"wildlife photography flock of flamingo" by Matthew Cabret on Unsplash

System design for Twitter

Narendra L

Sep 8, 2018 · 7 min read

I this article I am going to talk about the core logic of Timeline computation, Calculating trends using apache storm and message flow through the system.

Traffic: Twitter now has 300M worldwide active users. Every second on an average, around 6,000 tweets are tweeted on Twitter. Also, every second 6,00,000 Queries made to get the timelines!!

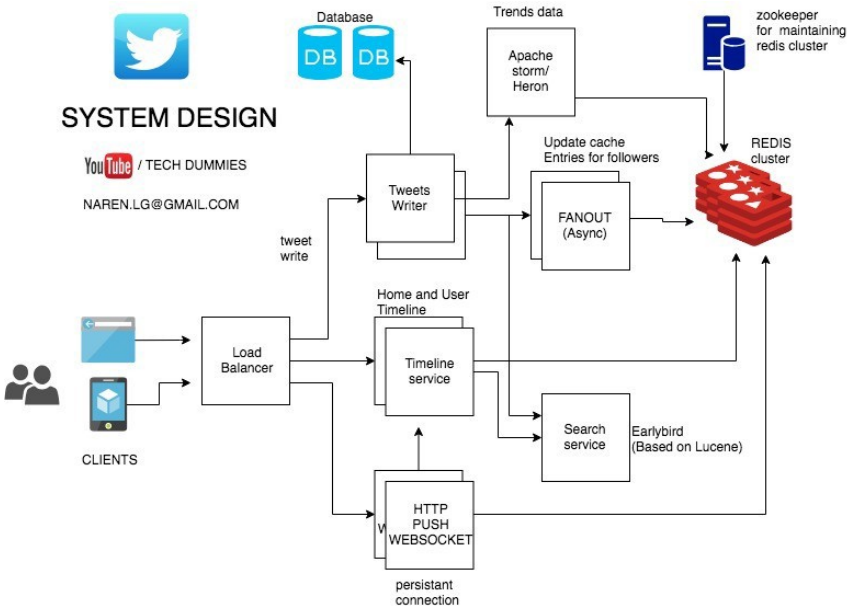
I have made a video on the same topic if you are too lazy to read do watch my video

User Features?

- The user should be able to tweet as fast as possible
- The user should be able to see Tweet Timeline(s)
- User timeline: Displaying user's tweets and tweets user retweet
- Home timeline: Displaying Tweets from people user follow
- Search timeline: Display search results based on #tags or search keyword
- The user should be able to follow another user
- Users should be able to tweet millions of followers within a few seconds (5 seconds)
- The user should see trends

Considerations before designing twitter

- If you see all of the features, it looks like read heavy, compared to write
- Its ok to have Eventual consistency, It's not much of a pain if the user sees the tweet of his follower a bit delayed
- Space is not a problem as tweets are limited to 140 characters



So lets design how to provide different timelines

As we know the system is read heavy let's use REDIS to access most of the information faster and also to store data but don't forget to store a copy of tweet and other users related info in Database.

Basic Architecture of Twitter service consists of a User Table, Tweet Table, and Followers Table

- User Information is stored in User Table
- When a user tweets it gets stored in the Tweet Table along with User ID.
- User Table will have 1 to many relationships with Tweet Table
- When a user follows another user, it gets stored in Followers Table, and also cache it Redis

So now how to build USER TIMELINE? which is shown below

- Fetch all the tweets from Global Tweet Table/Redis for a particular user
- Which also includes retweets, save retweets as tweets with original tweet reference
- Display it on user timeline, order by date time

As optimization save user timeline in the cache, eg: celebrities timelines are accessed million times so it's not much use to get it from DB always.

Home timeline

Strategy 1:

- Fetch all the users whom this user is following (from followers table)
- Fetch tweets from global tweet table for all
- Display it on home timeline
- Drawback: This huge search operation on relational DB is NOT Scalable. Though we can use sharding etc, this huge search operation will take time once the tweet table grows to millions.

Strategy 2: Solution is a write based fanout approach. Do a lot of processing when tweets arrive to figure out where tweets should go. This makes read time access fast and easy. Don't do any computation on reads. With all the work being performed on the write path ingest rates are slower than the read path. So precompute timelines for every active user and store it in the cache, so when the user accesses the home timeline, all you need to do is just get the timeline and show it

You can store this data in DB but what's the point?

A simple flow of tweet looks like,

- User A Tweeted
- Through Load Balancer tweet will flow into back-end servers
- Server node will save tweet in DB/cache
- Server node will fetch all the users that follow User A from the cache
- Server node will inject this tweet into in-memory timelines of his followers
- Eventually, all followers of User A will see the tweet of User A in their timeline

BUT does it work always? is it efficient?

