Facebook's New Realtime Analytics System: HBase to Process 20 Billion Events Per Day

Tuesday, March 22, 2011 at 9:55AM

Todd Hoff in Example, analytics, facebook

Facebook did it again. They've built another system capable of doing something useful with ginormous streams of realtime data. Last time we saw Facebook release their New Real-Time Messaging System: HBase To Store 135+ Billion Messages A Month. This time it's a realtime analytics system handling *over 20 billion events per day (200,000 events per second) with a lag of less than 30 seconds*.

Alex Himel, Engineering Manager at Facebook, explains what they've built (video) and the scale required:

> *Social plugins have become an important and growing source of traffic for millions of websites over the past year. We released a new version of Insights for Websites last week to give site owners better analytics on how people interact with their content and to help them optimize their websites in real time. To accomplish this, we had to engineer a system that could process over 20 billion events per day (200,000 events per second) with a lag of less than 30 seconds.*

Alex does an excellent job with the presentation. Highly recommended. But let's take a little deeper look at what's going on...

The need for such a high powered analytics system is driven by Facebook's brilliant plan for world wide web domination via the viral propagation of social plugins, all tying the non-Facebook web back into Facebook and the Facebook web back into the non-Facebook web. Basically anything that people can do is captured and fed back through Facebook and anything done on Facebook can be displayed on your website, building closer relations between the two.

Facebook's Social Plugins are Roman Empire Management 101. You don't have to conquer everyone to build an empire. You just have control everyone with the threat they could be conquered while making them realize, oh by the way, there's lots of money to be made being friendly with Rome. This strategy worked for quite a while as I recall.

You've no doubt seen Social Plugins on websites out the wild. A social plugin lets you see what your friends have liked, commented on or shared on sites across the web. The idea is putting social plugins on a site makes the content more engaging. Your friends can see what you are liking and in turn websites can see what everyone is liking. Content that is engaging gives you more clicks, more likes, and more comments. For a business or brand, or even an individual, the more engaging the content is, the more people see it, the more it pops up in news feeds, the more it drives traffic to a site.

The formerly lone-wolf web, where content hunters stalked web sites silently and singly, has been turned into a charming little village, where everyone knows your name. That's the power of social.

Posts here on HighScalability, for example, now have Like buttons. TechCrunch has famously moved to using Facebook's commenting system. Immediately the debate centered on the quality of the comment system itself, but that was hardly the point, the point was to plunge

TechCrunch more deeply into Facebook's ecosystem of 500+ million users. Other plugins include: Recommendations, Activity Feed, Login, Registration, Facepile, and Live Stream.

All that data doesn't mean much unless you can make sense of it and also prove to content providers that social plugins actually do make their sites more engaging. That's where Facebook's Insights System comes in. It's an analytics system giving you access to all that juicy data being collected. It offers stats like Like button analytics, Comments box analytics, Popular pages, Demographics, and Organic sharing.

Imagine millions of websites and billions of pages and millions of people continually streaming data in via these social plugins. How do you make sense of all that data in real-time? It's a challenging problem.

Value Proposition

With an Insights System content producers can see what people like, which will enable content producers to generate more of what people like, which raises the content quality of the web, which gives users a better Facebook experience.

System Goals

- Give people realtime counters, in a very reliable way, across a bunch of different metrics, and account for data skew.
- Provide anonymous data. You can not figure out who the people are.
- Show why plugins are valuable. What value is your business deriving from it?
- Make the data more actionable. Help users take action to make their content more valuable.

- New UI metaphors. Use the idea of a funnel.
- How many people see a plugin, how many people take action on it, and how many are converted to traffic back on your site.

- Make the data more timely.
  - They went realtime. Went from a 48-hour turn around to 30 seconds.
  - Multiple points of failure were removed to make this goal.

Challenges

- **Lots of Event Types**
  - Tracking 100+ metrics.
  - Plugin impressions.
  - Likes
  - News Feed Impressions
  - News Feed Clicks
  - Demographics
- **Massive Amounts of Data**
  - 20 billion events per day (200,000 events per second)
- **Data Skew - Uneven Distribution of Keys**
  - Likes follow something like a power law distribution. The long tail gets very few likes, but some resources get huge numbers of likes.
  - This brings up issues of hot regions, hot keys, and lock contention.

