Presentation Summary “High Performance at Massive Scale: Lessons Learned at Facebook” | Idle Process

idleprocess.wordpress.com · November 24, 2009

Recently, we were fortunate to host Jeff Rothschild, the Vice President of Technology at Facebook, for a visit for the CNS lecture series. Jeff’s talk, “High Performance at Massive Scale: Lessons Learned at Facebook” was highly detailed, providing real insights into the Facebook architecture. Jeff spoke to a packed house of faculty, staff, and students interested in the technology and research challenges associated with running and Internet service at scale. The talk is archived here as part of the CNS lecture series. I encourage you to check it out; below are my notes on the presentation.

- Facebook is the #2 property on the Internet as measured by the time users spend on the site.
- Over 200 billion monthly page views.
- >3.9 trillion feed actions proceessed per day.
- Over 15,000 websites use Facebook content
- In 2004, the shape of the curve plotting user population as a function of time showed exponential growth to 2M users. 5 years later they have stayed on the same exponetial curve with >300M users.
- Facebook is a global site, with 70% of users outside of the US.
- Today, there are 1.3B people in the world who have quality

Internet connectivity, so there is at least another factor of 4 growth that Facebook is going after. Jeff presented statistics for the number of users that each engineer supports at a variety of high-profile Internet companies: 1.1M for Facebook, 190,000 Google, 94,000 Amazon, 75,000 Microsoft.

Photo sharing on Facebook:

- Facebook stores 20 billion photos in 4 resolutions
- 2-3 billion new photos uploaded every month
- Originally provisioned photo storage for 6 months, but blew through available storage in 1.5 weeks.
- Facebook serves 600k photos/second –> serving them is more difficult than storing them.

Scaling photos, first the easy way:

- Upload tier: handles uploads, scales the images, sotres on NFS tier
- Serving tier: Images are served from NFS via HTTP
- NFS Storage tier built from commercial products
- Filesystems aren't really good at supporting large numbers of files

Scaling photos, 2nd generation:

- Cachr: cache the high volume smaller images to offload the main storage systems.
- Only 300M images in 3 resolutions
- Distribute these through a CDN to reduce network latency.
- Cache them in memory.

Scaling photos, 3rd Generation System: Haystack

- How many IO's do you need to serve an image? Originally, 10 I/O's at

Facebook because of the complex directory structure.
- Optimizations got it down to 2-4 IOs per file served
- Facebook built a better version called Haystack by merging multiple files into a single large file. In the common case, serving a photo now requires 1 I/O operation. Haystack is available as open source.

Facebook architecture consists of:

- Load balancers as front end requests are distributed to Web Servers retrieve actual content from a large memcached layer because of the latency requirements for individual requests.
- Presentation Layer employs PHP
- Simple to learn: small set of expressions and statements
- Simple to write: loose typing and universal “array”
- Simple to read

But this comes at a cost:

- High CPU and memory consumption.
- C++ Interoperability Challenging.
- PHP does not encourage good programming in the large (at 3M lines of code it is a significant organizational challenge).
- Initialization cost of each page scales with size of code base

Thus Facebook engineers undertook implementing optimizations to PHP:

- Lazy loading
- Cache priming
- More efficient locking semantics for variable cache
- Memcache client extension
- Asynchrnous event-handling

Back-end services that require the performance are implemente in C++. Services Philosophy:

- Create a service iff required.
- Real overhead for deployment, maintenance, separate code base.
- Another failure point.
- Create a common framework and toolset that will allow for easier creation of services: Thrift (open source).

A number of things break at scale, one example: syslog

- Became impossible to push large amounts of data through the logging infrastructure.
- Implemented Scribe for logging.
- Today, Scribe processes 25TB of messages/day.

