Data Management in Large-Scale Distributed Systems

Storing Large Scale Graph Data

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

http://tropars.github.io/

2021

References

- The presentation slides of N. Bronson (TAO, Facebook's Distributed Data Store for the Social Graph)

In this lecture

- Managing Graph data

- Design of a large-scale geographically distributed database
  - Fast read requests
  - Low risks of inconsistency

Agenda

Introduction

Data Model

Working at scale

Facebook TAO

- Distributed Data Store for social graph
- Paper published by Facebook in 2013 (N. Bronson et al.)
  - Used to store and efficiently navigate through the data of a social media
  - A data model
  - An advanced replication + caching strategy to be able to go world scale
  - MySQL servers for storing data
- Evolution
  - Implement the database as a column-family LSM-tree based data store for better performance

Agenda

Introduction

Data Model

Working at scale

Social data representation

Representing people, actions and relationships

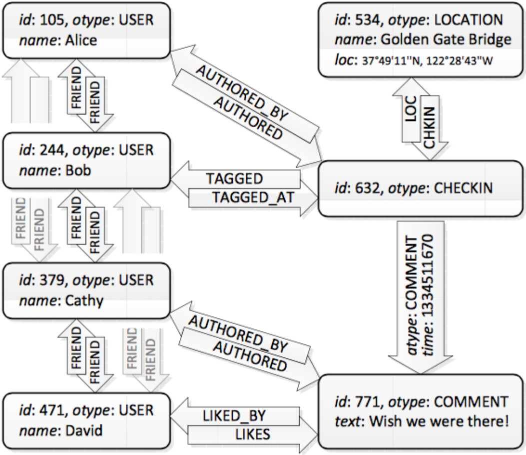
- = entities and connections
- Represented as nodes and edges in a graph

Data Model

Objects (e.g. user, place) with unique IDs

Associations (e.g. tagged) between two IDs

Both have key-value data as well as a time field



API

Object/Association API

- Allocate, update, delete objects/associations

Association Query API

- Starting from a tuple: (object , associationType)
  - Association List: (id1, atype) $\rightarrow [a_{new} \ldots a_{old}]$
  - Newer associations are returned first
- Examples of queries supported by the API:
  - assoc_get(id1, atype, id2set, high?, low?)
    - List associations between specific ids (with time bounds)
  - assoc_range(id1, atype, pos, limit)
    - Returns elements of the (id1, atype) association list with index $\in [pos, pos + limit]$
    - The 50 most recent comment on Alice's checkin: *assoc_range(632, COMMENT, 0, 50)*
  - assoc_time_range(id1, atype, high, low, limit)

Agenda

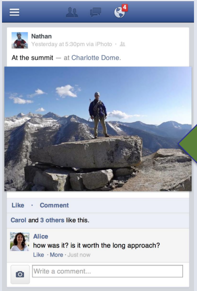
Challenges

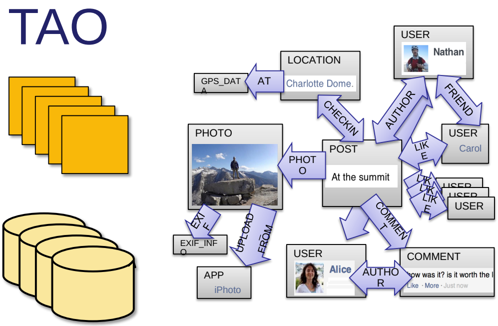
- A very large dataset
- A large number of read and write requests
- Many geographically distributed data centers across the world.