Implemented a Bunch of Different Prototypes

- **MySQL DB Counters**
  - Have a row with a key and a counter.

  - Results in lots of database activity.
  - Stats are kept at a day bucket granularity. Every day at midnight the stats would roll over.
    - When the roll over period is reached this resulted in a lot of writes to the database, which caused a lot of lock contention.
    - Tried to spread the work by taking into account time zones.
    - Tried to shard things differently.
  - The high write rate led to lock contention, it was easy to overload the databases, had to constantly monitor the databases, and had to rethink their sharding strategy.
  - Solution not well tailored to the problem.
- **In-Memory Counters**
  - If you are worried about bottlenecks in IO then throw it all in-memory.
  - No scale issues. Counters are stored in memory so writes are fast and the counters are easy to shard.
  - Felt in-memory counters, for reasons not explained, weren't as accurate as other approaches. Even a 1% failure rate would be unacceptable. Analytics drive money so the counters have to be highly accurate.
  - They didn't implement this system. It was a thought experiment and the accuracy issue caused them to move on.
- **MapReduce**
  - Used Hadoop/Hive for previous solution.
  - Flexible. Easy to get running. Can handle IO, both massive writes and reads. Don't have to know how they will query ahead of time. The data can be stored and then queried.
  - Not realtime. Many dependencies. Lots of points of failure. Complicated system. Not dependable enough to hit realtime goals.

- **Cassandra**
    - HBase seemed a better solution based on availability and the write rate.
    - Write rate was the huge bottleneck being solved.

The Winner: HBase + Scribe + Ptail + Puma

- At a high level:
    - HBase stores data across distributed machines.
    - Use a tailing architecture, new events are stored in log files, and the logs are tailed.
    - A system rolls the events up and writes them into storage.
    - A UI pulls the data out and displays it to users.
- Data Flow
    - User clicks Like on a web page.
    - Fires AJAX request to Facebook.
    - Request is written to a log file using Scribe.
        - Scribe handles issues like file roll over.
        - Scribe is built on the same HTFS file store Hadoop is built on.
        - Write extremely lean log lines. The more compact the log lines the more can be stored in memory.
    - Ptail
        - Data is read from the log files using Ptail. Ptail is an internal tool built to aggregate data from multiple Scribe stores. It tails the log files and pulls data out.
        - Ptail data is separated out into three streams so they can eventually be sent to their own clusters in different datacenters.
            - Plugin impression
            - News feed impressions

            - Actions (plugin + news feed)
    - Puma
        - Batch data to lessen the impact of hot keys. Even though HBase can handle a lot of writes per second they still want to batch data. A hot article will generate a lot of impressions and news feed impressions which will cause huge data skews which will cause IO issues. The more batching the better.
        - Batch for 1.5 seconds on average. Would like to batch longer but they have so many URLs that they run out of memory when creating a hashtable.
        - Wait for last flush to complete for starting new batch to avoid lock contention issues.
    - UI Renders Data
        - Frontends are all written in PHP.
        - The backend is written in Java and Thrift is used as the messaging format so PHP programs can query Java services.
    - Caching solutions are used to make the web pages display more quickly.
        - Performance varies by the statistic. A counter can come back quickly. Find the top URL in a domain can take longer. Range from .5 to a few seconds.
        - The more and longer data is cached the less realtime it is.
        - Set different caching TTLs in memcache.
    - MapReduce
        - The data is then sent to MapReduce servers so it can be queried via Hive.
        - This also serves as a backup plan as the data can be recovered from Hive.
        - Raw logs are removed after a period of time.
- HBase is a distribute column store.

