CS 542 Research Report

Facebook: Data Storage Solutions and Evolution

By

Zhongyuan Fu

Rob Proulx

Yupu Song

6 Mar 2015

Abstract:

An examination of Facebook's database challenges and solutions is presented. The different major database structures in use at Facebook and their technologies are examined, presented with regard to the growth- or project-related reasons to use them.

Table of Contents

Introduction

Scope

Originally intended to be survey of how various tech companies have evolved their data solutions as they grew, this report instead focuses specifically on Facebook from that perspective. Facebook's

resources and their specific data challenges make that company an extremely interesting specimen to examine from contexts of evolution and overall database solutions.

The original database for Facebook was a MySQL solution. This report will examine this original MySQL stack in terms of steps taken to keep its performance high, as well as work for the future. Additionally, we will discuss the other database solutions added as Facebook evolved both from a user and data size point of view, and from a features point of view.

The Overall Picture

With over 1 Billion active users[1], Facebook may be the most popular and visible social media outlet in existence. Facebook allows users to communicate, to share content, to interact through shared online space or through various applications. All of this content, and the interconnectivity of them require immense data solutions.

In addition to the huge data solutions required to drive the user-facing features, the need to monetize all of this interconnectivity creates the need to store any click or user action that could generate ad revenue or useful trends when aggregated. Satisfying this need requires a massive data warehouse. While this data warehouse doesn't necessarily have the same real time performance challenges of the user-facing content, the pure size appears staggering.

Figure 1 shows a rough functional map of the Facebook data footprint. A dividing line shows the separate between which components are user-facing, and which are backend datastore. All of these major solutions are open source. Scribe, Hive and Cassandra (not shown in this high-level plot) were even developed by Facebook.

[1] http://newsroom.fb.com/company-info/

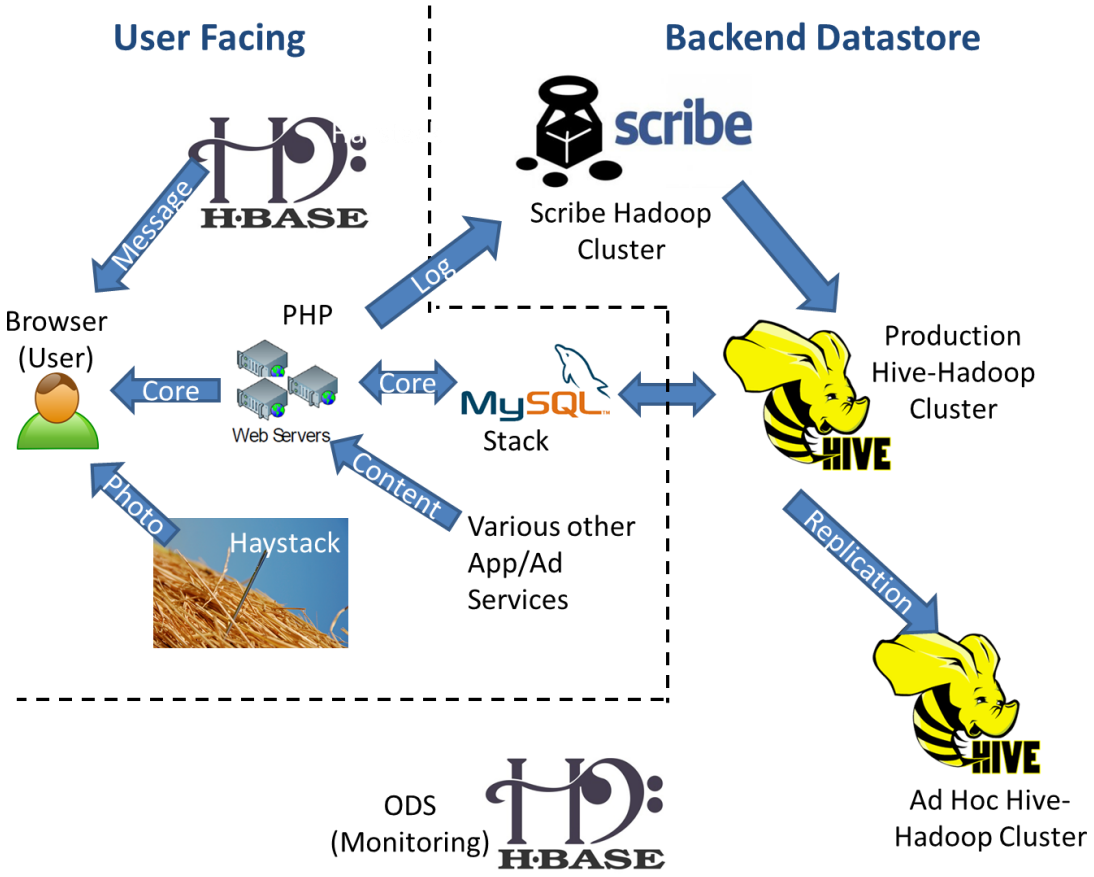

Figure 1—Facebook Database Map

These technologies will be discussed in more detail further, but Table 1 below summarizes the various components, their platforms and uses.

Facebook uses four fundamental database solutions to provide their required functionality. Table 1, adapted from Borthakur , 2012[2], summarizes these database functions, solution technology, and bottleneck factor.

