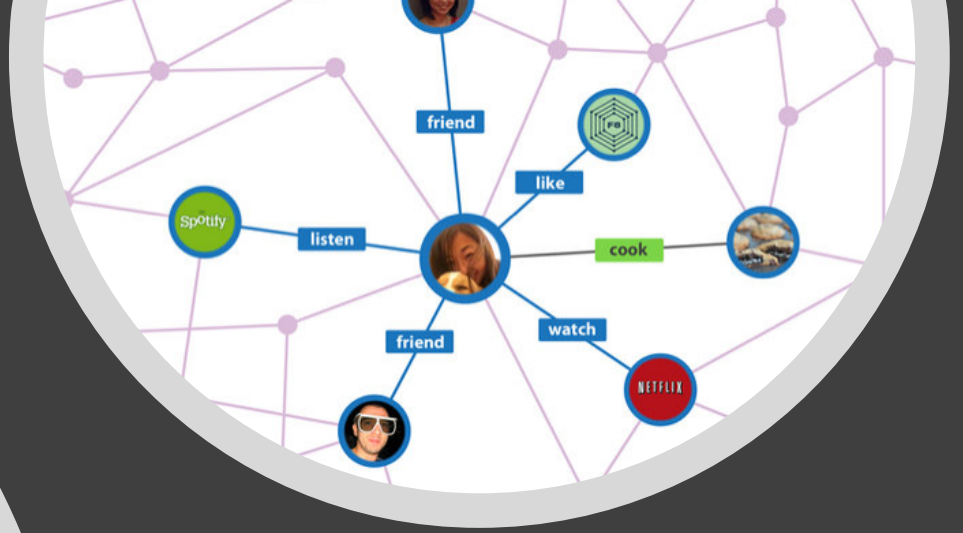
Tao: Facebook's Distributed Data Store for the Social Graph

Authors : Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov
Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov
Dmitri Petrov, Lovro Puzar, Yee Jiun Song, Venkat Venkataramani
Facebook, Inc.

Presented By : Tuhin Tiwari

“I don’t know whether it was the nature of the paper but it took so long to read this paper. Every single line seems important” - Anonymous

Image Source



The Social Graph, where Facebook's billion active users record every detail of their lives. Most users consume more content than they create, resulting into a read-heavy workload.

Memcache

- Initially, Facebook's web-servers used memcache as a lookaside cache.
- This lookaside cache was used for all the reads & writes, directly accessing MySQL.
- When a web-server needed data, it first requests from memcache and higher cache-hit rate meant good performance.

