Computer Science 75 Lecture 12: December 7, 2009
Fall 2009 Andrew Sellergren
Scribe Notes

Contents

1 Announcements (0:00–2:00)

- Today's the last lecture of the semester!
- We'll be taking a week off next week, but we'll return the week after for the Computer Science Fair. On Monday, December 21, we'll be meeting in Maxwell-Dworkin Room 119 (one level up from the ground floor) and you'll have the opportunity to mingle with students and families of students from E-75 as well as E-7, a digital photography course. The event is very casual and a lot of fun—bring your laptop and be prepared to brag about your project to passersby.

2 Scalability (2:00–89:00)

- Scalability issues can really come back to haunt you if you don't plan ahead. If you do plan ahead, however, they can be fun problems to solve.
- A few books you might consider looking over if you're concerned about scalability:
  - *Building Scalable Websites* by Henderson
  - *High Performance MySQL* by Zawodny and Balling
  - *MySQL Clustering* by Davis and Fisk
  - *Scalable Internet Architectures* by Schlossnagle

  Know that all of these tend to take a hand-waving approach, so don't be disappointed if you don't find the in-depth discussion that you're looking for.

2.1 Vertical Scaling

- Thus far, you've been developing your projects on your own laptop or on `cs75.net`. Neither is particularly high-performing, but `cs75.net` at least comfortably supports 140 users with its Virtual Private Server (VPS) rented from ServInt which grants us a decent amount of RAM and disk space. The downside is that the operating system is quite archaic, so installing new packages is really a pain. Next year we'll be ditching this setup altogether in favor of managing the server ourselves.
- In the fall of last year, for CS 50, we experimented with an office hours queueing application that would allow students to line up and submit a question which could be fielded by a staff member. What we didn't anticipate was that with 50+ students at office hours, the server couldn't handle even 50 requests per half-second or second because of the way we had written the code (using Ajax, among other tools). Back to the drawing board we went to solve the problem of scale!

- Recall from last time the `top` command that allows you to examine the processes that are running and the resources that are being used by a Linux system.

- *Vertical scaling*, generally speaking, is the approach of throwing money at the problem. In short, if you have a machine which is suffering under its current load, just buy a better machine! More RAM, the fastest CPU, more L2 cache, higher-RPM hard drives, etc. The upside of this approach is that it requires very little problem-solving on your part—you don't have to fix poorly designed code, for example. What problems arise with this approach, though? There are, of course, real world limits to CPU and RAM, so there may come a time when you can't throw any more hardware at the problem. Not to mention—it's expensive! You will definitely pay a premium for the top-of-the-line hardware and it might not be worth it in the long run.

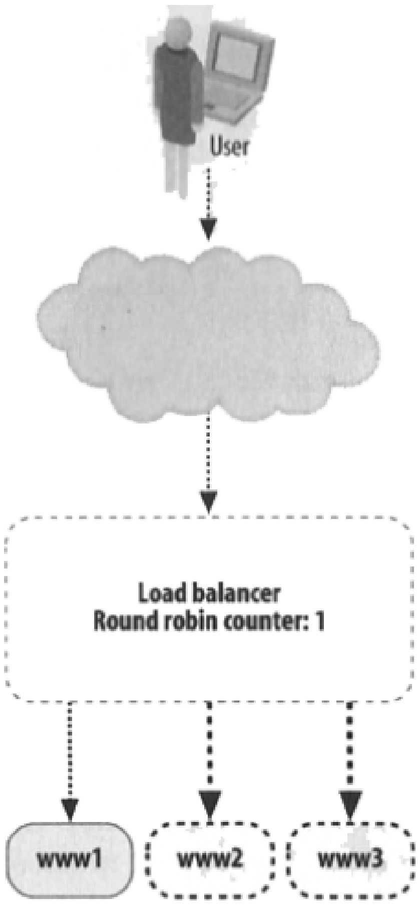
2.2 Horizontal Scaling

- The alternative to vertical scaling is to buy cheaper hardware, but a lot of it. For example, instead of buying one 3-GHz server, perhaps you could buy 2 1-GHz servers. One advantage of this approach is redundancy: if one server goes down, at least the other one will still be up and running, albeit slowly. So we're two hardware failures as opposed to one hardware failure away from being completely down.

