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

Published November 24, 2009 academia , cloud , data center , networking , research , storage 56 Comments

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.

Site Statistics:

- 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.

Site Architecture
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.

Memcache

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 multiget/multiset operations.

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.

Incast

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 Storage

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.

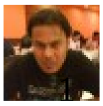
Lots of examples of Facebook’s contribution back to open source here.

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.

56 Responses to “Presentation Summary “High Performance at Massive Scale: Lessons Learned at Facebook””


1. **Abhinav Singh** December 7, 2009 at 11:11 am

Facebook technology and advancement is simply laudable, and their contribution to open source development make it all look excellent

Reply

2. **mahmud ahsan** December 7, 2009 at 1:04 pm

Nice article. Thanks for sharing.

Reply

3. **f.baube** December 7, 2009 at 2:45 pm

SkyNet, thy name is Faceborg

Reply

4. **alexis** December 8, 2009 at 5:49 am

What is a ‘feed action’ and why are there so many of them per day?

Reply

5. **aminvahdat** December 8, 2009 at 7:56 am

A feed action is an update to your profile information (new status update, new picture, etc.) or an access to a set of feeds to compose a page (e.g., when you go to Facebook’s home page and retrieve updates from your social network).

Reply

6. **Jeff** December 8, 2009 at 9:34 am

Can you explain a bit more about the “graph based caching and storage systems?” What do they have in mind? Any links to this stuff?

Thanks,
Jeff

Reply

7. **aminvahdat** December 8, 2009 at 10:54 am

The question is whether certain data (profiles, updates, pictures, etc.) can be cached together on the same machine for locality of access. For example, if all the updates for your friends were stored together on one computer, then the system could be architected around many fewer internal RPC’s (currently 100s of machines have to be accessed to compose an HTTP response). The challenge lies in the limited locality in the social graph.

Reply

8. **Alexander Ainslie (@AAinslie)** December 22, 2009 at 2:52 pm

I am curious, have you guys looked into Neo4J?

9. **Fisher** January 11, 2010 at 1:27 am

Thanks a lot for sharing!
It’s a great idea for using social graph to help cache system.
But I don’t really get why locality is limited in social graph.
Would you plz explain it?

Thanks!
Fisher

10. **aminvahdat** January 11, 2010 at 7:58 am

If profiles were to be co-located based on the social relationship of any one person, it would violate locality for some other set of users. In essence, there are not enough clusters of well connected social relationships (with few connections outside the clusters) to make co-placement a sufficiently big enough win.

11. **Raul Macias** December 8, 2009 at 3:29 pm

Very interesting and helpful at the same time. Thanks for sharing it.

Reply

12. **Dave** December 8, 2009 at 6:15 pm

Awesome article. Love reading articles on scalability.

Reply

13. **akangaziz** December 8, 2009 at 9:19 pm

Thx Vahdat.. very nice article.. great!

Reply

14 **KabarMadura.Com** December 8, 2009 at 9:44 pm

really cool, thanks this part very helpful to us … to implementation to our Social Media Applications.

but one question for u ?
how about query optimization on apps ? are we must synchronize with Good Database Architecture ?

thanks aminvahdat
We are from Madura – Indonesia

Reply

15 **Sundar** December 8, 2009 at 9:52 pm

Hello,

Thanks for sharing the architecture, statistics and technical details of Facebook. Great post.

Yours,
Sundar

Reply

16 **Mike Clements** December 9, 2009 at 10:28 am

250k requests/second/machine is only 4 microsecs per request. On a 3GHz server that’s only 12,000 clock cycles.

This would seem impossible for HTTP GET requests. Even if the requests are handled in memory, to maintain an average of 4 microsecs per request, a cache hit ratio of 99.6% (1 hit in 250 misses the cache) would still give less than 1 millisec for the hit that missed.

Can you explain what that number means and how they achieve it?
Thanks -Mike

Reply

17 **aminvahdat** December 9, 2009 at 10:47 am

Good question. I of course cannot reply authoritatively but the misses can be handled asynchronously without affecting the other requests that are proceeding in parallel with hits in main memory. Also note that each machine likely has at leaast four cores, so this would take the budget per request to ~50k cycles.

Reply

18 **Mike Clements** December 10, 2009 at 9:52 am

Good point – multicore needs to be considered. An 8 core machine would give about 32 micros per request. Even that isn’t much!

I’m trying to find a way to make sense of 250k requests/sec/machine which seems very high. Suppose a request that hits the cache can be processed in 10 micros, and one that requires actual processing takes 1 milli (100x slower). In that case to get 32 micros average, each miss needs 44 hits; the cache would need a 98% hit ratio.

Those numbers seem very tight. If they actually achieved anything in this ballpark, that is very impressive.

