the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

TAO: Facebook's Distributed Data Store for the Social Graph

MAY 19, 2015

tags: Datastores, Facebook, Graph, Social Networks

TAO: Facebook's Distributed Data Store for the Social Graph (https://www.usenix.org/conference/atc13/technical-sessions/presentation/bronson) Bronson et al. (Facebook) 2013

> *A single Facebook page may aggregate and filter hundreds of items from the social graph. We present each user with content tailored to them, and we filter every item with privacy checks that take into account the current viewer. This extreme customization makes it infeasible to perform most aggregation and filtering when content is created; instead we resolve data dependencies and check privacy each time the content is viewed. As much as possible we pull the social graph, rather than pushing it. This implementation strategy places extreme read demands on the graph data store; it must be efficient, highly available, and scale to high query rates.*

TAO is the graph store at Facebook that handles this load, continually processing a billion reads and millions of writes per second. All of Facebook runs off of a single instance of TAO in production, spread across several geographic regions. TAO is optimized heavily for reads, and favours efficiency and availability over consistency. It grew out of an existing system of memcaches in front of MySQL, queried in PHP. TAO continues to use MySQL for persistent storage, but mediates access to the database and uses its own *graph aware* cache.

Data model and API

TAO implements an *objects* and *associations* model. Objects are typed nodes in the graph, and associations are typed directed edges between objects. Objects and associations may contain data as key-value pairs, the schema of the corresponding object or association type determines the possible keys, value types, and default value. At most one association of a given type can exist between any two objects. Every association has a time field, which is critical for ordering query results by recency (e.g.

most recent comments on a post). An association may also have an inverse type configured – bidirectional associations are modelled as two separate associations. There are no restrictions on the edge types that can connect to a particular node type, or the node types that can terminate an edge type.

> *The starting point for any TAO association query is an originating object and an association type. This is the natural result of searching for a specific type of information about a particular object… TAO's association queries are organized around association lists. We define an association list to be the list of all associations with a particular id1 and atype, arranged in descending order by the time field. For example, the list (i, COMMENT) has edges to the example's comments about i, most recent first.*

Architecture & Implementation

All of the data for objects and associations is stored in MySQL. A non-SQL store could also have been used, but when looking at the bigger picture SQL still has many advantages:

> *…it is important to consider the data accesses that don't use the API. These include back-ups, bulk import and deletion of data, bulk migrations from one data format to another, replica creation, asynchronous replication, consistency monitoring tools, and operational debugging. An alternate store would also have to provide atomic write transactions, efficient granular writes, and few latency outliers.*

The space of objects and associations is divided into shards, each shard is assigned to a logical MySQL database with a table for objects and a table for associations. Objects of different types are therefore stored in the same table (with some separate custom tables for objects that benefit from separate data management). To avoid potentially expensive SELECT COUNT queries, association counts are stored in a separate table.

> *Each shard is contained in a logical database. Database servers are responsible for one or more shards. In practice, the number of shards far exceeds the number of servers; we tune the shard to server mapping to balance load across different hosts. By default all object types are stored in one table, and all association types in another. Each object id contains an embedded shard id that identifies its hosting shard. Objects are bound to a shard for their entire lifetime. An association is stored on the shard of its id1, so that every association query can be served from a single server. Two ids are unlikely to map to the same server unless they were explicitly colocated at creation time*