- One downside, though, is more labor on behalf of the developer. Take the example of the CSV file you parsed for Project 3 and imagine that it was 40 billion lines long instead of 40 thousand lines long. Just because you now have two servers doesn't mean that your running time for that parsing script will magically be cut in half. Rather, you'll need to perhaps split the file in half so that each server parses part of it. No matter what, you'll have to do a little more work to take advantage of that extra hardware.

- Another problem that arises is the sharing of data between servers. Session data, as we've learned, is stored on the file system of the web server. If a user adds items to his shopping cart on one server and then somehow gets bounced to the other (perhaps the first goes down), then suddenly his shopping cart will be empty. This problem of sharing data applies to database servers as well: if you spread data across multiple database servers, how can you efficiently query for that data? Well return to this problem later.

- If we have multiple web servers and we want to take advantage of them, we need to make sure to balance traffic across them. This is called *load balancing*. One way of implementing load balancing would be with a round-robin approach:



This hearkens back to Lecture 0, in fact, when we discussed DNS. When a user accesses our website, the DNS lookup returns him an IP address associated with that hostname. But if we have multiple servers, then we can have multiple IP addresses associated with a single hostname. So, to balance load, we would simply assign the first user to the first IP address, the second user to the second IP address, and so on, looping back to the first IP address when we run out of servers.

- Actually, the round-robin approach is one we can implement using BIND (Berkeley Internet Name Domain), which is standard on most Linux servers. Basically, we would have DNS records that look like the following:

```
www  IN  A  64.131.79.131
www  IN  A  64.131.79.132
www  IN  A  64.131.79.133
www  IN  A  64.131.79.134
```

By default, BIND will take a round-robin approach when returning a response for DNS lookups. So the first user who requests our domain will be given `64.131.79.131`, the second will be given `64.131.79.132`, and so on.

- In theory, this approach will achieve equal load across all servers. In practice, however, the caching of DNS responses poses a problem. Instead of making a new DNS lookup each time a user navigates to your domain, an ISP will often cache the results of an old DNS lookup so as not to waste traffic. Likewise, your browser might do the same. The problem for your load balancing would arise, then, if a disproportionate number of ISPs and browsers happened to cache a single IP address out of the four you have available. That single server would then experience higher load than the other three.

- Caching on the whole is advantageous because it reduces traffic and optimizes response time. However, in addition to posing a problem to your load balancing, it can pose a problem if your hosting situation ever changes. At the end of the course, we'll ask you to transfer your websites off `cs75.net`. If you remove your files completely from our server and then transfer them to the new one, users who have previously visited your site may hit a dead end because of ISP caching: the old DNS response, the IP address for the course server, no longer has a live copy of the site. To avoid this, you can leave your files on the course server for a few days after you transfer them to the new server. Eventually, the new lP address will propagate to those users who have a cached DNS response and they will be led to the new server. Then you can take down the old site. Of course, this still presents a problem if you have a database-driven dynamic site, as now the new site's database will be out of sync with the old.

- The round-robin approach to load balancing could still result in disproportionate load on a single server if a large number of users who happen to stay on your website for a long period of time are all on the same server.

- Another downside of the round-robin approach is that if one of your servers goes down, then many users will hit a dead end when trying to access your website. And, because of caching, even if you quickly update your DNS records, this dead end might persist for a while.

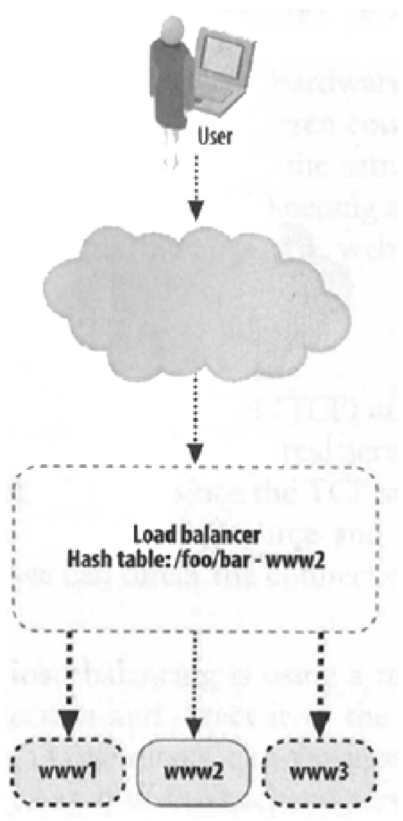
- Of course, the upside of the round-robin approach is that it requires no extra hardware and is very easy to implement.