Main objectives

- Efficiency
  - Low read latency
- Consistency
  - Timeliness of writes
  - Achieve Read-After-Write consistency most of the time
- High read availability

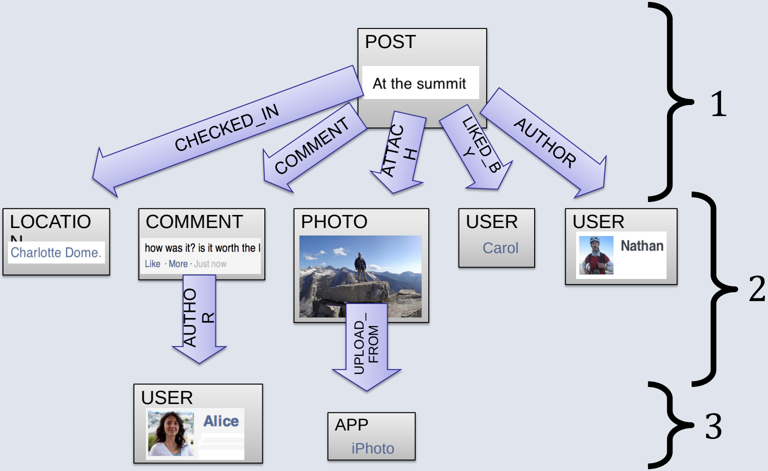
Dynamically Rendering the Graph





- 1 billion queries/second
- many petabytes of data

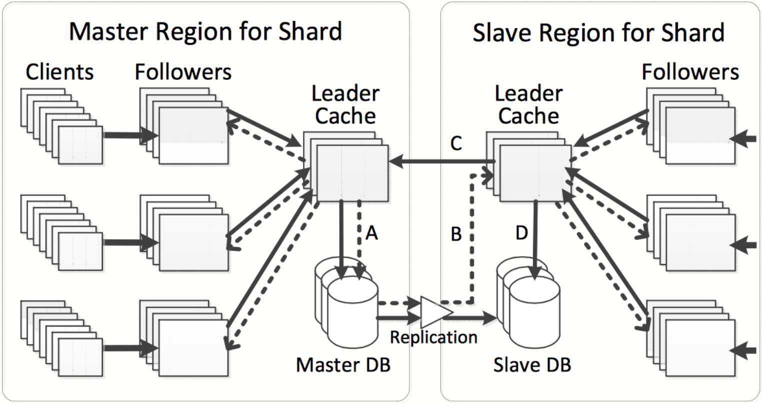
Dynamic Resolution of Data Dependencies



Design Principles

- The system is divided into regions
  - *Small* latency between data centers inside a region
  - A full copy of the social graph inside each region
- Replication of the data between regions
  - Master/slave replication
  - All writes go first to the master
- Data partitioning (sharding) over multiple database instances
- An in-memory cache is used to improve read performance
  - Based on Memcached KV store

Architecture



MySQL databases ➔ durability

Leader cache ➔ coordinates writes to each object

Follower caches ➔ serve reads but not writes

TAO's caching architecture

Caching tiers

- Multiple servers
  - A set of servers form a caching tier
  - Data distributed based on sharding inside a tier
  - Clients send request to the correct server depending on the target object id.

A hierarchical architecture

- A single *leader* tier and multiple *follower* tiers
- A client contacts the closest follower tier

Read and write operations

Assuming a single region

Read requests

- Served by the follower caching tiers
- Forwarded to the master tier in case of miss

Write requests

- Forwarded to the master caching tier (write-through strategy)
  - Improves timeliness of writes
- The master caching tier orders the updates to the same objects and apply them to the database
  - The issuing follower is updated synchronously
  - The other followers are updated periodically (eventual consistency)

Write operations and geo-distribution

Some numbers

- Regions can be 1000's Kms away (high latency)
- 25 times more reads than writes

Geo-distribution

- Read requests are always served locally (inside a region)
  - Risk of stale data but low latency
- Write request always go to the master region
  - Requests forwarded to the leader caching tier
    - Simple replication protocol
    - Other databases replicas are updated asynchronously
  - Local leader cache updated synchronously

More on geo-distribution

Load balancing

- A different region has the leader role for different shards
- A region can be the leader for multiple shards

Locality

- An association is stored on the shard of its `id1`
- Association queries are served by a single server

Is consistency good enough?

- In practice, 99.99% of reads to vertices return a consistent result
- See "*Existential Consistency: Measuring and Understanding Consistency at Facebook*, 2015."

Additional references

Suggested reading

- *TAO: Facebook's Distributed Data Store for the Social Graph*, N. Bronson et al., USENIX ATC, 2013.