If you see Celebrities they have Millions of Followers

Twitter system and it can take up to 4 minutes for a tweet to flow from TaylorSwift to her 83 million followers.

So If a person with millions of the followers on Twitter tweets, We need to update millions of lists which is not very scalable. So what we can do is,

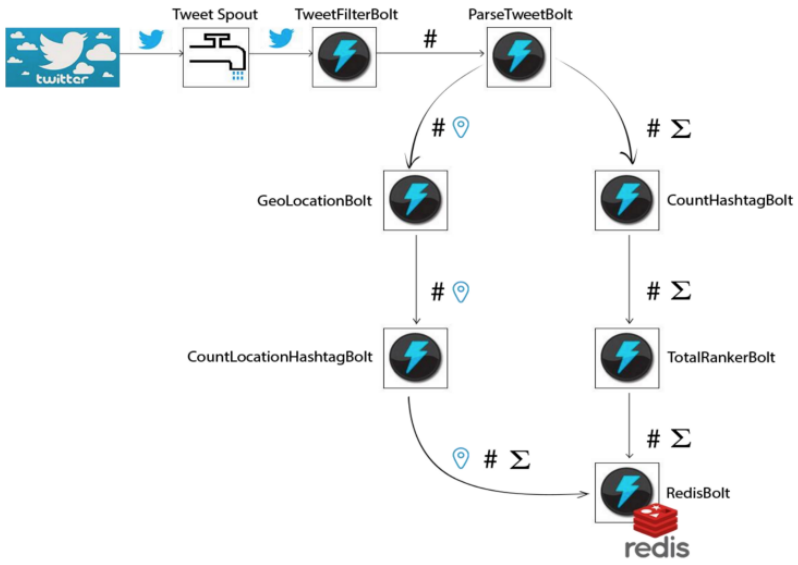
- Precomputed home timeline of User A with everyone except celebrity tweet(s)
- When User A access his timeline his tweet feed is merged with celebrity tweet at load time.

- For every user have the mapping of celebrities list and mix their tweets at runtime when the request arrives and cache it.

What are the other optimizations we can make?

- Don't compute timeline for inactive users who don't log in to the system for more than 15 days!!!

So how Trends / Trending topics are calculated?



Twitter uses Apache Storm and Heron framework to compute trending topics

These tasks run on many containers, These applications create a real-time analysis of all tweets send on the Twitter social network which can be used to determine the so-called trending topics.

Basically, method implies the counting of the most mentioned terms in the poster tweets in the Twitter social network.

The method is known in the domain of data analysis for the social network as "Trending Hashtags" method. Suppose two subjects A and B, the fact that A is more popular than B is equivalent to the fact that the number of mentions of the subject A is greater than the number of mentions of the subject B

Information needed

1. The number of mentions of a subject (hashtags)

2. Total time taken to generate that volume of tweets

TweetSpout: Represents a component used for issuing the tweets in the topology

TweetFilterBolt: Reads the tweets issued by the TweetSpout and executes the filtering. Only tweets that contain coded messages using the standard Unicode. also, violation and CC checks are made.

ParseTweetBolt: Processes the filtered tweets issued as tuples by the component TweetFilterBolt. Taking into consideration that the tuple is filtered, at this level we have the guarantee that each tweet contains at least one hashtag

CountHashtagBolt: Takes the tweets that are parsed through the component ParseTweetBolt and counts each hashtag. this is to get the hashtag and number of references to it

TotalRankerBolt: Makes a total ranking of all the counted hashtags. converts count to ranks in one more pipeline.

GeoLocationBolt: It takes the hashtag issued by the ParseTweetBolt, with the location of the tweet.

CountLocationHashtagBolt: Presents a functionality similar to the component CountHashtagBolt uses one more dimension ie. Location

RedisBolt: inserts in to redis

Searching:

Early Bird uses inverted full-text index. This means that it takes all the documents, splits them into words, and then builds an index for each word. Since the index is an exact string-match, unordered, it can be extremely fast. Hypothetically, an SQL unordered index on a varchar field could be just as fast, and in fact I think you'll find the big databases can do a simple string-equality query very quickly in that case.

Lucene does not have to optimize for transaction processing. When you add a document, it need not ensure that queries see it instantly. And it need not optimize for updates to existing documents.

However, at the end of the day, if you really want to know, you need to read the source. Both things you reference are open source, after all.

It has to scatter-gather across the datacenter. It queries every Early Bird shard and asks do you have content that matches this query? If you ask for "New York Times" all shards are queried, the results are returned, sorted, merged, and reranked. Rerank is by social proof, which means looking at the number of retweets, favorites, and replies.

Databases:

- 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 copies of the data, and job scheduling.
- 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.

And I believe this article helps you to learn about how Twitter works, if not completely at least a bit.