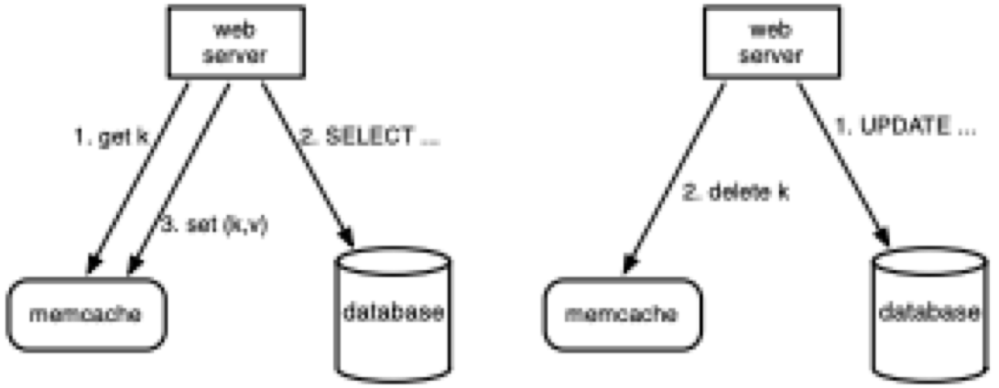
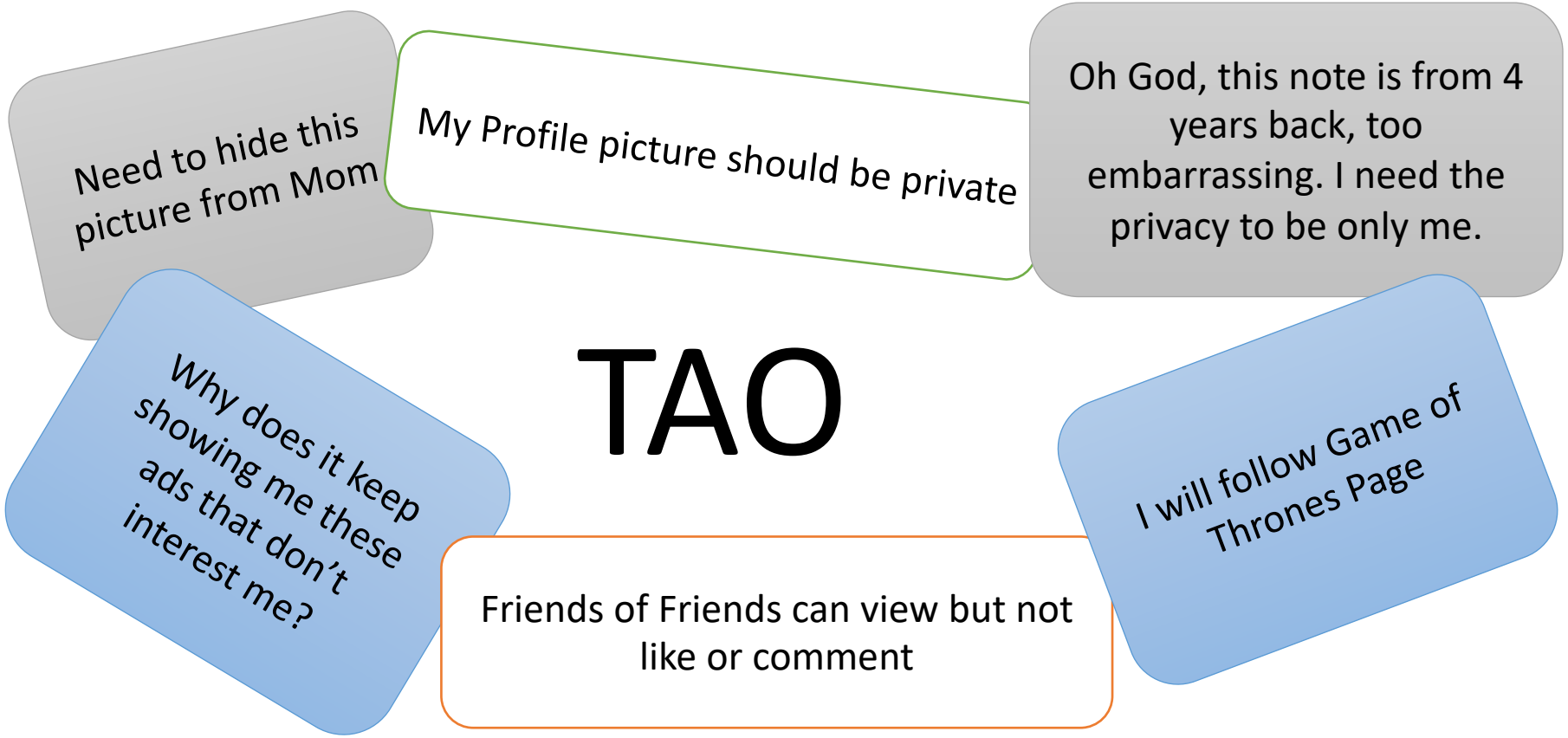


