Social Network System-Design

Aysun Far

September 28, 2020

1 Objective:

I am building a system for a social network company, which has over millions of active users who record their relationships, share their interests through posting, liking, commenting and sharing each other's posts from their timeline.

1.1 Traffic:

We assume the social network has 100 million worldwide active users.

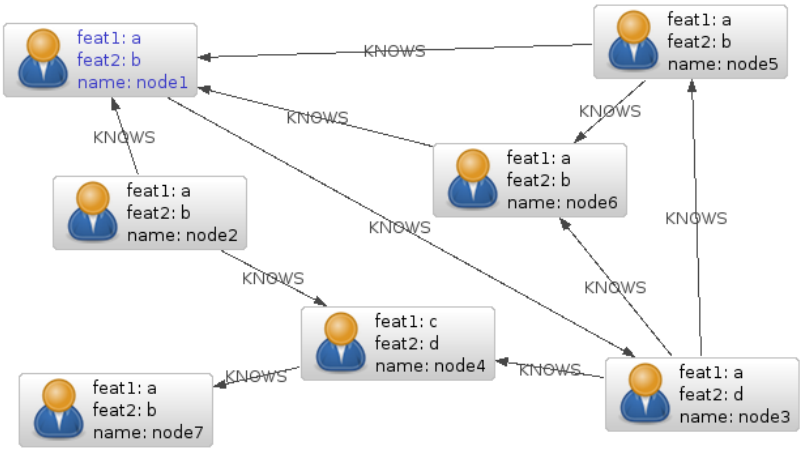
I will mainly discuss two aspects of our backend system:

- How the social graph is modeled and stored.
- How we can scaled the infrastructure, including servers, cache and databases, to serve 1 billion reads and millions of writes per second.

2 The Data Model For Social Graph

We can store majority of users' data, such as profiles, friends, posts and comments, inside a single giant social graph. There are two elements inside a social graph, nodes and edges.

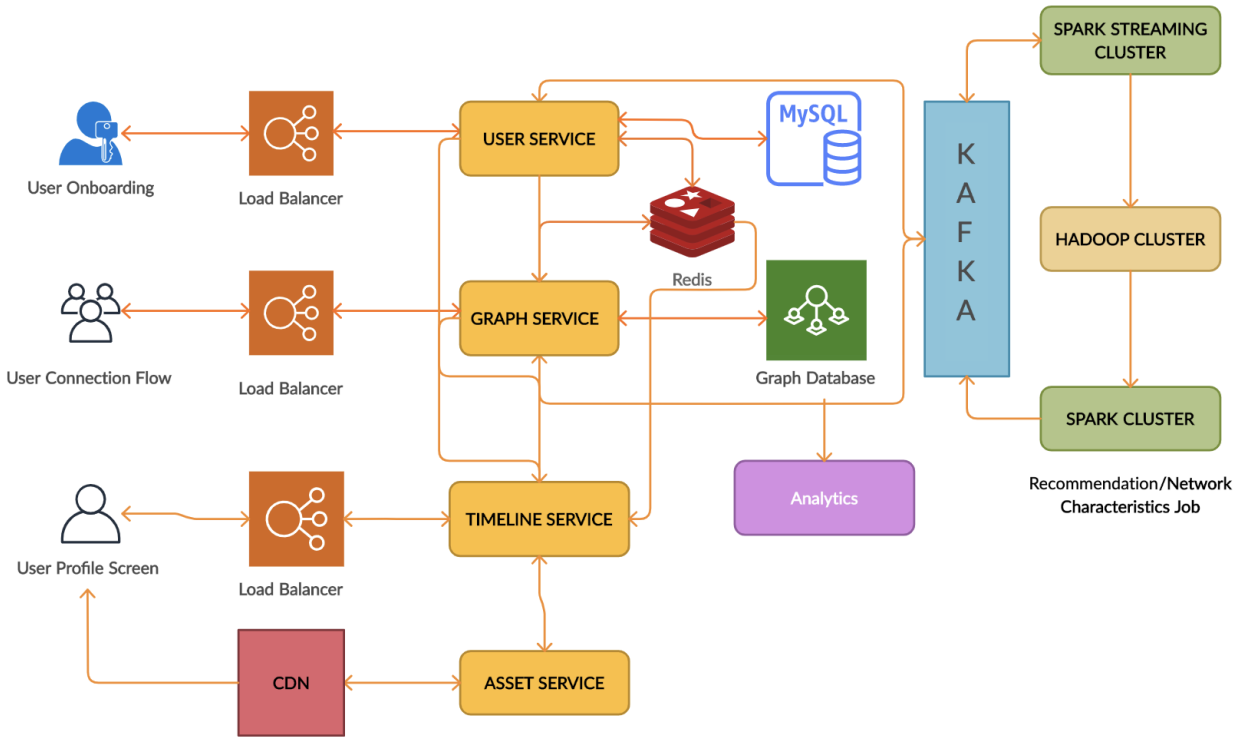
- Node: A graphical representation of an individual.
- Edge: represents a relationship between two individuals
- Degree: The number of edges connected to a nodes. These can be further broken down into in-degree (i.e. the number of incoming edges) or out-degree (the number of outgoing edges).