Table 1—Facebook's Database Technologies

Database Function:	Solution	Bottlenecks
Facebook Graph	MySQL Stack	Random Read IOPS
Facebook Messages / Time Series Data	HBase	Write IOPS and storage capacity
Facebook Photos	Haystack	Storage capacity
Data Warehouse	Hive, Scribe (Hadoop)	Storage capacity
Monitoring	ODS (HBase)	Write IOPS

Facebook Graph represents the core Facebook functionality from a user point of view, such as Wall posts, likes, comments, etc. The Facebook API allows developers access to this Graph functions. As will

[2] Borthakur, Dhruba, "Petabyte Scale Data at Facebook," XLDB Conference at Stanford University, Sep. 2012

be investigated in detail further, Graph runs on an advanced and massive MySQL stack. Facebook describes this database as very heavily biased toward reads, thus the primary performance limitation is Random Read IOPs.

When Facebook added Messaging and their timeline functions, they deployed it on HBase for a number of reasons, not the least of which was dealing with the sheer magnitude of the content.

Facebook decided to specifically segregate their photo media content into a separate data store. For this purpose they chose Haystack[3]. The massive size causes the performance limitation to be storage capacity.

As Figure 1 shows, several backend functions occur, transparent to the user. Scribe collects all of the web server action details, aggregates it and delivers it to the Production Hive Store. On this store regular reports (revenue based, etc.) are run. This store is replicated to another Hive store that is used for Ad Hoc and irregular reporting and analysis.

Another Hadoop tool constantly running is ODS, Facebooks hardware and software monitoring system, running on HBase.

[3] Beaver et al.

MySQL and The Univeral Database UDB:

Until fairly recently in Facebook's history, UDB stood for User Database. Recently, though, it has been rebranded at the Universal Database. This database is also Facebook's core and its "Graph," which it exposes to applications through its API.

The UDB contains the main data store about Facebook's users. Each relationship, each comment, each "like," and the text of each post are stored here.

Facebook has larger databases, but in the order of single digit Petabytes, the UDB stands as an impressive achievement. The UDB is also recognized as the largest MySQL data store on the planet.[4]

Why MySQL?

Originally the answer to Why MySQL was probably that it was a well performing database system that was open source, so free. As it grew, however, some have been critical of Facebook's MySQL store as a viable solution, as it was not developed to be deployed at Facebook's massive scale. Most famously Prof. Michael Stonebraker referred to Facebook as "Trapped in MySQL" and that this untenable situation was "a Fate Worse than Death."[5] He was also critical of Facebook's use of a cache layer in front of MySQL, and suggested that a NewSQL solution such as VoltDB would be a better fit for their usage, reducing their server count.

But Facebook has reasonable rationale to stay with MySQL, and others agree with them. The folks at Scalebase.com actually drafted their imagined rebuttal letter to Stonebraker from Facebook.[6] They raise good reasons for Facebook's choice: MySQL is a mature, tested solution, as opposed to an unknown new solution like VoltDB, and that a new databases technology would mean new and expensive DBAs to manage it, to name a few.

Facebook themselves put forth their own reasons to use MySQL. Aditya Agarwal, Director of Engineering at Facebook stated that MySQL is "really quick."[7] He further stated that MySQL was good at data reliability, and that they had never lost data, at least as of 2008. Also mentioned was that MySQL was

---

[4] Callaghan, "MySQL at Facebook."
[5] Harris, Derrick. "Facebook trapped in MySQL 'fate worse than death'"
[6] Campaniello, Paul, "Facebook IT Responds to Stonebraker," 11 Jul 2011
[7] Agarwal, Aditya, "Facebook: Science and the Social Graph," QCon SF 2008

easier to scale CPUs on the web tier than other solutions. Also, open source means no license cost, so Facebook can spend more of their money on hardware.

Beginnings:

While reference materials regarding Facebook tend to focus on the present massive scale of their solutions, some information about their initial deployment can be found. Founder Mark Zuckerberger initially used a fairly standard "LAMP" configuration[8,9], namely Linux OS, Apache HTTP server, MySQL database, and PHP webpage generation. Additionally, Facebook used memcached early on as well, to help with performance and the MySQL sizing[10].

Dealing with Growth:

As the server load grew, Facebook has taken several steps to keep the availability and performance of their MySQL database high.

1. Add MySQL shards and memcached servers

While Facebook is vague about exactly how many servers it uses, some older data is available. In April of 2008, Facebook had 1800 MySQL servers and 805 memcached servers.[11]Three years later, Facebook had 4000 MySQL shards, and 9000 memcached servers[12] showing huge hardware growth.

All the data for a single user is stored on one shard, so as many operations as possible can be performed without going to multiple shards.[13]

2. Optimize Usage for Their Data

Facebook uses MySQL primarily as a <key, value> store. Most of keys are global identifiers of objects and most data access is based on these global identifiers. They claim that it is pretty much impossible to do logical user data migration because of the size and how random the data is. Facebook solved this problem by creating a large number of logical databases and loading balance them over varying number of physical nodes.

Facebook uses no joins in production. Both because of performance reasons and that they don't make logical sense for their queries.[14]

3. Develop tools to manage the size: MySQL Pool Scanner (MPS), discussed in detail below
4. Drive bug-fixes and patches to help with robustness, availability and testing and debug.
5. TAO caching, discussed further below.