- Question: it's possible to flush the cache of your browser via the settings or preferences and you can also reboot your router to flush its cache, but you'll have little or no control over your ISP's cache.

- Question: websites technically should have two DNS servers for the sake of redundancy, but we cheat a little bit on this requirement for `cs75.net`.

One of our DNS servers is actually the same machine as our web server—we figure if the web server goes down, it doesn't really matter if we have an extra DNS server handling queries.

- We can also take care of load balancing at the application layer using a full-fledged load balancer:



You may have noticed while ordering pizzas from Domino's, for example, that after a few clicks, you are redirected to a URL beginning with `www1` or `www2`. This is a very simple way of balancing load. The upside of this approach is that if the server hosting `www1` goes down, the `.htaccess` file which is handling redirects on the load balancer can very quickly be updated so as to take `www1` out of the rotation. Of course, if the user has already bookmarked that specific URL, then he'll still be led to a dead end.

- Load balancers can be much more than just redirection devices. More sophisticated load balancers might act as true middle men in the process such that when a request comes into them, they not only channel it to a specific web server, they also channel the response back to the user. As a true middle man, the load balancer can make more intelligent decisions. There's no need, now, to have different usernames for different servers, so users all see the same URL when accessing your site. What's more, if you need users to be mapped to the same server every time they visit your site (to maintain a shopping cart for example), the load balancer can check the cookie before routing the user to a specific server.

- Question: load balancers can even handle SSL if they host the certificate themselves. For that reason, they are sometimes called *SSL accelerators*.
- Of course, the major downside here is that we've re-introduced a bottleneck. If our load balancer goes down, then it doesn't matter that we have three web servers to serve content because none of them are accessible. It's clear that we need to add redundancy to our load balancer set up as well.
- If we add an additional load balancer into our setup, we can actually configure it so that to the outside world, it appears that we only have a single load balancer. The advantage of this is that we don't need to have different IP addresses for our load balancers, which would again introduce the possibility of a dead end if one of those load balancers were to go down and its IP address had been cached by a browser or ISP.

  We'll tie a single IP address to the ethernet card of one our load balancers and leave the other one in passive mode. The passive backup load balancer periodically sends a *heartbeat* to the active load balancer to check that it is alive. If the active load balancer ever goes down, the backup load balancer will immediately take over the IP address and pick up where the dead load balancer left off. The single IP address is said to *float* between the two load balancers.
- The major downside of this approach is that it's labor-intensive. It takes a lot of time, energy, and money to set up! Of course, sites like Amazon can place a very large dollar amount on the business they lose to server downtime, so it's definitely worth it to them. For your small startup website, maybe it's not worth the effort.
- To implement *sticky sessions*, or sessions that persist even if a user is bounced between different web servers, you might use load balancing at the application layer, but you could also use shared storage (e.g. FC, iSCSI, NFS) or cookies (assuming the load balancer can inspect the cookie value and direct the user accordingly).
- Question: realize that NFS can represent a single point of failure if there's only one file server. And, again, adding redundancy is a pain since you have to worry about synchronization.

- A number of options, both software-based and hardware-based, exist for implementing load balancing:
  - Software
    - LVS
    - Perlbal
    - Pirhana
    - Pound
    - Ultra Monkey
  - Hardware
    - Cisco
    - Citrix
    - F5

  Hardware-based load balancers are definitely not cheap. Typically, you'll be required to buy them in pairs, the cheapest being around $100,000. If you're a lowly startup, you may want to go with a free software-based option instead!