Overall, Facebook currently runs approximately 30k servers, with the bulk of them acting as web servers.
The Facebook Web Server, running PHP, is responsible for retrieving all of the data required to compose the web page. The data itself is stored authoritatively in a large cluster of MySQL servers. However, to hit performance targets, most of the data is also stored in memory across an array of memcached servers. For traditional websites, each user interacts with his or her own data. And for most web sites, only 1-2% of registered users concurrently access the site at any given time. Thus, the site only needs to cache 1-2% of all data in RAM. However, data at Facebook is deeply interconnected; each user is interested in the state of hundreds of other users. Hence, even with only 1-2% of the user population at any given time, virtually all data must still be available in RAM.
Data partitioning was easy when Facebook was a college web site, simply partition data at the level of individual colleges. After considering a variety of data clustering algorithms, found that there was very little win for the additional

complexity of clustering. So at Facebook, user data is randomly partitioned across indiviual databases and machines across the cluster. Hence, each user access requires retrieving data corresponding to user state spread across hundreds of machines. Intra-cluster network performance is hence critical to site performance. Facebook employs memcache to store the vast majority of user data in memory spread across thousands of machines in the cluster. In essence, nodes maintain a distributed hash table to determine the machine responsible for a particular users data. Hot data from MySQL is stored in the cache. The cache supports get/set/incr/decr and

Initially, the architecture needed to support 15-20k requests/sec/machine but that number has scaled to approximately 250k requests/sec/machine today. Servers have gotten faster to keep up to some but Facebook engineers also had to perform some fundamental re-engineering of memcached to improve its performance. System performance improved from 50k requests/sec/machine to 150k to 200k to 250k by adding multithreading, polling device drivers, stats locking, and batched packet handling respectively. In aggregate, Memcache at Facebook processes in 120M requests/sec.

One networking challenge with memcached was so-called Network Incast. A front-end web server would collect responses from hundreds of memcache machines in parallel to compose an individual HTTP response. All responses would come back within the same approximately 40 microsecond window. Hence, while overall network utilization was low at Facebook, even at short time scales, there were significant, correlated packet losses at very fine timescales. These microbursts overflowed the limited packet buffering in commodity switches (see my earlier post for more discussion on this issue).

To deal with the significant slow down that resulted by synchronized loss in relatively small TCP windows, Facebook built a custom congestion-aware UDP-based transport that managed congestion across multiple requests rather than within a single connection. This optimization allowed Facebook to avoid the, for example, 200 ms timeouts associated with the loss of an entire window’s worth of

data in TCP.
Authoritative Facebook data is stored in a pool of MySQL servers. The overall experience with MySQL has been very positive at Facebook, with thousands of MySQL servers in multiple datacenters. It is simple, fast, and reliable. Facebook currently has 8,000 server-yearas of runtime experience without data loss or corruption.
Facebook has learned a number of lessons about data management:

- Shared architecture should be avoided; there are no joins in the code.
- Storing dynamically changing data in a central database should be avoided.
- Similarly, heavily-referenced static data should not be stored in a central database.

There are a number of challenges with MySQL as well, including:

- Logical migration of data is very difficult.
- Creating a large number of logical dbs, load balance them over varying number of physical nodes.
- Easier to scale CPU on web tier than on the DB tier.
- Data driven schemas make for happy programmers and difficult operations.

Lots of examples of Facebook's contribution back to open source here.

Given its global user population, Facebook eventually had to move to replicating its content across multiple data centers. Facebook now runs two large data centers, one on the West coast of the US and one on the East coast. However, this introduces the age-old problem of data consistency. Facebook adopts a primary/slave replication scheme where the West coast MySQL replicas are the authoritative stores for data. All updates are applied to these master replicas and asynchronously replicated to the slaves on the East coast. However, without synchronous updates, consecutive requests to the same data item from the same

user can return inconsistent or stale results.
The approach taken at Facebook is to set a cookie on user update requests that will redirect all subsequent requests from that user to the West coast master for some configurable time period to ensure that read operations do not return inconsistent results. More details on this approach is detailed on the Facebook blog.
Areas for future research at Facebook:

- Load balancing
- Middle tier: balance between programmer productivity and machine efficiency
- Graph-based caching and storage systems
- Search relevance via the social graph
- Object discovery and ranking
- Storage systems
- Personalization

Jeff also relayed an interesting philosophy from Mark Zuckerberg: ”Work fast and don’t be afraid to break things.” Overall, the idea to avoid working cautiously the entire year, delivering rock-solid code, but not much of it. A corollary: if you take the entire site down, it’s not the end of your career.
idleprocess.wordpress.com · November 24, 2009