19 **aminvahdat** December 10, 2009 at 9:59 am

Once again note that the miss can be handled asynchronously. The thread waiting for the read response need not block waiting for the response. Other requests that likely will hit in the cache can proceed to effectively perform latency hiding.

Having said that, the disk can only handle so many misses a second. With 5 ms for typical random seek time and, say, 4 disks per machine, there can only be ~800 misses/second/machine. Facebook achieves such high hit rates by throwing a lot of DRAM at the problem spread across the infrastructure.

20 **Mike** December 22, 2009 at 12:48 pm

You can also use solid state drives instead of regular hard drives, improving access times to like 0.1 ms. I heard that some site (possibly MySpace) exclusively uses SSDs for its data tiers.

21 **Mike Kingston** December 21, 2009 at 10:33 am

Yea I don’t buy that 250K number….

22 **whatislife** December 27, 2009 at 12:18 pm

The 250K requests/sec/machine with memcache will be either TCP or UDP not HTTP GET requests; I guess Facebook took the UDP route. And 250K requests/sec/machine indicate the Throughput and should not be used to determine latency; the latency in term for the machine making the request may be higher. With 8 cores per machine and careful optimization I think this is hard but not impossible.

Reply

23 **Pandian** December 9, 2009 at 10:30 am

Thanks… Lots of Questions in mind.. But the architecture is Mind Blowing..

Reply

24 **Vijay** December 9, 2009 at 11:07 am

Hi Amin,
Thank you for the wonderful notes. You’ve done a great job explaining the details. However, I would like to check out the video. It doesn’t seem to be available at the site you listed. Could you please check the link/content again?

thanks,
Vijay

Reply

25 **Robin Majumdar** December 9, 2009 at 3:50 pm

Vijay – are you referring to the webcast by Jeff Rothschild at http://video-jsoe.ucsd.edu/asx/JeffRothschildFacebook.asx ? It worked fine for me just a few moments ago. If you’re reading this article using Firefox, it will try opening it as a separate app using Windows Media Player…

Reply

26 **Vijay** December 14, 2009 at 2:20 pm

Thanks Robin.
That is the link I was referring to. I was not able to open it with windows media player. The error I get is “the server might not be available”.

thanks,
Vijay

27 **Jonathan Collins** December 10, 2009 at 3:05 pm

“PHP does not encourage good programming in the large”

Is that PHP? Or is it this?

“Work fast and don’t be afraid to break things.”

Reply

28 **Vivek Gupta** December 13, 2009 at 9:42 pm

Great walk through article.

Reply

29 **اس ام اس عاشقانه** December 15, 2009 at 6:54 am

the blog is good i like it very mcuh

Reply

30 **Sebastian** December 20, 2009 at 7:14 am

quote:
PHP does not encourage good programming in the large (at 3M lines of code it is a significant organizational challenge).

Most languages do not do this. Maybe the growing business did not take this challenge serious enough! Aka inves enough ressources in this part of the job.
It’s always a tradeoff and a hard thing just to post the “organizational challenge” without mentioning the gains 😉

Reply

31 **UX-admin** December 20, 2009 at 9:11 am

“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.”

Stereotypical “hack-it-’till-it-works” attitude. The antithesis of everything a professional engineer would stand for.

And they want people with this kind of mentality to work on Opensolaris; it’s disgusting.

Reply

32 **Matt** December 20, 2009 at 4:24 pm

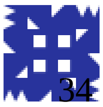
If there are no “there are no joins in the code” – then how are the DB tables normalized? Or is it not normalized for performance reasons and there is a lot of redundant data floating around?

Reply

33 **aminvahdat** December 20, 2009 at 5:43 pm

I imagine that per-query performance trumps total data storage in the Facebook architecture so likely there is some redundant storage in “pre-joined” tables.

Reply


34 **Stefan Parvu** December 21, 2009 at 5:14 am

> 15-20k requests/sec/machine but that number has scaled to
> approximately 250k requests/sec/machine today.

Wonder if this is simple trash or something real ?
Might be useful to ask Facebook folks how can they support
such high number of req/sec per server ?

Scary point of view 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.”

Probable his goal was to make money not a quality build in the first place. However, I see same approach as well applied to
other web sites.

This sucks big times. The current web solutions are
poor tested, and basically they dont have a clue
about how many users they support or how their application
scale. Performance testing is somehow a confusion teritory for
such folks where they fix a problem with dozens of builds all
over again.

