Computer Science 75
Spring 2009
Scribe Notes

Lecture 12: May 4, 2009
Andrew Sellergren

Contents

Computer Science 75
Spring 2009
Scribe Notes

Lecture 12: May 4, 2009
Andrew Sellergren

1 Introduction (0:00–2:00)

- The Computer Science Fair will be held on Monday, May 18 between 6:30 and 8:30 PM at a location on Harvard's campus to be announced. It's a chance for you to showcase your final project and mingle with the classmates and staff you haven't yet met!
- Today's lecture will focus on scalability!

2 The Problem of Scale (2:00–10:00)

- You can get away with sloppy code when you only have a few users accessing your site, but what happens when thousands or even hundreds of thousands of users are accessing it? You might end up paying the price.
- In the fall, 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!
- There exist some great virtualization tools out there that make the business of load testing (which might've prevented the above problem) quite simple. ServInt and Amazon's EC2 service are two that come to mind.
- Virtual hosting allows you to pay as you go to use servers which you have root access to. The old model for load testing was to buy new servers which could hammer your current setup. But when you were done, you were left with extra servers that you may or may not have had use for. Now, you can spawn servers on demand and use them only for as long as you need.
- A useful tool for load testing comes free with Apache: Apache Bench. Just type `man ab` from the command line to find out how to use it. You can tell it to spawn multiple threads, all of which will make HTTP requests to your server nearly simultaneously.
- David's position in New York is as a consultant for an advertising network similar to Google Analytics which gathers data from multiple other unaffiliated sites. The more hits those sites get, the more this advertising company's servers are hit, up to as many as 500 million a day and 6000 per second. CS 75's servers would surely crash under this load.
- 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.

- If you tried connecting from your own machine to our server using the hostname `cs75.net` as your MySQL connection parameter, you found that it failed. This is because we have a firewall running on the server that blocks all connections on port 3306, the default MySQL port. This prevents the transmission of your credentials in the clear, which the `mysql_connect()` function does by default. There are ways to encrypt this data before sending it over the wire, but we chose to take the stricter precaution of only allowing database connections from the local server.

3 Horizontal and Vertical Scaling (10:00–45:00)

- Once you've decided how much traffic you'd like your configuration to be able to handle—and perhaps how much you'd like to spend in doing so—you have two general approaches you can take: *horizontal scaling* and *vertical scaling*.

- We've all experienced server downtime at one point or another. Perhaps you tried to access cnn.com during a national emergency or myspace.com during American Idol. Random spikes in load are often termed *The Slashdot Effect*.

- *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. What problems arise with this approach, though? Well, it might be either too late—if your server is being hammered unexpectedly, you won't be able to install all the new hardware in time to handle it—or too early—if you never experience high traffic at all.

- Similarly, vertical scaling is ineffective if it does no more than mask the underlying inefficiencies of your code. At some point, you will butt up against the real-world limitations of your hardware.

- The advantage of vertical scaling, of course, is that it's very clean to implement. If you have a single web server and a single database, then it's very easy to keep track of data. If you have multiple servers over which you're balancing load, how do you retrieve a user's shopping cart if the last time he visited your site, he was routed to a different web server?

- The crude approach to load balancing is to simply hardcode different URLs for each web server. We saw this approach used by Domino's and php.net. What's the problem with this though? If one of the servers goes down, a user might lose his purchase. Not the best solution, really.
- One quick way of increasing the speed with which your site is served to the public is to use a PHP accelerator. What these implement is something called *opcode caching*, which stores a compiled version of your PHP applications so that it doesn't have to be recompiled every time it is requested. Here is a list of PHP accelerators:
  - Alternative PHP Cache (APC)
  - eAccelerator
  - XCache
  - Zend Platform
- Now we'll move from Layer 5, the Application Layer, to Layer 4, the Transport Layer. This is where load balancing takes place. Having multiple servers not only allows us to support many more users, it also introduces redundancy into our configuration so that if one server dies, another can quickly replace it. Not to mention that administratively, we can maintenance a single server without users experiencing downtime on our site.
- How does a load balancer operate exactly? 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. This approach is called *Round-robin DNS*.
- One gotcha with this approach is the issue of ISP caching. If Comcast and many other ISPs all happened to cache the same IP address for your hostname, then the load will be skewed toward that single server. Given a certain amount of randomness, however, this is unlikely to happen.
- The problem with caching, though, is that it might create the illusion of extended downtime. If you take a server offline whose IP address has been cached by various ISPs, then users might be directed to a dead end even when your other servers are up and running.
- One way to ensure that users always have access to their data, such as the items in their shopping cart, is simply to implement a hash table that will map them to the same server every time they visit. Two other approaches are to use shared storage or cookies. Cookies, of course, offload some of the responsibility for file storage to the client. Shared storage such as

NFS, meanwhile, opens up a single disk to an entire LAN so that multiple servers can read from it and write to it.

- So what options are available to you if you want to implement load balancing? A few are listed below:
  - Software
    - LVS
    - Perlbal
    - Pirhana
    - Pound
    - Ultra Monkey
  - Hardware
    - Cisco
    - Citrix
    - F5

  Know that most hardware-based load balancers are quite expensive–on the order of $10,000. There exist free alternatives, however, that can get the job done just as well. In the fall, for CS 50, we used Linux Virtual Server (LVS).

4 Caching (45:00–60:00)

- For any website in which content that is served to users changes less frequently than it is requested, there exists the opportunity for caching. In this way, you won't incur the overhead of repeated database calls. Instead, you can make a single database call and remember the result.
- This, of course, explains why static HTML content can be served up so much more quickly than a webpage which requires a database call. In the first case, you can skip the connection and query steps which are so expensive.
- Interestingly, craigslist still leverages this. 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.
- The downside to this approach is that a lot of extra content (e.g. the HTML tags) is redundantly stored whereas it wouldn't be in a database. Even so, the whole page is not more than a few kilobytes, so the tradeoff is worth it for craigslist's purposes—they can serve up content much faster at the expense of disk space.
- Another downside of this approach, of course, is that content is not dynamic. Craigslist is certainly one of the ugliest websites out there.

- MySQL has a query cache which you can enable simply by adding the following 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.

- For our office hours scheduling in the fall, we implemented our own cache. Instead of making a request to Google Calendar every time a user visited our website, we made the request once and wrote the response to disk. Then when a user accessed our website, we pulled up this file rather than making another request to Google, assuming the data hadn't changed.

5 Database Considerations (60:00–80:00)

- 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.

- A common database setup is called master-slave replication. This is a configuration which duplicates database writes across multiple database servers. Obviously, this is compelling in the case that your database goes down since it ensures that you will have multiple backups. One other reason it's compelling, however, is that it might allow you to spread database reads across multiple servers, thus increasing performance.
- 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 load balance across database writes as well as reads.
- The biggest problem with any configuration is having a single point of failure. If you have multiple web servers but only a single load balancer, then only the load balancer needs to fail for your configuration to break down.
- 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.
- 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.
- One final option we'll mention briefly is that of the MySQL cluster, whose use tends to be discouraged due to its complexity. You can think of it as a database implementation of RAID such that if a single database fails, another can take over for it on-the-fly.