---

[8] Barrigas, Hugo, "Overview of Facebook Scalable Architecture," ISDOC '14 Proceedings of the International Conference on Information Systems and Design of Communication, pp. 173-176.
[9] Nishtala, Rajesh and Venkat Venkataramani, "Scaling memcache at Facebook,"
[10]Barrigas, Hugo, "Overview of Facebook Scalable Architecture,"
[11] http://www.adweek.com/socialtimes/facebook-infrastructure-up-to-10000-web-servers/211874
[12]https://gigaom.com/2011/07/07/facebook-trapped-in-mysql-fate-worse-than-death/
[13] Callaghan, "MySQL at Facebook."
[14] Agarwal, Aditya, "Facebook: Science and the Social Graph," QCon SF 2008

6. Advanced development with WebScaleSQL, see below.

Sharding and MySQL Pool Scanner:

Sharding

In general, sharding splits tables horizontally, reducing indexes and workloads or for duplicating data.[15] Figure 2 shows a schematic of how a single database server might be divided.[16] The server may run two instances of MySQL. Each instance may have many shards.

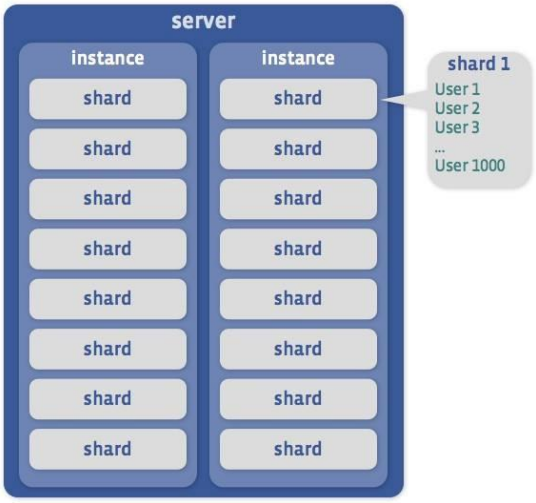

Figure 2—MySQL Server Divisions

In order to ensure high availability and performance, each instance has several copies present on other servers. These copies often reside in a different physical location. The duplicates ensure that if one copy fails, the data can be accessed somewhere else. Further, with instances hosted in various geographical locations, using a local data source for queries can increase speed.

Overview of MySQL Pool Scanner (MPS)[17]

Facebook has created a tool to let the system run by itself with little human intervention. This tool, called MySQL Pool Scanner, can do "nearly everything a conventional MySQL Database Administrator (DBA) might do so that the cluster can almost run itself." [18] Not only does MPS handle availability but, it also lets administrators perform operations such as copy the entire Facebook dataset with a single command.

---

[15] http://en.wikipedia.org/wiki/Shard_%28database_architecture%29

[16] Priymak, Shlomo, "Under the hood: MySQL Pool Scanner (MPS)"

[17] Ibid

[18] Harris, Derrick, "Facebook tells how it makes its MySQL cluster 'almost run itself'",

The regular operation of MPS relies on two “building block” operations: creating a copy/replacing a server, and promoting a master instance.

When a replacement is required, MPS creates a copy of an instance on a new host and then removes an instance after the copy is completed. Figure 3 shows this replecation procedure.

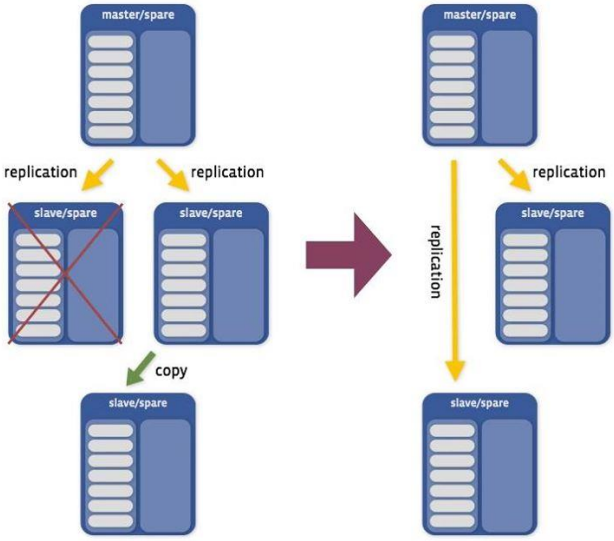

Figure 3— Creating a Replication on a Different Host

For promoting a master instance, MPS chooses a target that is aimed to be promoted, then it stops writes to the replica set and points the new writes to the master.

All the database hosts are connected with MPS, holding both current and past MPS copy operation. A server can collect data about itself, triage the host for problems, and make sure the the server is registered and contains up-to-date metadata in the central repository and so on when it functions.

When MPS meets a failure, it will deal with that problem in the background, perform maintenance operations in every aspect, or even copy an entire Facebook data set at a new data center if needed.

MPS allows Facebook to manage its database system in an automatic way. With this tool, engineers at Facebook are able to be pay attention to more challenging work.

TAO Global Cache

Prior to the advent of TAO in approximately 2013, web servers accessed the MySQL stack directly, using memcached as a look-aside cache. Now TAO is a single geographically distributed cache layer inserted between the web servers and the MySQL stack, significantly increasing performance.[19]

.

[19] TAO: Facebook’s Distributed Data Store for the Social Graph

WebScaleSQL:

In March 27 2014, Facebook, Google, LinkedIn and Twitter announced that they would be together to develop WebScaleSQL, an open source database management system created as a software branch of MySQL 5.6.[20] As its name suggests, WebScaleSQL is a custom version of MySQL designed for large scale Web companies.

How large a database can WebScaleSQL hold? The engineer from Facebook, Stephan Greene, shared his expectation: "We aim to handle more than 1.23 billion people who use Facebook to share and to connect with each. Our goal, in launching WebScaleSQL, is to enable the scale - oriented members of MySQL community to work more closely together in order to prioritize the aspects that are most important to us."[21]

The work Facebook has done :

Facebook has improved and optimized some WebScaleSQL scanning performance and has made changes on avoiding potential problems which may cause error on the system.