- Database interface to Hadoop. Facebook has people working internally on HBase.
- Unlike a relational database you don't create mappings between tables.
- You don't create indexes. The only index you have a primary row key.
- From the row key you can have millions of sparse columns of storage. It's very flexible. You don't have to specify the schema. You define column families to which you can add keys at anytime.
- Key feature to scalability and reliability is the WAL, write ahead log, which is a log of the operations that are supposed to occur.
    - Based on the key, data is sharded to a region server.
    - Written to WAL first.
    - Data is put into memory. At some point in time or if enough data has been accumulated the data is flushed to disk.
    - If the machine goes down you can recreate the data from the WAL. So there's no permanent data loss.
    - Use a combination of the log and in-memory storage they can handle an extremely high rate of IO reliably.
- HBase handles failure detection and automatically routes across failures.
- Currently HBase resharding is done manually.
    - Automatic hot spot detection and resharding is on the roadmap for HBase, but it's not there yet.
    - Every Tuesday someone looks at the keys and decides what changes to make in the sharding plan.

- **Schema**
    - Store on a per URL basis a bunch of counters.
    - A row key, which is the only lookup key, is the MD5 hash of

the reverse domain

    - Selecting the proper key structure helps with scanning and sharding.
    - A problem they have is sharding data properly onto different machines. Using a MD5 hash makes it easier to say this range goes here and that range goes there.
    - For URLs they do something similar, plus they add an ID on top of that. Every URL in Facebook is represented by a unique ID, which is used to help with sharding.
    - A reverse domain, *com.facebook/* for example, is used so that the data is clustered together. HBase is really good at scanning clustered data, so if they store the data so it's clustered together they can efficiently calculate stats across domains.
  - Think of every row a URL and every cell as a counter, you are able to set different TTLs (time to live) for each cell. So if keeping an hourly count there's no reason to keep that around for every URL forever, so they set a TTL of two weeks. Typically set TTLs on a per column family basis.
- Per server they can handle 10,000 writes per second.
- Checkpointing is used to prevent data loss when reading data from log files.
  - Tailers save log stream check points in HBase.
  - Replayed on startup so won't lose data.
- Useful for detecting click fraud, but it doesn't have fraud detection built in.
- **Tailer Hot Spots**
  - In a distributed system there's a chance one part of the system can be hotter than another.
  - One example are region servers that can be hot because more keys are being directed that way.
  - One tailer can be lag behind another too.

    - If one tailer is an hour behind and the others are up to date, what numbers do you display in the UI?
        - For example, impressions have a way higher volume than actions, so CTR rates were way higher in the last hour.
        - Solution is to figure out the least up to date tailer and use that when querying metrics.
- **Future Directions**
    - **Top Lists**
        - Very hard to find the top URLs, the URLs with the most likes, for domains like YouTube which have millions of URLs shared very quickly.
        - Need more creative solutions to keep an in-memory sorts and keep it up to date as data changes.
    - **Distinct User Counts**
        - How many people across a time window liked a URL. Easy to do in MapReduce, hard to do with a naive counter solution.
    - **Generalize for Applications other than Social Plugins**
    - **Move to Multiple Datacenters**
        - Single datacenter currently, but hope to move to multiple datacenters.
        - Current fallback plan is to use the MapReduce system.
        - The backup systems are tested each night. Queries against Hive and this new system are compared to see that they match.
- **Project**
    - Took about 5 months.
    - Two engineers first started working on the project. Then a 50% engineer was added.
    - Two UI people worked on the front-end.
    - Looks like about 14 people worked on the product in from

engineering, design, PM, and operations.

When we look at the messaging system and this analytics system, we notice what the two systems have in common: large numbers, HBase, real-time. The challenge of dealing with huge write loads in a reliable and timely fashion is a common substrate to these problems. Facebook is focussing on the HBase, Hadoop, HDFS ecosystem and counting on the operational quirks to be ironed out later. Others choose Cassandra because they love it's scalability, multi-datacenter functionality, and ease of operational use, but it doesn't fit as cleanly into the overall analytics stack.

What does this mean for you? Even if you aren't Facebook this architecture is simple enough and composed of enough off the shelf tools that it could work for much small projects too.

Related Articles

- Medialets Architecture - Defeating The Daunting Mobile Device Data Deluge - another take on the analytics platform.
- Twitter’s Plan To Analyze 100 Billion Tweets
- Scaling Analytics Solutions Tech Talk (3/2/11) [HQ]
