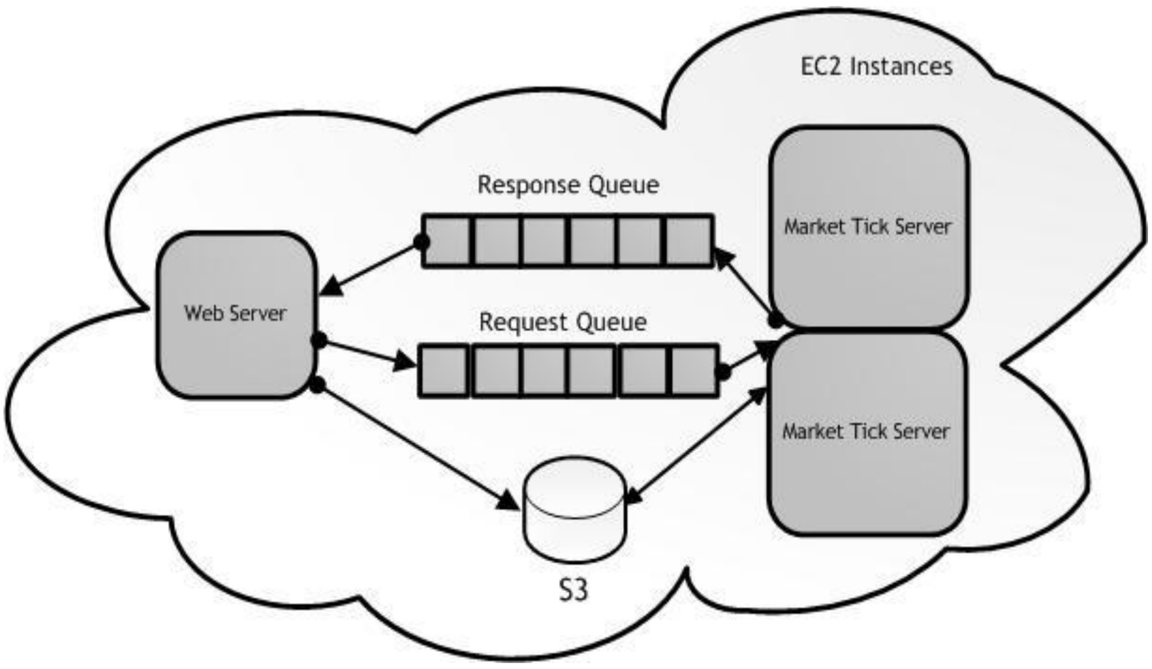

Fig 3. Example taken from [14]

4. Wrap Up

This paper has briefly introduced some topics in scaling web services. It is interesting to note the cycle that is currently taking place where large companies such as Google and Amazon are providing large amounts of computing power to take care of tasks where clients traditionally used to design and implement their own solutions. This is similar to the situation that I described at the start of the paper where in previous models of distributed computing, a customer tried to implement the entire solution on their own systems. One of the most important concepts that have arisen from the current trend is the idea of where to locate your data. This decision occurs at the system level. There are 3 important ideas associated with this concept

- Where the data is
- Where the data needs to be
- What collections of it you need to process operations

Concepts like map-reduce and as a result Hadoop [11] mean that there are very efficient methods and services available to efficiently deal with large amounts of data. Technologies such as this enable a company to concentrate on growing their business instead of worrying about looking after huge amounts of data and hardware. For example if you are starting an online business, you have the option of using something like Google Apps or as already mentioned Amazon Web Services. Using these technologies at the start removes so many possible problems. Using a service like this will mean a lot of the basic headaches of auto scaling will already be taken care of for you.

Another recent trend that is worth mentioning here is the move back towards static web sites. So much of the early 2000's were focus on getting your site onto a platform such as Wordpress [12] where a user could simply add plugins for every type of scenario. Problem was that some of these sites did not scale well as some of the plugins were not designed with large traffic volumes in mind. Many popular blog platforms are moving back to serving static web pages and using services such as Disqus [13] to serve content via JavaScript libraries.

As each of the services described in section 3 is being developed at a very fast pace and new products are being brought to the market constantly, it is important to note that it is some of the basic principles that were discussed in section 2. It is the fact that these services can leverage the basic principles of http that makes them so powerful in the area of scaling so many different and unique web applications.

Appendix

1 www.w3.org

2 http://www.mnot.net/cache_docs

3 http://wiki.squid-cache.org/FrontPage

4 http://www.apache.org

5 http://www.varnish-software.com

6 http://devblog.seomoz.org/2011/05/how-to-cache-http-range-requests

7 https://www.varnish-software.com/sites/default/files/varnish_cache_whitepaper.pdf

8 http://aws.amazon.com/elasticloadbalancing

9 http://techcrunch.com/2009/08/06/serious-twitter-outage-ongoing

10 https://github.com/robey/kestrel

11 http://hadoop.apache.org/

12 http://wordpress.org/

13 http://disqus.com/

14 http://sqs-public-images.s3.amazonaws.com/Building_Scalabale_EC2_applications_with_SQS2.pdf

15 http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

16 http://www.infinibandta.org/