The work Facebook is working on

Contributing an asynchronous MySQL client; preparing to move Facebook's tested database into WebScaleSQL; adding the Logical Read-Ahead mechanism that Facebook has proven it can speed the full table scans.

What to expect in the future

Facebook will keep all the WebScale work open, and continue to follow the most up-to-date upstream version of MySQL. Also, Facebook is willing to collaborate with others to work together in a more efficient and transparent way that will benefit all the people.

Cassandra and Inboxes:

Prior to the development of Cassandra, Facebook used its MySQL database to store and query inbox content. Inbox searching enables users to search through their Facebook Inboxes. This application requires massive write throughput, on the order of billions of writes a day and beyond as the userbase scales.[22]

[20] https://github.com/facebook/mysql-5.6
[21] Greene, Steaphan, "WebScaleSQL: A collaboration to build upon the MySQL upstream",
[22] Lakshman, Avinash and Prashant Malik, "Cassandra - A Decentralized Structured Storage System"

In addition to performing this Inbox search function, Facebook had several goals in mind as they designed Cassandra. Cassandra was to scale easily as the data volume scaled. The system was to be reliable, using off-the-shelf commodity type hardware where components fail somewhat routinely.

While these goals of size are challenging they are helped by the fact that the data model Cassandra provides was not intended to be fully relational, but rather extremely simple.

To accomplish their availability and scalability goals, a circular node connection system is used. Figure 4 shows how the client contacts one node, and that node automatically replicates its data to several other nodes in its circle.

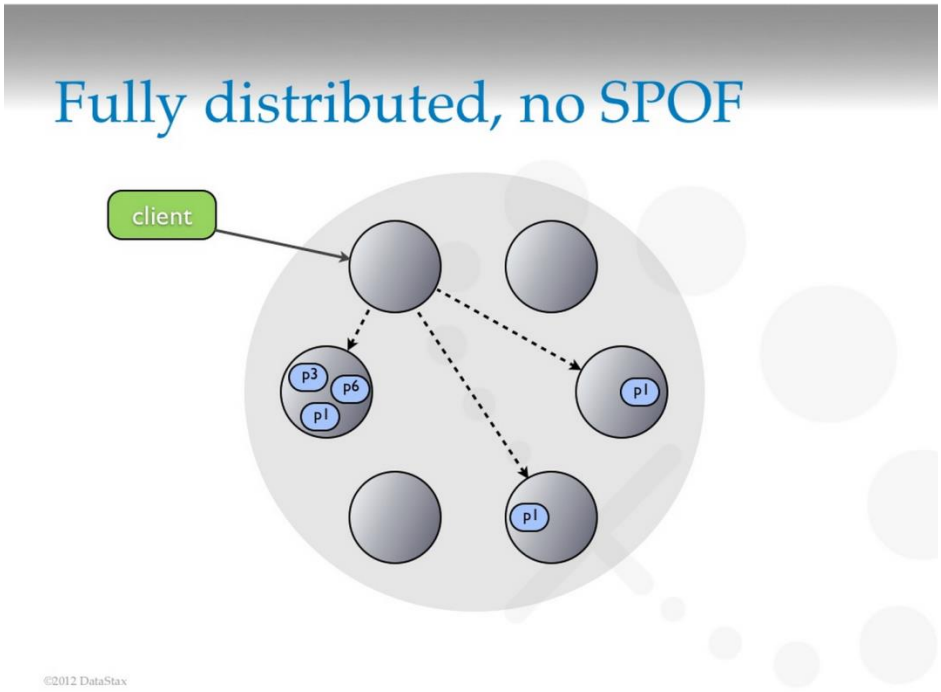

Figure 4—Cassandra Circular Node Replication [23]

Interestingly, Facebook decided to move on from Cassandra in lieu of HBase for their inbox search.

But, this was not the end of Cassandra for Facebook. After Facebook acquired Instagram, they switched to Cassandra for the data store. Instagram's Redis store was not doing a good job of keeping up with storing all of the diagnostic and administrative data alongside the content to facilitate security and site integrity. The Instagram engineers felt this operation was a sweet spot for Cassandra.[24]

[23] Cassandra.apache.com
[24] Branson, Rick, "Facebook's Instagram: Making the Switch to Cassandra from Redis, a 75% 'Insta' Savings,"

Haystack for Photos:

Facebook uses a multi-layered cache with a final Haystack backend to manage their binary photograph content.

Prior to their Haystack system, Facebook used a network file system (NFS) / network attached storage (NAS) system, though they quickly found this system to have substantial drawbacks.

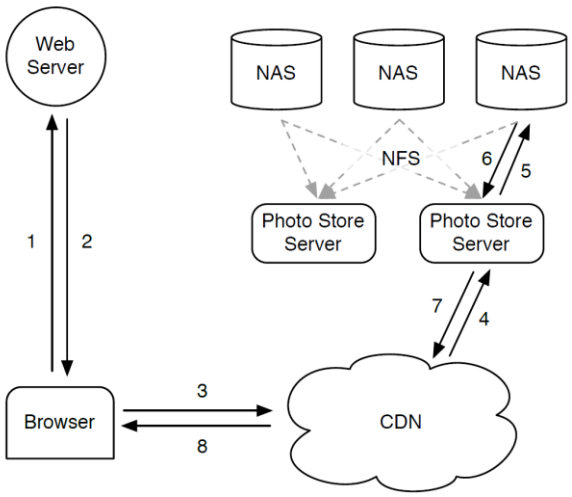

Figure 5—NFS / NAS System Used for Photo Store Prior to Haystack

Figure 5 shows a schematic of the prior system, from Beaver et al 2008[25], with a content delivery network, such as Akamai, connecting the user's browser and the photo servers. Further a network file system connected the NAS appliances to the photo servers.