Consistent performance testing and scalability analysis do require a bit or work and a more engineering approach.

http://www.perfdynamics.com/iBook/gcap.html
http://www.perfdynamics.com/Classes/Outlines/guerilla.html

Enjoy

Reply

35 **aminvahdat** December 21, 2009 at 6:58 am

There has been lots of reaction to the point of “working fast.” While there are obviously lots of tradeoffs to be considered, the alternative of “working scared” would likely not have resulted in the transformative impact that Facebook has had. The fact of the matter is that we do not yet have the tools to build a fully robust site that scales from 0-350M users in a cost effective manner. This is especially true when you throw in novel access patterns with no data locality across a multi-petabyte dataset. There are a few companies who may have known how to do this “better”, but there is no book that describes how and certainly the necessary software tools were developed in house over a period of years.

As a company matures (as Facebook is), there will naturally be more of a shift toward “process” rather than “outcome”. But it’s not clear to me that a web startup out to change the world with a new paradigm can afford to focus on engineering the system perfectly from day one. Hiring the necessary people to do so would be prohibitively expensive.

Reply

36 **Stefan Parvu** December 21, 2009 at 7:39 am

There are enough tools/methods to carry on robust performance testing and ensure that certain piece of software scales well.

Problem is software is treated with no respect and there are
no engineering methods in ensuring its quality process. People
just produce big amounts of code. That’s it.

When it comes to test and ensure whatever was created is robust and scales well a lot of confusion raises. And then you simple add more servers to survive… Why ? Because from day 1 Performance was a bastard child.

USL Universal Scalability Law is a clear method, low cost which ensures whatever you test is mathematical valid and correct. That’s the starting point in analyzing the performance of an application/site. Consider reading that book, it is very educative and have a direct approach how to handle the performance analysis process.

If we will build bridges and roads like we build today web sites and software I would rather use my MTB Moto in forest.

Cheers

37 **Scott** December 21, 2009 at 5:42 am

“PHP does not encourage good programming in the large (at 3M lines of code it is a significant organizational challenge).”

Clearly never done any enterprise PHP development. Code management isn’t a function of the language. You can write spaghetti or poorly structured code in ANY language. Developing good code requires a well-defined set of guidelines and constant peer-driven code review.

These are basics that should be followed in any large project, regardless of the language chosen. I’ve seen horrific Ruby/Perl/C++/C#/{insert language here}.

Reply

38 **Php** December 21, 2009 at 5:53 am

Your blog is running PHP! Oh my! =P

Reply

39 **Evan** January 26, 2010 at 4:50 pm

4 trillion feed actions per day works out to 13,000 per user per day. A little high, don’t you think? Probably more like 4 billion? Still a big number, just not THAT big…

Reply

40 **Miraç** January 24, 2011 at 11:18 am

Really cool!

Thank you for nice article =)

Reply

1 **Peter Van Dijck’s Guide to Ease** Trackback on December 7, 2009 at 3:06 pm
2 **Knowtu » links for 2009-12-07** Trackback on December 7, 2009 at 6:07 pm
3 **links for 2009-12-07 « Bloggitation** Trackback on December 7, 2009 at 11:04 pm
4 **links for 2009-12-09 | @neilmiddleton** Trackback on December 9, 2009 at 8:05 am
5 **Интересные ссылки №197 - max - блог разработчиков** Trackback on December 11, 2009 at 3:19 am
6 **The Environmental Impact of PHP Compared To C++ On Facebook | JetLib News** Trackback on December 20, 2009 at 9:32 am
7 **Facebook Infrastructure – Number of Servers @Facebook and more « Social Mode** Trackback on December 20, 2009 at 9:52 am
8 **Facebook的PHP代码对环境的冲击 « 每日IT新闻，最新IT资讯，聚合多站点消息，保证你与世界同步** Trackback on December 20, 2009 at 9:28 pm
9 **finally! facebook & its 30,000 servers « meditashunz** Trackback on December 22, 2009 at 10:51 am
10 **Web-based Application Scalability | Project management and time tracking blog for web designers and small business :: the Intervals Blog by Pelago** Trackback on December 29, 2009 at 11:59 am
11 **OpenQuality.ru | Декабрьская лента: лучшее за месяц** Trackback on December 31, 2009 at 12:13 pm
12 **Die Facebook Serverfarmen | Webmaster, Security und Technik Blog** Trackback on January 4, 2010 at 3:12 am
13 **ScribeMedia.org | | How Green is Your Code** Trackback on January 4, 2010 at 5:02 am
14 **Facebook的PHP代码对环境的冲击 - 网来网去-http://www.webcomgo.com专注互联网分析、SEO、电子商务、管理营销、GTD、生活日志 爱皇冠 乐淘淘** Trackback on January 4, 2010 at 5:46 pm
15 **Die Facebook Serverfarmen « Blogger City** Trackback on February 1, 2010 at 9:57 am
16 **PortLand Code Release « Idle Process** Trackback on May 18, 2010 at 10:41 am

Leave a Reply