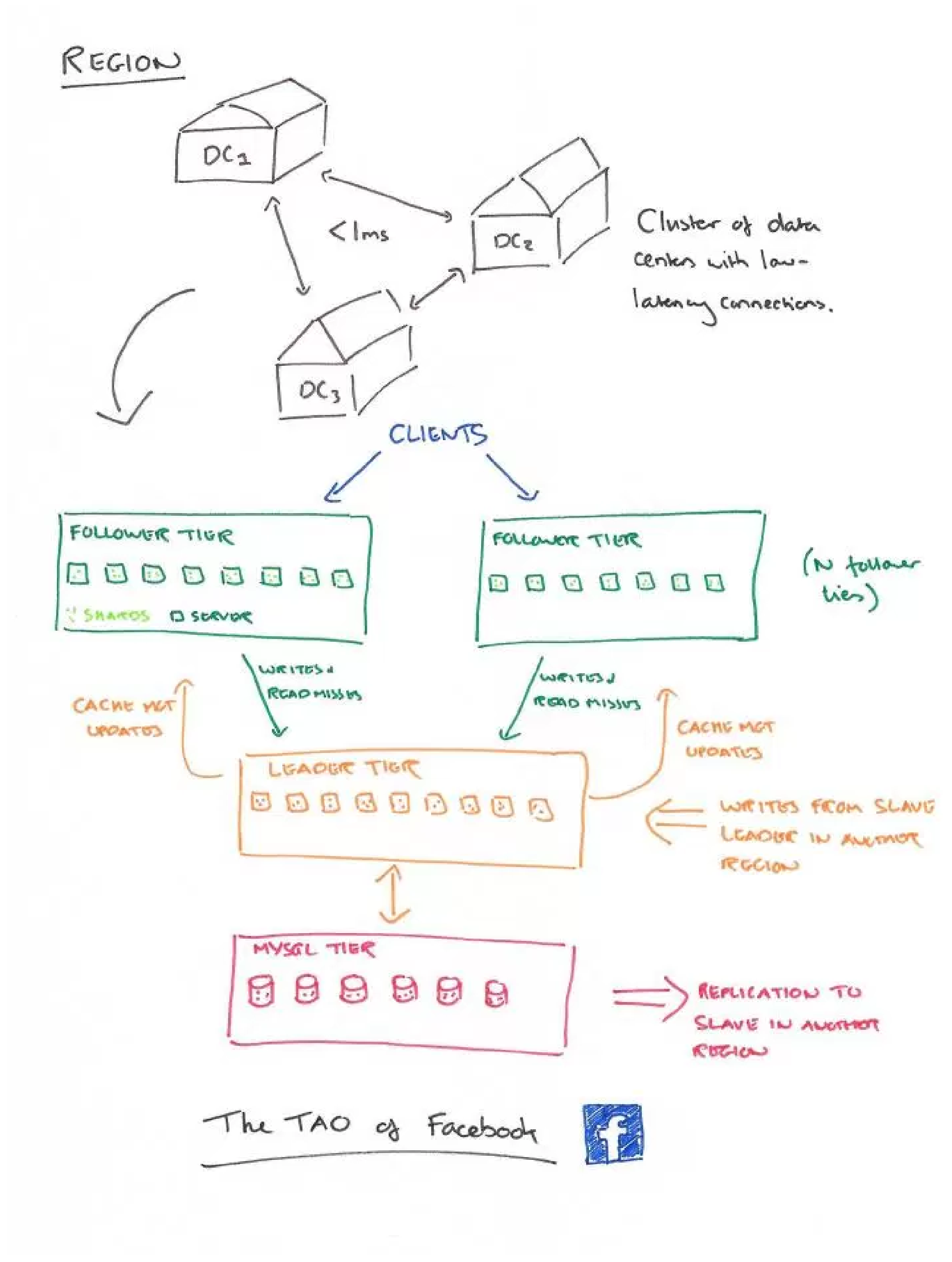
TAO's caching layer implements the full API for clients and handles all communication with the databases.

> *The social graph is tightly interconnected; it is not possible to group users so that cross-partition requests are rare. This means that each TAO follower must be local to a tier of databases holding a complete multi-petabyte copy of the social graph.*

It's too expensive to keep a full copy of the Facebook graph in every datacenter. Therefore several datacenters in a geographic region with low-latency interconnection (sub millisecond) between them are designated a *region*, and one copy of the graph is kept per-region. The master of a shard will have a slave in another region.

> *We prefer to locate all of the master databases in a single region. When an inverse association is mastered in a different region, TAO must traverse an extra inter-region link to forward the inverse write.*

Within a region, there is a two-layer caching hierarchy to help handle the load.



(https://adriancolyer.files.wordpress.com/2015/05/tao.jpg)

A region has several *follower* tiers. Each tier contains a number of TAO servers, responsible for all of the shards between them. Follower tiers forward writes & read misses to a single TAO leader tier. This leader tier is responsible for interacting with the MySQL tier that holds the MySQL databases for all of the shards. All interactions with the data tier go through the leader tier. The leader tier asynchronously sends cache management messages to followers to keep them up to date.

> *Write operations on an association with an inverse may involve two shards, since the forward edge is stored on the shard for id1 and the inverse edge is on the shard for id2. The tier member that receives the query from the client issues an RPC call to the member hosting id2, which will contact the database to create the inverse association. Once the inverse write is complete, the caching server issues a write to the database for id1. TAO does not provide atomicity between the two updates. If a failure occurs the forward may exist without an inverse; these hanging associations are scheduled for repair by an asynchronous job.*

Replication lag in normally less than one second, and TAO provides read-after-write consistency within a single tier. Writes to MySQL are synchronous so the master database is a consistent source of truth.

> *This allows us to provide stronger consistency for the small subset of requests that need it. TAO reads may be marked critical, in which case they will be proxied to the master region. We could use critical reads during an authentication process, for example, so that replication lag doesn't allow use of stale credentials.*

TAO deliberately supports a restricted graph API. Facebook has large-scale offline graph processing systems similar in spirit to Google's Pregel, but these operate on data copied from TAO's databases and not within TAO itself.

> *TAO is deployed at scale inside Facebook. Its separation of cache and persistent store has allowed those layers to be independently designed, scaled, and operated, and maximizes the reuse of components across our organization. This separation also allows us to choose different tradeoffs for efficiency and consistency at the two layers, and to use an idempotent cache invalidation strategy. TAO's restricted data and consistency model has proven to be usable for our application developers while allowing an efficient and highly available implementation.*

from → Datastores, Distributed Systems, Large Scale Systems

7 Comments leave one →

Trackbacks

1. FaRM: Fast Remote Memory | the morning paper
2. Scalability For May 22nd, 2015 | blog1
3. GraphX: Graph Processing in a Distributed Dataflow Framework | the morning paper
4. Fast Database Restarts at Facebook | the morning paper
5. Holistic Configuration Management at Facebook | the morning paper
6. Existential Consistency: Measuring and Understanding Consistency at Facebook | the morning paper
7. JavaScript for extending low-latency in-memory key-value stores | the morning paper