The system was found not to be practical as scale increased. In particular, due to inefficiencies in how the NAS manages file metadata, often 10 disk operations were required to read a single image.[26] Figure 6 shows a comparison of IO count per photo for Facebook's solutions as they evolved. Haystack wins at 1 I/O per photo.

[25] Beaver et al 2008
[26] Ibid.

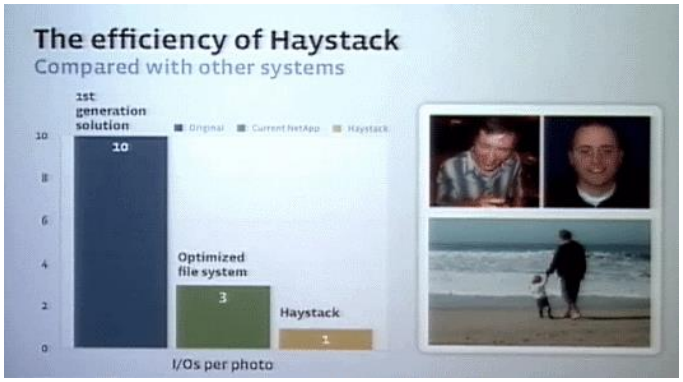

Figure 6—I/O Count of Facebooks' Photo Store Solutions

Facebook engineers determined that the need to regularly access a large number of old photos was an application poorly suited to MySQL, a NAS solution, or Hadoop, as these existing storage systems lacked the right RAM to disk storage ratio. They decided, therefore, to create a streamlined system so that each photo had small enough metadata overhead that keeping the metadata in memory was achievable.

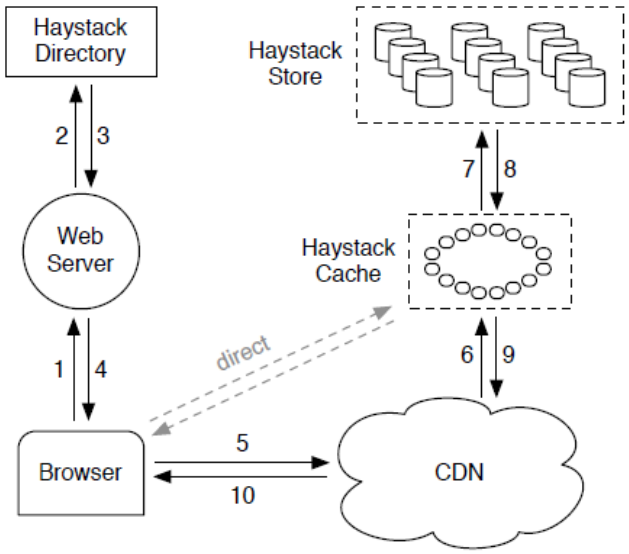

Figure 7—Photo Lookup After Haystack

Figure 7 shows the Photo Lookup Architecture after Haystack was implemented. Key changes are that now the Web Server uses the Haystack Directory to determine the best machine ID, cache path, and logical volume to find the photo in the Haystack data Store. The browser can also connect directly to the Haystack Cache, reducing the use on content delivery partners.

Figure 8 shows the basic schema for the database. The Magic Number fields allow parser to rebuild the data in the event of a key corruption. Apart from data needed to find the object stored, to rebuild the object, no additional overhead is stored.

Header Magic Number	Magic number used to find the next possible needle during recovery
Cookie	Security cookie supplied by the client application to prevent brute force attack
Key	64-bit object key
Alternate Key	32-bit object alternate key
Flags	Currently only one signifying that the object has been removed
Size	Data size
Footer Magic Number	Magic number used to find the possible needle end during recovery
Data Checksum	Checksum for the data portion of the needle
Padding	Total needle size is aligned to 8 bytes

Figure 8—Haystack schema.

In addition to developing the Haystack database technology, Facebook also put a substantial effort toward layers of caching, such that the most popular pictures are rapidly delivered, and only a small fraction of requests actually make their way to the Haystack server. Studies estimate that their caching scheme allows approximately 80% of photo requests to be found in cache, rather than retrieved from the server. [27] Figure 9 shows this high hit rate, showing that only the remaining 20% would need to get collected from the server.

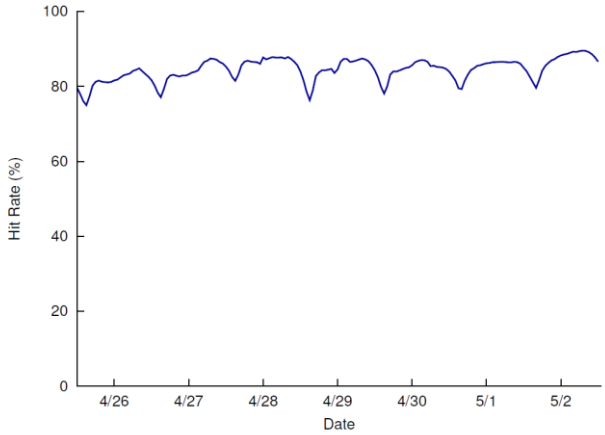

Figure 9—Example Haystack Cache Hit Rate.