We have an option of either using graph-based database or table-based database. I went with the graph database (ex: neo4j) because unlike other database management systems (DBMS), relationships take first priority in graph databases. In the graph database, connected data is equally (or more) important than individual data points. This connection-first approach to data means relationships and connections are persisted and not just temporarily calculated through every part of the data lifecycle: from idea, to design in a logical model, to implementation in a physical model, to operation using a query language and to persistence within a scalable, reliable database system.

Unlike other database systems, this approach means your application doesn't have to infer data connections using things like foreign keys or out-of-band processing, like MapReduce. Your data models are simpler and yet more expressive than the ones you'd produce with relational databases or NoSQL stores.

In finding the distance between users (first question) and advance analytics (second question), the information about the connections are important. Also, many of the graph algorithms such as minimum shortest path, max-flow min cut, etc. are already implemented in graph database. Therefore, we can easily and efficiently implement the task for first and second questions.



User onboarding, login, and **user profile** related operations are handled by a **user service**. The user service sits on a MySQL database, Mysql because it is a very relational data from a structure point of view. Also, user information will be more read-heavy than write, and MySQL is enough for such a query pattern. The user service is also connected to a Redis for caching all the user information. When user service receives a request, it first looks up in Redis. If Redis has the information it is returned to the user, otherwise, the user service checks in the MySQL DB inserts the information into Redis for future use, and then returns back to the user. Also whenever a new user is added or information is updated, an event is inserted into Kafka so that the other services in the system are aware of the change.
When a person sees some other user's timeline, a request is sent to the **timeline service**, which further queries user service to fetch user's profile. When a person tries to see their own timeline, the request is sent to the timeline service, which fetches all the information of all the friends from Redis.

2.1 Efficient method for calculating the separation degree

This is where the **graph service** comes in. It keeps track of relationships between users, their weights, etc. Graph service again sits on top of a **Graph database**. Now the graph service again maintains cache in a Redis so that every time we need to fetch a user's connection we don't have to call the database again and again.

We can use the recent algorithms developed for solving Fully Dynamic All Pairs All Shortest Paths (FD-APAS) problems to compute the degree of separation between two users and common users

between them. A fully dynamic (APAS) oracle is a distance reporting data structure that supports dynamic insert edge and delete edge operations in a graph. For example, here I summerize the algorithm suggested in Abraham, Chechik, Krinninger(2017). They suggest a data structure which supports any sequence of operations in (insertion or deletion edges) of $O(n^2 log(n))$ amortized time per update and one look-up in the worst case per distance query. Here n is the number of nodes in the graph.

The algorithm has there phases:

- **Pre-processing:** The algorithm randomly selects a sequence of subsets of the nodes such as $S_1, S_2, \ldots, S_{\log n}$ where $|S_i| = O(nlog(n)/2^i)$. For each node $c \in S_i$, we compute the BFS(c) tree rooted at c and depth $(i+1)$. This step takes O(nm) where m is the number of edges in the graph.
- **Update:** As an insertion or deletion occurs, we can modify the BFS trees. This operation takes at most $O(n^2 log(n))$ amortized time in total.
- **Query response:** It can be shown that every shortest path of length i belongs to one of this BFS(c) for some $c \in S_i$ with high probability. For every given s, t, we can use a hashmap to find the BFS(c) such that both s, and t belongs to them. Then, $dist(s,t) = dist(s,c) + dist(c,t)$. This operation takes O(1). We can easily get the shortest path in $O(1)$ as well.

This data structure can respond so fast each query in O(1). However, the update phase takes quadratic time. Therefore, there is some limitation if the difference between updating time T0 and T1 decreases to the order of minutes for a large size social network. We can take advantage of parallel computation to speed up the whole process.

