Tao: Facebook's Distributed Data Store For The Social Graph
Bronson et. al., ATC 2013

Joy Arulraj
CMU 15-799 : Paper Presentation
Talk Overview

• Graph-aware cache backed by a database
  – Efficiency vs. consistency
Motivation

• Memcached
  – Distributed in-memory key-value store
  – Memory object caching system
  – Data mapping in client code (PHP API)
Limitations

• Association lists
  – Get entire list to update one edge

• Control logic
  – Clients manage lookaside cache
  – But, only have a local perspective

• Expensive read-after-write consistency
  – Writes forwarded to master
  – Local state updated asynchronously
Problem Statement

• Need a “smart” caching layer
  – Graph-aware
  – Distributed cache management
  – Provides read-my-write consistency

• Solution
  – Fix the API and leverage its constraints!
Example

*Alice* was at *CMU* with *Bob*

*Cathy*: Wish we were there!

*David* likes this

---

**Id**: 200 otype: User
Name: Alice

**Id**: 300 otype: User
Name: Bob

**Id**: 400 otype: User
Name: Cathy

**Id**: 500 otype: User
Name: David

**Id**: 600 otype: LOCATION
Name: CMU

**Id**: 700 otype: CHECKIN

**Id**: 800 otype: COMMENT
Text: Wish we were there!
Data Model

• Object
  – (id) -> (otype, (key->value)*)
  – Entities, repeatable actions
  – Ex: users, comments

• Association
  – (id1, atype, id2) -> (time, (key->value)*)
  – Relationships, actions that model state transitions
  – Ex: tagged at, likes
Data Model

• Association List
  – \((id1, atype) \rightarrow [a_{new}, \ldots, a_{old}]\)
  – Supports the Association Query API
  – Ex: ("CMU", "COMMENT")
API

• Association API
  – assoc_add(id1, atype, id2, time, (k->v)*)
  – assoc_delete(id1, atype, id2)

• Association Query API
  – [POINT] assoc_get(id1, atype, id2)
  – [RANGE] assoc_range(id1, atype, pos, limit)
  – [COUNT] assoc_count(id1, atype)
Client Queries

• All queries start from an <id, atype>

• 5 most recent comments on Alice’s checkin
  – assoc_range(“Alice”, “COMMENT”, 0, 5)

• Number of friends of Bob
  – assoc_count(“Bob”, “FRIEND”)
Tao’s Goals

- Low read latency
- Write consistency
- High read availability
Basic Architecture

Webservers
- Stateless

Cache servers
- Objects, Association Lists
- Partitioned based on <id>

Database
- Partitioned based on <id>
Low Read Latency

Webservers
- Too many network hops

Cache servers
- Hotspots with smaller shards
Datacenter-level Scalability

Tiers
- Distributed write logic

Database
- Thundering herds
Splitting the cache layer

Follower Cache

Leader Cache
Write Consistency

- **Followers**
  - Absorb read hits
  - Forward read misses and writes to leaders
  - Write-through cache

- **Leader updates**
  - Synchronously sent in reply to writer
  - Asynchronously sent to other followers
Write consistency

• Leaders
  – Serialize concurrent writes
  – Can prevent “thundering herds”

• Association list updates
  – Refills instead of invalidates
  – Idempotent pull-based incremental updates
Multi-datacenter Scalability

Forwarded writes

Async DB replication
High Read Availability

- **Follower failure**
  - Client contacts backup follower tier
  - May break read-after-write consistency

- **Leader failure**
  - Follower tiers reroute read misses directly to DB
  - Writes sent to another member of leader tier
Handling Hot Spots

• Consistent hashing
  – Simplifies cluster expansion
  – Request rerouting

• Load balancing
  – Shard cloning
  – Small client-side cache
Results

• Reads dominate writes
  – 99.8% read requests
  – 40% of requests are range queries

• Most edge queries have empty results
  – Tao can use cached assoc_count
  – Key advantage of app-aware caching
Results

• Availability
  – Fraction of failed queries : $4.9 \times 10^{-6}$

• Follower Throughput
  – 8 core Xeon + 144GB RAM + 10Gb Ethernet
  – 30-60K requests/sec
Tao Summary

• Low read latency
  – Application-aware cache layer

• Write consistency
  – Replication model

• High read availability
  – Fault-tolerance
Talk Summary

• Graph-aware cache backed by a database
  – Efficiency vs. consistency

• Why did they not use a graph database?
  – They trust MySQL
  – Tao’s cache layer handles their demands

Thanks!