The process begins at the user's browser, where an HTML file from Facebook's web-servers is received. A reference to a photo object in the HTML file will have various data coded into its URL, including: a unique photo identifier, image dimensions, and fetch path. The fetch path specifies the path to use in trying to find a matching photo.

First, local cache on the user's computer/device is consulted for an object matching the unique photo identifier. If that check fails, the next location specified on the fetch path will be that of an edge cache server. As of 2013, Facebook had nine edge cache server locations, and had plans to add more.[28] The fetch path will specify which specific edge cache server to use. Each edge cache server keeps a hash

[27] Ibid.
[28] Ibid.

table in memory to facilitate match checking, and stores of Flash memory for quick retrieval. The edge cache uses a FIFO eviction policy.

If the matching content is found at the edge cache, it is downloaded. Failing that, the process continues along the fetch path to the Origin cache. Like the Edge caches, the Origin cache also has a hash map stored in memory, a Flash store, and uses a FIFO eviction policy.

If no match occurs at the Origin cache, then the Haystack database is queried for the matching photo.

The fact that Facebook continues to use Haystack as the photo store in conjunction with its much newer Messaging HBase solution suggests that Facebook is quite satisfied with this photo store solution.

HBase and Messages:

About HBase:

HBase is a non-relational, distributed database modeled after BigTable by Google.[29] It was originally developed in 2007 by the company Powerset (now a subsidiary of Microsoft) for natural language processing, wherein large datasets would need to be traversed. HBase is now considered a top-level Apache project.

The tool boasts several key benefits, including: low-latency reads and writes, fault tolerance, and peta-byte scale capabilities. In fact, HBase does auto-sharding wherein table data is dynamically redistributed if the grow too large.[30]

A Region in HBase provides the horizontal scalability. Regions contain subsets of the table's data and are a chunk of rows stored together.

At the outset, there is only one region to a table. As rows are added and the table grows too large, the region is automatically split into two at approximately the middle. Figure 10 shows this Region splitting.[31]

[29] http://en.wikipedia.org/wiki/Apache_HBase
[30] Bertozzi, Matteo, "HBase - Who needs a Master?",
[31] Ibid.

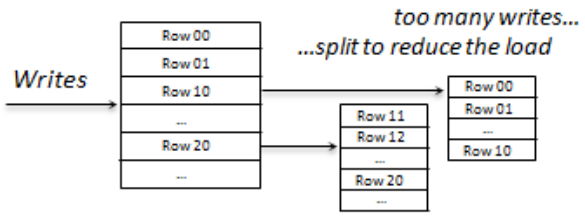

Figure 10- HBase Region Splitting

Regions servers contain regions, while an HMaster coordinates the entire cluster. The HMaster to reassign Regions to different Region Servers and manages Region Server failures.

Figure 11 shows how the metadata stored maps the table data to the various Regions across various Region Servers.[32]

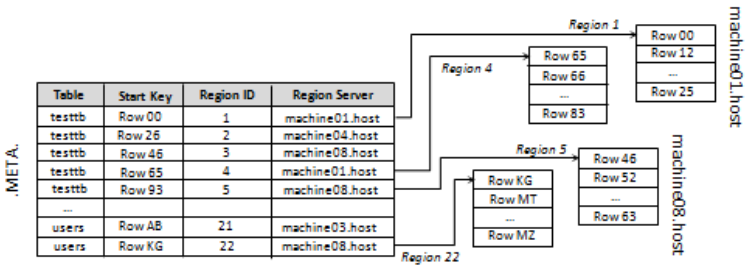

Figure 11 Table Mapping from META in HBase

The end result is a robust large-scale distributed database.

HBase At Facebook

Previous to implementing HBase to drive Message and Timeline content, non-chat messages were stored in the MySQL. Facebook stored chat messages in memory, and these were lost after a few days. Chat messages were approximately 20 times the non-chat messages and email would likely be larger still. It was clear to the Facebook engineers that a new robust solution would need to be implemented to add the message functions.

In their "Storage Infrastructure Behind Facebook Messages Using HBase at Scale" IEEE bulletin, several Facebook Engineers explain why they chose HBase[33]:

1. High Write Throughput: They expected the Message workload to cause a B+ tree to have repeated rewrites as it mutates. The Log Structured Merge Tree scheme in HBase was found to be much more efficient at handling this random write workload.
2. Low Latency Reads: The read performance during their initial testing was sufficient, and expected to further improve after adding cache layers in front of HBase store, and tuning.

[32] Ibid.
[33] Aiyer, Amitanand, et. al., "Storage Infrastructure Behind Facebook Messages Using HBase at Scale,"

3. Elasticity: With continual expected growth of the Message datasets, Facebook looked to leverage HBase's and HDFS's inherent clustering elasticity.
4. Cheap and Fault Tolerant: Enabling Facebook to target inexpensive and less reliable commodity hardware.
5. Consistency in a Data Center: HBase allows for consistency regardless of the Region replica used, which would be necessary to carry on Message functions
6. HDFS Experience: Facebook had already used HDFS for their backend solutions.

Additionally Facebook uses HBase for its ODS internal monitoring system. ODS continuously monitors software and hardware system performance.