3 Advanced Analytics

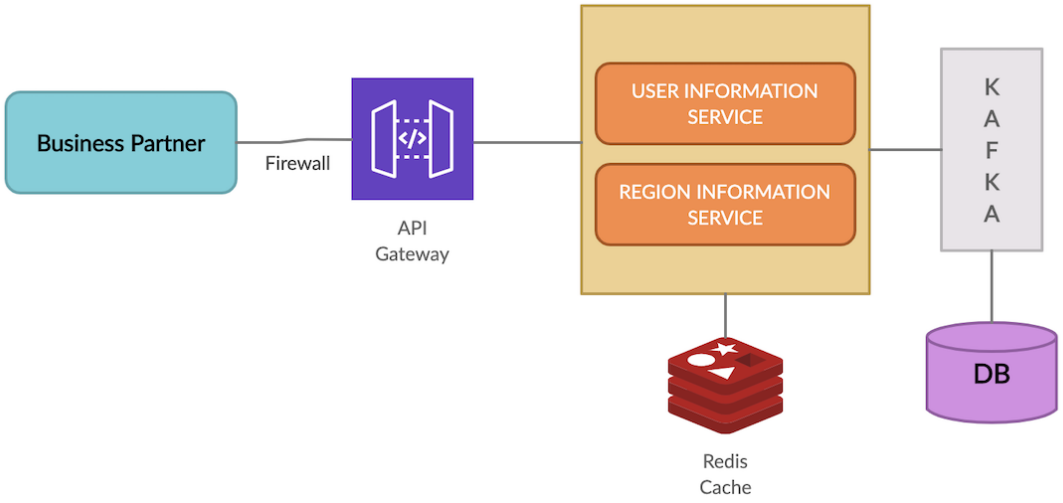
The **analytics** is responsible for running spark stream to answer advance analytics questions and return the responses through an API. In addition, the results can be visualized and the the metric trends are displayed in a dashboard. Here, we can take advantage of the **Spark GraphX**, which is the Spark API for graphs and graph-parallel computation. We will have a spark streaming running on Kafka which will store the informations such as connections and visited profiles of the users into Hadoop cluster. The GraphX will store the user's information as node. Furthermore, the properties of a directional edge between nodes(users) A to B will include whether A is connected to B and how often A visits B profile. Graphx has many built-in algorithms including the **PageRank** and **collaborate filtering** which can provide a better insight about the structure of the graph and recommendation.

Here is a list of potential analytics the system will handle:

- **PageRank**: measures the transitive influence or connectivity of nodes. We can use this algorithm to find important accounts based not only on whether they're followed by lots of other accounts, but whether those accounts are themselves important.
- **Homophilly**: is the similarity between two connected users. Individuals are more likely to be connected to other individuals that have at least some similarity to them.
- Summary statistics of the users activity: how many hours she spend on platform, how many profile pages she visited, etc.

- **Betweenness Centrality** detects the amount of influence a node has over the flow of information in a graph. It is often used to find nodes that serve as a bridge from one part of a graph to another.
- **Degree Centrality**, which measures the number of relationships connected to a node. We can use this, to find out which users have the most followers, which is the metric that people most frequently reference when identifying social media influencers.
- Graph structure: graph density, graph diameter, clustering coefficient, Cheeger constant (to check if the graph has any bottlenecks) number of connected component, degree of strong-connectivity.
- User network connection recommendation: we can use a collaborate filter to recommend new connection to each user. This recommendation will be based on number of visit of a profile, number of common friend connection, some type of the users information/profile (demographic, region).

4 Business API



The **business partner** connects to the microservices through an **API Gateway**. The microsevice collects/aggregate the requested information by connecting to the database and Spark Cluster which I described earlier through Kafka messaging system. To speed up the process we can use a Redis cache for answering the most frequent requests.

