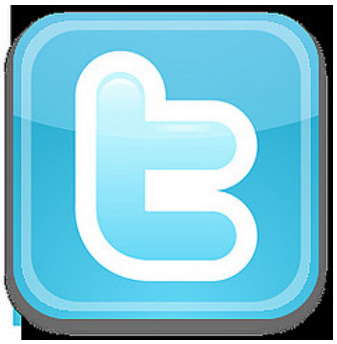
How Twitter Stores 250 Million Tweets a Day Using MySQL

Monday, December 19, 2011 at 9:05AM

Todd Hoff in Strategy, Twitter



Jeremy Cole, a DBA Team Lead/Database Architect at Twitter, gave a really good talk at the O'Reilly MySQL conference: Big and Small Data at @Twitter, where the topic was thinking of Twitter from the data perspective.

One of the interesting stories he told was of the transition from Twitter's old way of storing tweets using temporal sharding, to a more distributed approach using a new tweet store called T-bird, which is built on top of Gizzard, which is built using MySQL.

Twitter's original tweet store:

- Temporally sharded tweets was a good-idea-at-the-time architecture. Temporal sharding simply means tweets from the same date range are stored together on the same shard.
- The problem is tweets filled up one machine, then a second, and then a third. You end up filling up one machine after another.
- This is a pretty common approach and one that has some real flaws:
  - **Load balancing**. Most of the old machines didn't get any traffic because people are interested in what is happening now, especially with Twitter.
  - **Expensive.** They filled up one machine, with all its replication slaves, every three weeks, which is an expensive setup.

    - **Logistically complicated**. Building a whole new cluster every three weeks is a pain for the DBA team.

Twitter's new tweet store:

- When you tweet it's stored in an internal system called T-bird, which is built on top of Gizzard. Secondary indexes are stored in a separate system called T-flock, which is also Gizzard based.
- Unique IDs for each tweet are generated by Snowflake, which can be more evenly sharded across a cluster. FlockDB is used for ID to ID mapping, storing the relationships between IDs (uses Gizzard).
- Gizzard is Twitter's distributed data storage framework built on top of MySQL (InnoDB).
    - InnoDB was chosen because it doesn't corrupt data. Gizzard us just a datastore. Data is fed in and you get it back out again.
    - To get higher performance on individual nodes a lot of features like binary logs and replication are turned off. Gizzard handles sharding, replicating N copes of the data, and job scheduling.
    - Gizzard is used as a building block for other storage systems at Twitter.
- Gizzard implements a tweet store that isn't perfect in the sense of load balancing, but it allows:
    - **Growing slowly** over time without having to worry when machines are going to fill up or when they have to make a hard cut over at an exact time.
    - **DBAs can get some sleep**. They don't have to make those decisions as frequently and they don't have to scale as inefficiently cost wise.
- MySQL works well enough most of the time that it's worth using. Twitter values stability over features so they've stayed with older releases.

- MySQL doesn't work for ID generation and graph storage.
- MySQL is used for smaller datasets of < 1.5TB, which is the size of their RAID array, and as a backing store for larger datasets.
- Typical database server config: HP DL380, 72GB RAM, 24 disk RAID10. Good balance of memory and disk.

MySQL isn't used for everything:

- **Cassandra** is used for high velocity writes, and lower velocity reads. The advantage is Cassandra can run on cheaper hardware than MySQL, it can expand easier, and they like schemaless design.
- **Hadoop** is used to process unstructured and large datasets, hundreds of billions of rows.
- **Vertica** is being used for analytics and large aggregations and joins so they don't have to write MapReduce jobs.

Some other ideas:

- **Loose coupling**. Whenever something breaks they realize it wasn't loosely coupled enough and something pushed back way to hard on something else.
- **Soft launches**. Decouple pushing a release out and driving new traffic to something. Features can be turned on and off.
- **Open source**. Twitter is noted for open sourcing much of their infrastructure. Jeremy talks about open sourcing in a way that I've never heard before. His idea is as a developer, open sourcing code, means you can think about your software knowing it won't go away. It's not just another throw away piece of software inside a corporate blackhole. That gives you permission to think of corporate software development in a different, more lasting way, more meaningful way.

Great job Jeremy!

Related Articles

- An Unorthodox Approach To Database Design : The Coming Of The Shard

---