Scribe:

Facebook naturally wants to track and aggregate the user actions for ad and revenue purposes. To capture this data, which comes from thousands of web server logs, perform this function, Facebook developed Scribe. Scribe runs on HDFS. Figure 12 shows a conceptual schematic of how Scribe funnels the browser to web server data and passes it on to Facebook's Production Hive.[34]

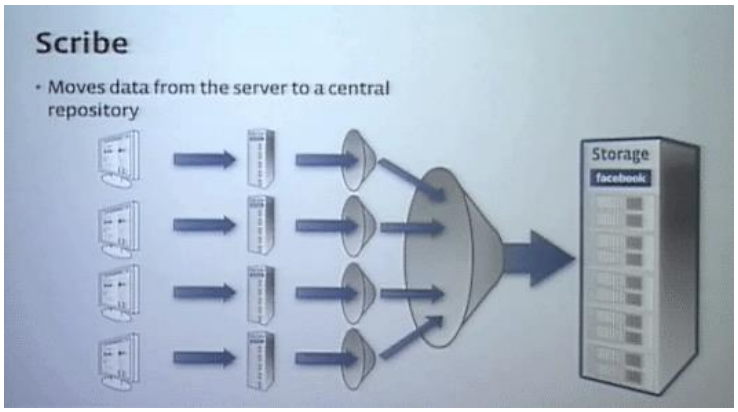

Figure 12—Facebook Scribe's Conceptual Workflow

The reason Facebook developed their own solution was to have a scalable log aggregation solution that could grow with its ever increasing data volume.[35]

In October of 2008, Facebook released Scribe to the open source community.

[34] http://www.publickey1.jp/blog/09/facebook8php.html
[35] Thusoo, Ashish, et. al. "Data Warehousing and Analytics Infrastructure at Facebook,"

Hive and Backend Analytics:

Facebook had numerous analytics that took hours to complete on their existing MySQL store. They needed to pull the data onto something built for the size, such as Hadoop. The problem they encountered was that many employees were used to the SQL type data structures and searches, and writing the map reduce functions required to work with Hadoop would be prohibitive.

In January of 2007 Facebook started work on Hive as a way to bridge the gap between the Hadoop size and the SQL query language. Facebook open sourced their result in August of 2008.

Hive users can use HiveQL, a SQL-like script language, to write their queries or they can use traditional map-reduce functions. Figure 13 shows an example select statement written in Hive QL, illustrating that the ANSI SQL syntax must be used:

```
SELECT t1.a1 as c1, t2.b1 as c2
FROM t1 JOIN t2 ON (t1.a2 = t2.b2);
```

Figure 13—HiveSQL Syntax.

Additionally, Hive supports datatypes such as tables, arrays, lists and structs natively to make database operations even simpler. Figure 14 shows these data types.

- Integers – bigint(8 bytes), int(4 bytes), smallint(2 bytes), tinyint(1 byte). All integer types are signed.
- Floating point numbers – float(single precision), double(double precision)
- String

Hive also natively supports the following complex types:

- Associative arrays – map<key-type, value-type>
- Lists – list<element-type>
- Structs – struct<file-name: field-type, ... >

Figure 14—Hive Datatypes

The overall architecture can be thought of as two parts: the Hive front end and the Hadoop backend. Figure 15 shows the conceptual plot of the database.[36] The command line interface and web interface

[36] http://www.cubrid.org/blog/dev-platform/platforms-for-big-data/

connect to the Driver interpreter directly. Other Facebook tools can use ODBC or JDBC connectivity, via Facebook's Thrift interconnectivity tool, to also connect to the Driver. The Driver executes initiates the HDFS workload and receives the results.

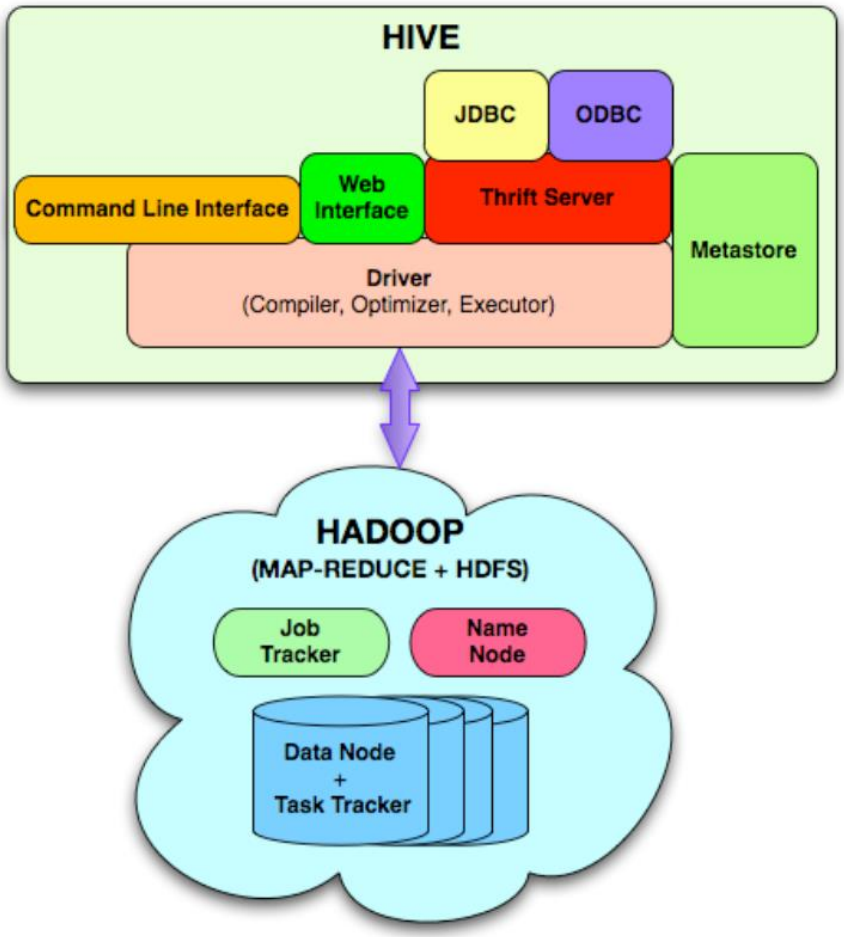

Figure 15—Overall Hive Architecture

Hive quickly became very popular at Facebook. Analysts then had a new way an effective way to sift through the underlying data and run the analytics that many of Facebook's products need. Some of these products include their Ad Product and their Lexicon product.