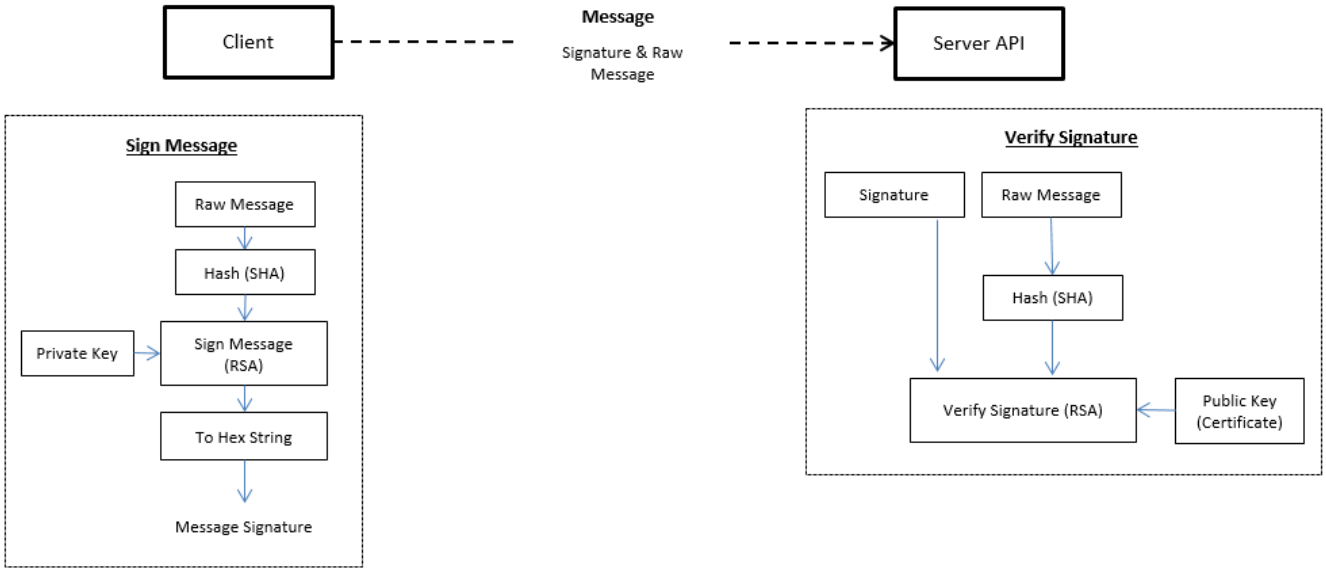
To incease the performance, scalability, and to improve the overall user experience we can use **Rate limits** (API Gateway). The Rate limits protects the API from becoming overloaded. If the API is the target of a malicious DoS attack, for example, it can go down entirely. Rate limiting allows API developers to ensure an API will reject requests that exceed a set limit. Rate limits also greatly help with scalability. We can use the following three types of rate limits:

- User-rate limit
- Time-based rate limiting
- Server rate limiting

4.1 API Security

Each request made to the API must attach some form of credentials which has to be validated on the server.

- **Digital Signature**: relies on private-public key pair is a useful mechanism for securing server to server communication. It works by signing the message content with a private key to produce a security signature that can be verified using the corresponding public key (certificate). A key pair is usually provided by a certificate authority.
  - The client will provide its public key to the server. Each request is then made to the server by signing the message content using the private key. This allows the server to verify the message authenticity without knowing the private key of the client.
  - Private keys must be kept private by the owner. The client holds the Private Key used to sign the message. The server holds the Public Key (Certificate) used to verify the message signature.
  - For two way communication, both servers will hold their own private keys and exchange public keys with each other.



5 Considerations

Decoupled applications: All processes should be decoupled to avoid a breakdown in case of failures. As an example, failure to process one set of data pipelines should not block processing the rest of the data pipelines.

Eliminate single point of failure: Single point of failure can lead to full system breakdown. Rather a multiple points of failure ensure the processing of non-impacted data pipeline as engineers work to resolve the failure. This also helps in preventing attacks like (DDoS). This approach should be implemented for both hardware and applications.

