L. Lancia
G. Salillari
Cloud Computing
Master Degree in Data Science
Sapienza Università di Roma

Facebook Tao

Distributed Data Store for the Social Graph

Table of Contents

Introduction

What is TAO?

Tao

is a geographically distribute store

- deployed at Facebook
- with efficient and timely access to social graph
- using a fixed set of query
- replacing memcache
- running on thousands of machines
- provide access to many PB of data
- process a billion reads ad millions of writes each second!

The social graph

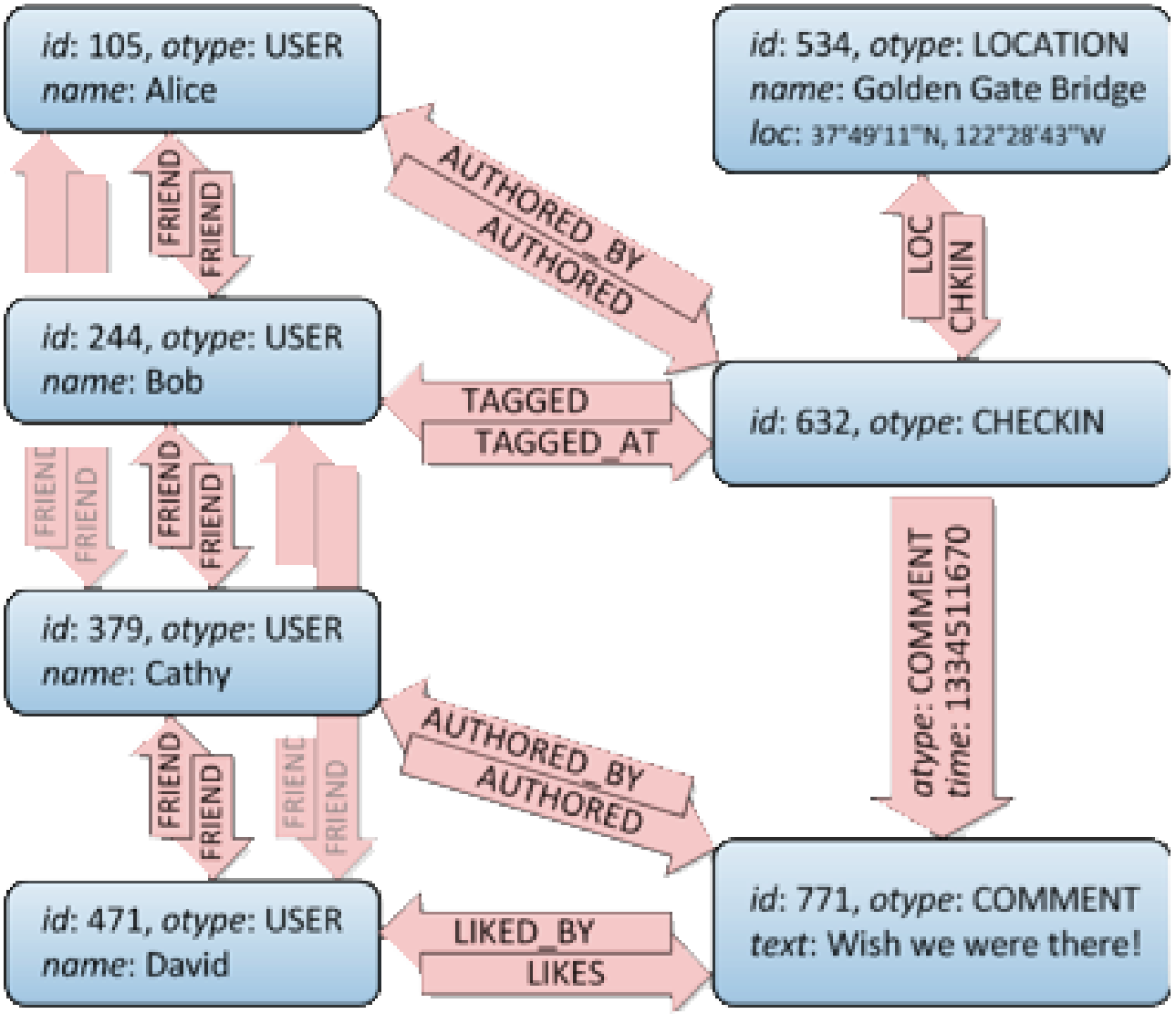
Facebook has more than 1 billion active user

- recording relationships,
- sharing interests,
- uploading pictures and ...