Conclusions:

This report shows an overall map of Facebook's Data Infrastructure. Further, a sense of the progression from the initial MySQL core to a vast array of solutions is shown.

A look at the underlying technology used by Facebook gives one an immense appreciation for the magnitude of challenge they face, and how well their solution performs in spite of it.

Bibliography

Agarwal, Aditya, “Facebook: Science and the Social Graph,” QCon SF 2008

Aiyer, Amitanand, et. al., “Storage Infrastructure Behind Facebook Messages Using HBase at Scale,” Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, June 2012 Vol. 35 No. 2

Barrigas, Hugo, et. al. “Overview of Facebook Scalable Architecture,” ISDOC '14 Proceedings of the International Conference on Information Systems and Design of Communication, pp. 173-176.

Beaver, Doug et al., “Finding a needle in Haystack: Facebook’s photo storage,” OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation, Article No. 1-8

Bertozzi, Matteo, “HBase - Who needs a Master?”, https://blogs.apache.org/hbase/entry/hbase_who_needs_a_master 25 Apr 2013

Borthakur , Dhruba, et. al. “Apache Hadoop Goes Realtime at Facebook,” SIGMOD ’11, June 12–16, 2011

Borthakur, Dhruba, “Petabyte Scale Data at Facebook,” XLDB Conference at Stanford University, Sep. 2012

Branson, Rick, “Facebook’s Instagram: Making the Switch to Cassandra from Redis, a 75% ‘Insta’ Savings,” http://planetcassandra.org/blog/interview/facebooks-instagram-making-the-switch-to-cassandra-from-redis-a-75-insta-savings/ 15 Oct 2014

Bronson, Nathan, et. al. “TAO: Facebook’s Distributed Data Store for the Social Graph,” 2013 USENIX Annual Technical Conference (USENIX ATC ’13)

Callaghan, Mark, “MySQL at Facebook,” 2010 O'Reilly MySQL User Conference & Expo

Campaniello, Paul, “Facebook IT Responds to Stonebraker,” https://www.scalebase.com/facebook-it-responds-to-stonebreaker/ 11 Jul 2011

Fisk, Harrison, “MySQL at Facebook, Current and Future,” Percona Live NYC 2011

Greene, Steaphan, “WebScaleSQL: A collaboration to build upon the MySQL upstream”, https://code.facebook.com/posts/1474977139392436/webscalesql-a-collaboration-to-build-upon-the-mysql-upstream 27 Mar 2014

Harris, Derrick, “Facebook tells how it makes its MySQL cluster ‘almost run itself’”, https://gigaom.com/2013/10/23/facebook-offers-a-peek-at-how-it-keeps-mysql-online/ 23 Oct 2013

Harris, Derrick. “Facebook trapped in MySQL ‘fate worse than death’”, 7 Jul 2011, https://gigaom.com/2011/07/07/facebook-trapped-in-mysql-fate-worse-than-death/

Harter, Tyler, et. al., “Analysis of HDFS Under HBase: A Facebook Messages Case Study”, Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST ’14), 2014

https://code.facebook.com/posts/1474977139392436/webscalesql-a-collaboration-to-build-upon-the-mysql-upstream

https://www.facebook.com/MySQLatFacebook

https://github.com/facebook/mysql-5.6

https://github.com/facebookarchive/mysql-5.1

http://hbase.apache.org/

http://hive.apache.org/

http://newsroom.fb.com/company-info/

Kumar, Sanjeev, "Analyzing the Facebook Workload," Workload Characterization (IISWC), 2012 IEEE International Symposium on, Nov. 2012

Lakshman, Avinash and Prashant Malik, "Cassandra - A Decentralized Structured Storage System", ACM SIGOPS Operating Systems Review, Volume 44 Issue 2, April 2010, pp. 35-40.

Logan, Becca, "Keeping Up," https://www.facebook.com/notes/facebook/keeping-up/7899307130 21 Dec 2007.

Mituzas, Domas, "High Concurrency MySQL," 2010 O'Reilly MySQL User Conference & Expo

Nishtala, Rajesh et. al, "Scaling Memcache at Facebook", 10th USENIX Symposium on Networked Systems Design and Implementation, 2013.

Nishtala, Rajesh and Venkat Venkataramani, "Scaling memcache at Facebook," https://www.facebook.com/notes/facebook-engineering/scaling-memcache-at-facebook/10151411410803920

Priymak, Shlomo, "Under the hood: MySQL Pool Scanner (MPS)", https://www.facebook.com/notes/facebook-engineering/under-the-hood-mysql-pool-scanner-mps/10151750529723920, 22 Oct 2013

Thusoo, Ashish, et. al. "Data Warehousing and Analytics Infrastructure at Facebook," SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 1013-1020.

Thusoo, Ashish et. al., "Hive – A Petabyte Scale Data Warehouse Using Hadoop," International Conference on Data Engineering - ICDE, pp. 996-1005, 2010.

Vinod Venkataraman, et. al., "Facebook ODS Real-time monitoring system built on HBase," Strata Conference + Hadoop World, Oct 2012