Drupal Architecture at the University of Iowa

Bill Bacher
Unix Admin
University of Iowa
Information Technology Services
bill-bacher@uiowa.edu

Being a (somewhat bigoted) Unix admin, I run Linux on all my personal computers. Because of this, I wasn't able to record my presentation at DrupalCorn 2013 (Windows and Mac only. Man, have I heard THAT before). This is an attempt to capture what I said, taking my notes and expanding them a bit. Hopefully, it helps someone.

Overview (Slide 3)

Pretty much everything in our typical Drupal 'Cluster' is fully redundant. While the diagram only shows one Network Load Balancer, there are actually 2 of them in an Active/Standby arrangement. Should the Active node have any problems, the Standby node takes over seamlessly. The NAS is also redundant.

We don't do any Session Management at the Load Balancer level. A request for a page will likely get half it's content from each web server. We're able to do this since Drupal stores session data in MySQL/memcached.

While we have a 'Replica' MySQL server, we've never tested actually switching over to it. That would require changing the DB Server setting for each site on the cluster. We haven't had need to do it so haven't actually tried it. In theory, we're set up to support it, should it be necessary. I hope we never have to verify that.

Overview (Slide 4)

We had Acquia on campus almost exactly 2 years ago for a training/consulting engagement, and much of our architecture is a result of that. We have 90 sites on our 'Custom' Production Cluster, with no indication of any issues from the servers. We really don't know how high we can push them. We have Development and Test clusters that are nearly identical to the Production cluster to allow the development of sites, modules, and for testing updates. We also have several clusters dedicated to certain types of sites. This is more for isolation and security than performance. There were concerns when we were first planning this about mixing student sites on the same servers as very critical things such as www.uiowa.edu. Putting each group in their own cluster seemed a reasonable solution.

We've just started the process of moving bits of www.uiowa.edu to Drupal. We are using the intelligent routing capabilities of the F5 Network Load Balancers to look at each request and route it to either our existing 'www' servers or to Drupal. This should allow us to gradually move content without any major disruptions.

Servers Specifics (Slide 5)

All our servers are Virtual Machines, running in a VMware High Availability Cluster. This means that in the event a VMware host has a problem, the HA feature will automatically move any guests that had been running on it to other hosts and reboot them. Typically, within 2-3 minutes, everything is back up and running.

We have affinity rules defined so only one server in a particular cluster can reside on a host at one time. This prevents both Web servers being on the same host and both being knocked off line in the event of a problem with that host.

The back end storage is on a NetApp NAS, using 'snap' storage. That means that several times a day a snap shot is taken of the disk image of each server. We can essentially do a point in time restore of the entire server using these images in the event the server is corrupted. We also do normal backups to a centralized backup system to allow file and directory restores.

Our web servers have 1 Virtual CPU and 8 GB RAM. Our database servers have 1 Virtual CPU and 4 GB of RAM.

Apache Configuration (Slide 6)

We're running Red Hat's standard Apache httpd Version 2.2.15. We're really not doing anything special with it. We are running ModSecurity at the strong urging of our IT Security Office. We had a lot of issues with it at first, and had to add quite a few exceptions to rules for various Drupal things (/admin and ajax pages seemed the worst) but once we got all of those worked out, it's been very quite. We might see a few issues if we add a new module that does something different, but really haven't had any ModSecurity issues for quite a while. Drupal and ModSecurity can run on the same server.

We do not run SSL on the web servers. We use the CAS module to authenticate to a central log in server. We essentially send the log in request to that server, which is using SSL, let the user log in there, and get a token back from it saying the user has been authenticated. That relieves us of having to support SSL for 90 different sites. So far, we haven't had anything on a site other than Admin functions that have required authenticated users or secure transport.

One other security thing we do is define the default Apache Vhost to go to a page with only a static html file on it. This way, script kiddies who are running IP scans don't see any indication that Drupal is on this server. If they hit a FQDN that has Drupal, then they'll see it, but this hides Drupal from the majority of drive by attacks. Security through Obscurity might not be a terrific strategy by itself, but it certainly doesn't hurt as one layer of a security stack.

Apache Configuration (Slide 7)

At Acquia's urging, we set up our configuration using both shared and local storage for different parts of the web servers. The theory being disk access would be faster for local storage, so put the critical stuff there.

In SHARED storage, we put the Apache configuration, ModSecurity Configuration and custom rules, the combined, rotated web logs, and, most importantly, the FILES directory, which contains sub directories for each Drupal site. Changes made to anything in this space are immediately available to all web servers. This is an NFS mount from our NetApp NAS.

in LOCAL storage, we have the active Apache logs and all the Drupal files (core, modules, site specific files). These have to be maintained separately on each web server.