The user experience of Fb comes from rapid, efficient and scalable access to the *social graph*

What's behind an entry in yours Fb page?

A single Fb page aggregate and filter hundreds of items from the social graph.


id: 105, otype: USER
name: Alice
id: 534, otype: LOCATION
name: Golden Gate Bridge
loc: 37°49'11"N, 122°28'43"W
AUTHORED_BY
AUTHORED
FRIEND
FRIEND
LOC
CHKIN
id: 244, otype: USER
name: Bob
TAGGED
TAGGED_AT
id: 632, otype: CHECKIN
FRIEND
FRIEND
FRIEND
FRIEND
atype: COMMENT
time: 1334511670
id: 379, otype: USER
name: Cathy
AUTHORED_BY
AUTHORED
FRIEND
FRIEND
FRIEND
FRIEND
id: 471, otype: USER
name: David
LIKED_BY
LIKES
id: 771, otype: COMMENT
text: Wish we were there!

Before Tao

- Facebook was storing the social graph to MySql
  - Quering it from PHP
  - Storing result in memcache

Over time Fb deprecated direct access to MySQL in favor of a graph (associations, nodes) abstraction

Limits

- Operations on lists are inefficient in memcache (update whole list)
- Complexity on clients managing cache
- Hard to offer read-after-write consistency

Also they want to access social graph from non-PHP services

TAO's Goals

- Efficiency at Scale
- Low read latency
- Timeliness of writes
- High read availability

TAO's Goals

- Efficiency at Scale
- Low read latency
- Timeliness of writes
- High read availability

TAO's Goals

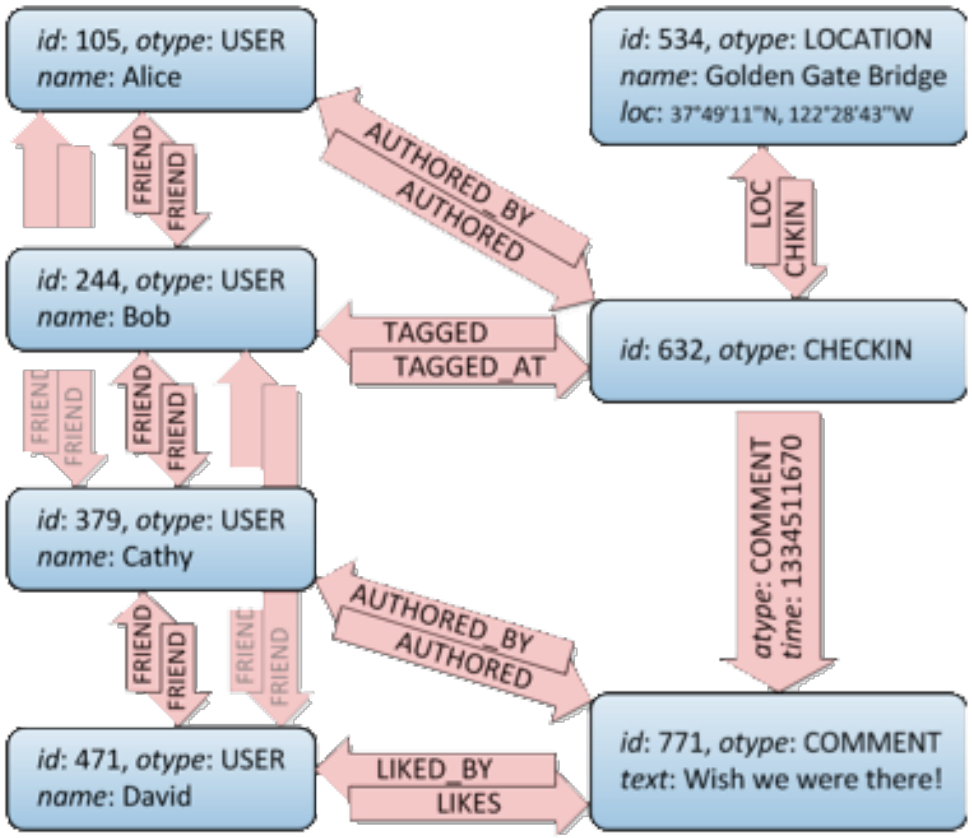
- Efficiency at Scale
- Low read latency
- Timeliness of writes
- High read availability