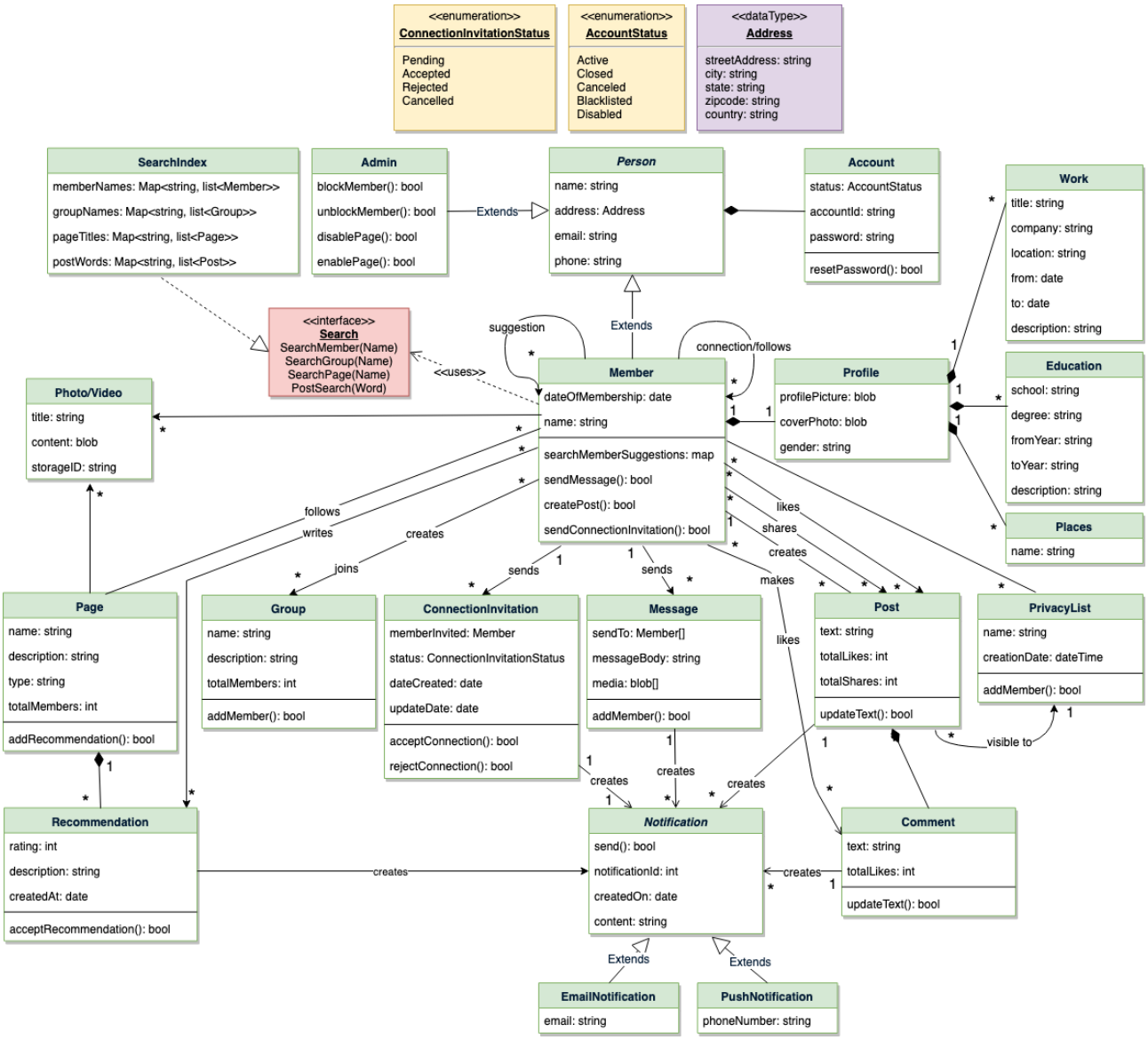
High Data availability: Replication across multi availability zone helps to achieve high data availability. Multi-region data replication ensures effective disaster recovery. For a business where users work across multiple regions, replicating data across the different regions can help in serving data faster as it lies closer to users or applications.

Governance and Auditing: Ability to apply governance rules, data immutability, and provide complete audit logs of data usage is critical and essential to meet regulatory and statutory requirements. Sensitive data must be protected, compliant with privacy laws and regulations.

- With the vast amount of data,it is difficult to track what data is available and may lead to data swamping. A solution to this is the Data catalog. **Data Catalog** is a collection of metadata combined with data management and search tools that helps analysts and other users to find the data that they need.The data catalog serves as an inventory of available data and provides information to evaluate fitness data for intended uses. To maintain a central data catalog and use it across various processing frameworks like Hadoop, Spark, and various other available tools. This ensures metadata integrity and applying easy data governance rules.

5.1 Appendix

As mentioned before the user profile related operations are handled by a user service. The user service sits on a MySQL database, Mysql because it is a very relational data from a structure point of view. The schema can be implemented as below:



5.2 References

Demetrescu, Camil, and Giuseppe F. Italiano. "A new approach to dynamic all pairs shortest paths." Journal of the ACM (JACM) 51.6 (2004): 968-992.

Abraham, Ittai, Shiri Chechik, and Sebastian Krinninger. "Fully dynamic all-pairs shortest paths with worst-case update-time revisited." Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2017.