Sidetrack – Storage (Slide 8)

Given our VMware based architecture, the reality is that both the 'local' storage and the 'shared' storage are NFS mounts to the NetApp. This is something I want to dig into more when we re-architect these clusters. It could be the VMware hosts have a faster, dedicated storage system than the shared drives use and there really is a performance difference, but I want to discuss that with our VMware and Storage engineers. Having everything in one shared bucket would certainly make things easier.

PHP Configuration – (Slide 9)

We use the standard Red Hat supplied PHP 5.3.3. We try to manage the PHP environment with RPMs to make building and updating easier. We build some RPMs ourselves and get others from EPEL (Extra Packages for Enterprise Linux) and other sources. These are all distributed from our RHN Satellite server.

We are running the 'Alternative PHP Cache' (APC). While we never actually bench marked it, looking at what it does and the statistics it generates, I'm convinced it's helping performance. It essentially stores the compiled PHP code after the first time it compiles it, then uses that compiled, cached code the next time that particular PHP file is called, skipping the compile process. When I look at at the stats page, it reports 100% hit rate. In reality, we typically see 15 million cache hits per day with 3,400 misses. I've never counted, but with 90 sites, and a variety of modules, I wouldn't be surprised to find out we have about 3,400 PHP files.

While this works well for a Multi-Site Drupal install, it doesn't do so well if you have 90 sites and they all have their own code base. If you have even 2,000 files per site, at 80 sites you're looking at 180,000 files to cache. That's going to require a lot more memory than the 256 MB we have allocated to APC.

PHP Configuration (Slide 10)

We're really not doing anything special with our PHP configuration. We've attempted to set some sane limits at the server level, but we do have “AllowOverride” set to all in most of the vhost configurations, so the developers can set things wherever they want on a site by site basis.

We do install phpMyAdmin for managing the back end database. It doesn't work well in a load balanced environment due to keeping session information in local files, so we go directly to one of the web servers to access it. We could manage this at the load balancer level, but given the few people who need to access it, this seemed simpler. I did suggest having one web server point to the Master MySQL server and the other point to the Replica, but I'm really the only one whoever accesses the replica, so there wasn't much interest in this. I use the command line for all my replication management anyway.

MySQL Configuration (Slide 11)

As noted earlier, we're using a Master/Replica MySQL setup. The web servers do all their reads and writes to the Master, and we run the nightly MySQL dumps against the Replica. In theory, we could configure Drupal to write to the Master and read from the Replica, but there hasn't been any need to do that. Performance has been fine.

We're running MySQL 5.5 from the Remi Collet repository instead of 5.1 as provided by Red Hat. Acquia recommended 5.5 for better performance, and we noticed much better replication stability with 5.5. Overall, since switching to MySQL 5.5, our replication process has been very stable. I can update and reboot one server, then update and reboot the other server, and when they're both back up and running, the replication hasn't missed a beat.

MySQL Configuration (Slide 12)

MySQL has actually caused us more problems than any other single part of the LAMP stack. Much of that was our ignorance. We don't have a dedicated DBA for MySQL. I pretty much play that roll. There's a reason I don't get DBA level pay ;-)

The first thing we hit was when using the preferred InnoDB Storage engine in it's default mode, it stores everything in one file. This file has the interesting attribute of never getting smaller. If the developers are having fun creating and cloning databases, the storage file grows to accommodate them. The problem is, when they delete those tables or databases, it never gives up that file space. It can and will reuse that space, but it will never give it back to the file system for other use. The only way we've found to reclaim the disk space is to dump all the databases, shut down MySQL, deleted the ibdata1 file and the ibdata logs, restart MySQL, and import the dump files back into it. This gets you your databases back and only uses the space they currently need, but it's fairly drastic. One side note – the 'mysql' table, by default, uses the MyIASM storage engine so isn't part of the problem. It also doesn't have to be dumped and reloaded if you want to reclaim space, making that process a bit easier.

MySQL Configuration (Slide 13)

The other solution to this problem is to tell MySQL to create a separate file for each InnoDB Table by adding the "innodb_file_per_table" setting to the my.cnf file. Using this, every time a new table is created, a new file is created on the file system to store its data. If the table is deleted, the storage file is deleted, too, freeing up the file space. Possible drawbacks of this include potentially using more disk space for all the individual files as opposed to one big file, and the fact that if you're writing to many individual files, you need to have many file handles open to the OS, and there are limits there. But it makes managing file space much easier than dump/delete/reload.

We're currently using the one file setup for our production server, but the individual file setup for our development and test servers. This seems to be working fairly well.