TAO's Goals

- Efficiency at Scale
- Low read latency
- Timeliness of writes
- High read availability

Tao Data Model

T.A.O. stands for “The Associations and Objects”



Objects

- Typed nodes (type is denoted by `otype`)
- Identified by 64-bit integers (unique)
- Contains data in the form of key-value pairs
- Models users and repeatable actions (e.g. comments)

API for objects:

- Allocate new object
- retrieve
- update
- delete

Objects

- Typed nodes (type is denoted by `otype`)
- Identified by 64-bit integers (unique)
- Contains data in the form of key-value pairs
- Models users and repeatable actions (e.g. comments)

API for objects:

- Allocate new object
- retrieve
- update
- delete

Associations

- Typed directed edges between objects (type is denoted by `atype`)
- Identified by source object `id1`, `atype` and destination object `id2`
- Contains data in the form of key-value pairs.
- Contains a 32-bit `time` field.
- Models actions that happen at most once or records state transition (e.g. like)
- Often inverse association is also meaningful (eg like and liked by).

Associations API

- Add new
- Delete
- Change type

Also inverse association is created or modified automatically

Querying TAO

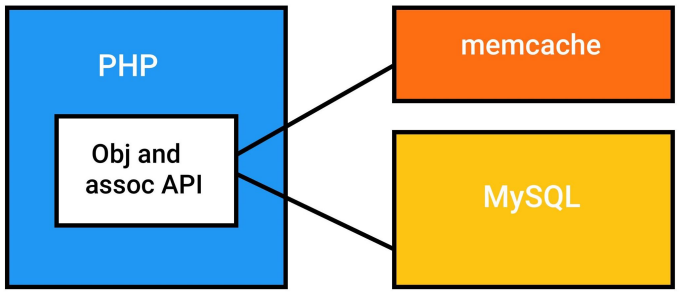
TAO's associations queries are organized around *associations list*

- `assoc_get(id1,atype, id2set, high?, low?)`
- `assoc_count(id1,atype)`
- `assoc_range(id1, atype, pos, limit)`
- `assoc_time_range(id1,atype, high, low, limit)`

Query results are bounded to 6000 results

Architecture

Before Tao

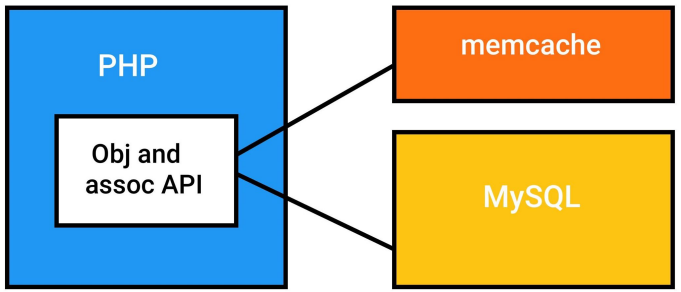


After Tao

Architecture

Before Tao

After Tao



Storage Layer

- Object and Associations are stored in MySql (before & with TAO)
- TAO API is mapped to a small set of SQL queries
- A single MySql server can't handle TAO volumes of data
  - We divide data into logical *shards*
  - *shards* are mapped to db
  - different servers are responsible for multiple shards
  - mapping is adjusted for load balancing
- Object are bounded to a *shard* for their entire lifetime
- Associations are stored in the *shard* of its `id1`

Implementation

title