2.3 Caching

- For any website in which content that is served to users changes less frequently than it is requested, there exists the opportunity for caching. Although PHP is an interpreted language, it is still compiled before any content is displayed to the user. Without caching, however, it will be compiled each time it is executed. If we can store a compiled version of a file, however, we can cut out this extra step and thus decrease response times. There are a number of free PHP accelerators that implement this kind of *opcode caching*:
  - Alternative PHP Cache (APC)
  - eAccelerator
  - XCache
  - Zend Platform
- Consider the homepage of the course website. The content is "dynamic" in the sense that it's being served up by PHP, yet the welcome message and all of the links and images are static. So why not just store the XHTML output since it can be served up slightly faster? Well, we don't really have to since we don't experience very high load. Sites that do experience high load, however, might opt for this approach; craigslist is one that does. Notice that when you go to a craigslist posting, its file extension is `.html`. Turns out that when you make a post, it is saved as an HTML file which is then added to a master index list of HTML files so it is searchable. By serving up HTML files rather than PHP files, craigslist can reduce load on its servers and ultimately save money.

- One downside of this approach is that there's some lag time before posts are live because the master index list of HTML files isn't updated instantly. Another downside is that updating the appearance of all the site's pages requires changing a large number of files rather than a single PHP header file or the like.

- Even MySQL implements caching for queries. To enable it, you need only add a single line to your `my.cnf` file:

  ```
  query_cache_type = 1
  ```

  What this means is that if you execute a given query more than once, it will remember the result of the query and retrieve it as long as the database hasn't changed. This is certainly compelling when the database has millions of rows.

- One well-known tool which implements caching is called Memcached and is used by Facebook. The general principle is to store the requested content in a variable which is then stored in RAM and perhaps disk. Then when an identical request is made, RAM is first checked, followed by disk, followed finally by the database. Take a look at the snippet of code below to see how it is used:

  ```
  $memcache = memcache_connect(HOST, PORT);
  $user = memcache_get($memcache, $id);
  if (is_null($user))
  {
      mysql_connect(HOST, USER, PASS);
      mysql_select_db(DB);
      $result = mysql_query("SELECT * FROM users WHERE id=$id");
      $user = mysql_fetch_object($result, User);
      memcache_set($memcache, $user->id, $user);
  }
  ```

  Basically, we're first checking the cache to see if the user id has been stored there. If it has, then we skip the next part, which involves a database lookup. Obviously, the cache needs to be flushed whenever the user's data is updated.

2.4 Database Considerations

- What about optimizing our database lookups? In fact, there are database engines specifically designed to help with this. One of them is called the MEMORY database engine. As you might've guessed, it allows you to implement database tables nearly entirely in RAM.

- It's important to do your homework regarding database engines. MyISAM is known to be extremely fast for reads, but not so much for writes. There are upsides and downsides for each engine, depending on the use case.

- One common database setup is called master-slave replication. This is a configuration in which multiple servers can be read from, but only one can be written to. In this way, you can optimize performance by spreading load across multiple servers. The downside, of course, is that you have a single point of failure in your master database server.

  In this configuration, writes to the master server will eventually propagate to the slave servers. In your PHP code, you can maintain multiple database connections, using the one to the master in the case of `INSERT` or `UPDATE` operations and ones to the slaves in the case of `SELECT` operations.

- Another configuration worth mentioning is master-master replication. As you might've guessed, this involves maintaining more than one master database which can handle writes. Each write is then replicated to all the other masters. With this configuration, you can spread database writes as well as reads across multiple servers. Load balancers, in fact, can be used for database servers as well as web servers.

- Another approach is called partitioning. In the early days of Facebook, for example, users accessed a hostname unique to their network—one for Harvard, one for MIT, etc. Presumably, each hostname had its own dedicated web server and database server. Of course, in the event that Facebook wanted to redesign its database schema, it had to update it in numerous places rather than just one. In terms of database servers, vertical scaling is often preferable to horizontal scaling because partitioned data is so difficult to maintain.

- The buzzword for databases and servers these days is high availability, which is just a fancy way of saying that you have built-in replication and failover mechanisms.