MySQL Configuration (Slide 14)

Some of the interesting bits from our my.cnf files. The 'sync-binlog=1' in the first stanza of the Master config forces MySQL to make sure it writes each transaction to the transaction log immediately upon execution. The "innodb_flush_log_at_trx_commit=2" entry in the 2nd stanza of the Master config tells MySQL to write every transaction to the disk buffer immediately, but only forces it to flush the buffer to actual disk at no greater than 1 second intervals. This means that in the event of a server crash, we could loose the last second of data (if it hadn't been flushed from the buffer to the actual disk). These two are kind of contradicting each other. All I can say is they evolved at different times and never were really looked at as a whole until I started writing this. This is one area we need to revisit.

MySQL Configuration (Slide 15)

The other disk gotcha we ran into was with the transaction logs themselves. The Master server writes every change it makes to a database to a file named mysql-bin.###### (configurable). The Replica server 'follows' that log in order to know what it needs to do to keep the replica in sync with the master.

A new log is created when they reach 1 GB in size, or when MySQL is restarted. They're removed after a certain amount of time (the default is 3 days, also configurable). The problem we ran into was while they're created based upon activity, they're managed based upon age. In a very dynamic environment, that is an issue.

MySQL Configuration (Slide 16)

We had a series of disk space issues on our MySQL servers when we first started, Between the InnoDB data file never shrinking and transaction logs growing like weeds (err, corn? DrupalCorn, even?), we were constantly running out of disk space. At this time, we were using MySQL for the Drupal Cache. Every time a user did pretty much anything, their cache information was updated, meaning updates to the cache tables in MySQL and more entries in the transaction log. In the early days of our Drupal efforts, between active/curious developers, and having the Drupal Cache point to MySQL, we were seeing 3-6 GB of transaction logs a day. With the default 3 day clean up setting, that meant we could have up to 18 GB of transaction logs on a single server. We hadn't set them up with that much disk space.

Our short term solution was to write a script that we run out of cron every 4 hours (every 2 hours on the production server since it was seeing the most cache traffic) that checks the replication status from the Replica server, then verifies the log file the Master is writing to is the same one the Replica is following. If all this is good, we remove any transaction logs older than the current one. It means we can't go back, but that hasn't been an issue.

This probably isn't necessary today. We've changed things to the point where it might take 2-3 days for a server to generate 1 GB of transaction logs. But, this script also notifies me if there are any abnormalities in the replication process, so I'm still running it.

Memcache (Slide 17)

The real solution to our transaction log problem was switching to memcached for our cache data. This removed all the cache updates from MySQL and moved them to RAM resident memcached. Besides a significant reduction in MySQL activity, the developers reported a 3X performance improvement in page load times with memcached. Everyone was happy!

Each web server also runs memcached. We allocate 4GB of the 8GB of RAM on the servers to memcached. When I checked recently, we were using about 2.5 GB on our production servers.

Memcached is not an overly secure protocol so we limit access to it with IPtables on the servers so only the two web servers can access it via the network.

In the sample configuration file, the only thing we changed was the cachesize setting, increasing it to 4 GB. You could change the port definition to help hide it, but locking it down at the system level seems a better solution.

Memcache (Slide 18)

A sample Drupal configuration for memcached. The two important bits here are if you're running a multi-site environment, it's important that you set a unique value for 'memcache_key_prefix' for each site. The base name for each value cached is the same for every Drupal site. If there isn't a unique prefix, then they start using each others data, creating some interesting mash ups of sites.

The other thing is to make sure you list the memcached servers in the same order in the configuration for each server (assuming you have multiple web servers and multiple memcached servers). While we never tried NOT doing this, everything we read stressed it to the point we simply accepted it. Sort of like “'Don't cross the streams' 'Why?' 'It would be bad'”

Summary (Slide 19)

This reflects our current state. In pulling all the information together for this presentation, there were things I saw that made me scratch my head, things I'll likely go back and revisit. This isn't necessarily the ideal setup for us, and certainly isn't necessarily the ideal setup for anyone else.

We are hoping to move to RHEL7 and Drupal 8 in a year or so. When we do that, everything here will be revisited, viewed through the wisdom we've acquired over 2-3 years of running Drupal. Most of this was based upon Acquia recommendations. While I'm not saying they were wrong, they don't know our environment the way we do.

- Does having Drupal 'local' to the web server really boost performance? It has a cost to it. We have to manage two Drupal installs. Update modules on two servers. Manage two sets of site configs. We're using Jenkins and drush to do this, but sometimes they stumble.
- Do we need two web servers at all? Given VMware can have a server back on line 2-3 minutes after the host crashes, is the complexity of two web servers worth it? Would one server with double the resources be better? I suspect we'll stick with two web servers, but we need to make sure it makes sense. We are successfully running Drupal multi-site installs on a single server for other departments on campus. It can be done.
- What value is our MySQL Replica server providing? Again, with VMware managing the servers, should something happen to the master, it will be back on line much sooner than we could update the DB server definitions for 90 sites. And even if we pointed everything to the replica and it worked, we'd still face the task of getting the master/replica system back in sync. That most likely would require a service outage.