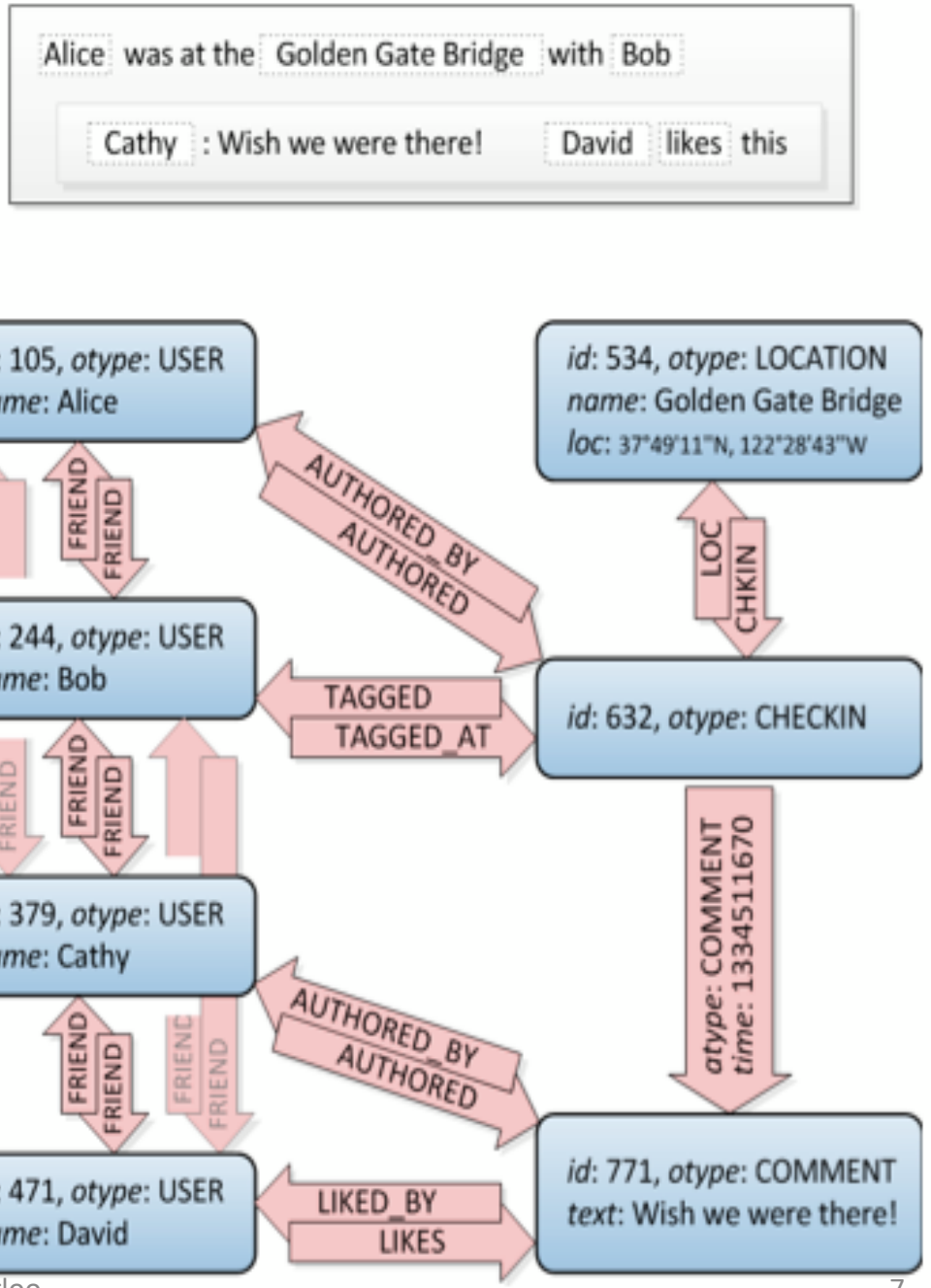
Figure 1: Memcache as a demand-filled look-aside cache. The left half illustrates the read path for a web server on a cache miss. The right half illustrates the write path.

Customize, Aggregate & Filter



Social Connections & Access

- Most information on Facebook is best represented using a social graph.
- The rendered content is extremely customizable depending on the users privacy settings and preferences. Data needs to be stored as-is and filtered when viewed/rendered.
- The representation of the information, on the right, as a key-value store like lookaside cache becomes time-consuming.
- TAO resolves data-dependencies and check privacy every time content is viewed. Social graph is pulled and not pushed.



Problems with lookaside architecture

- Inefficient edge lists : Key-value cache doesn't fit for list of edges. Changes to a single edge would require fetching the entire edge list and then reloading the list. – Other possible alternatives?
- Distributed control logic : No communication between clients where control logic of lookaside is run. In an object-associations model, TAO system controls everything and it can move the control logic into the cache itself which solves the problem.
- Expensive read-after-write consistency : Asynchronous master/slave replication for MySQL. Time elapsed between writes to the master and the local replica. TAO updates the replica's cache at write-time.

TAO Data Model & API

- TAO provides Objects & Associations as basic units of access in the system.
- Data model consists of two main entities:
  - Object : TAO objects are typed-nodes. Mapped from a “unique id” to “object type, and key-value pairs”.
  - Association : TAO associations are typed directed edges. Identified by the “source object, association type, and destination object”, which is mapped to a “32-bit time field and key-value pairs”.

Object: $(\text{id}) \rightarrow (\text{otype}, (\text{key} \rightarrow \text{value})*)$
Assoc.: $(\text{id1}, \text{atype}, \text{id2}) \rightarrow (\text{time}, (\text{key} \rightarrow \text{value})*)$

APIs – Objects & Associations

- Object API – Allocates a new object & id and performs all operations with that id.
- Association API – Bidirectional edges are modeled as two separate associations. Associations configured with an “inverse type” and different operations modify the link between two object ids accordingly with an association.

Association Query API

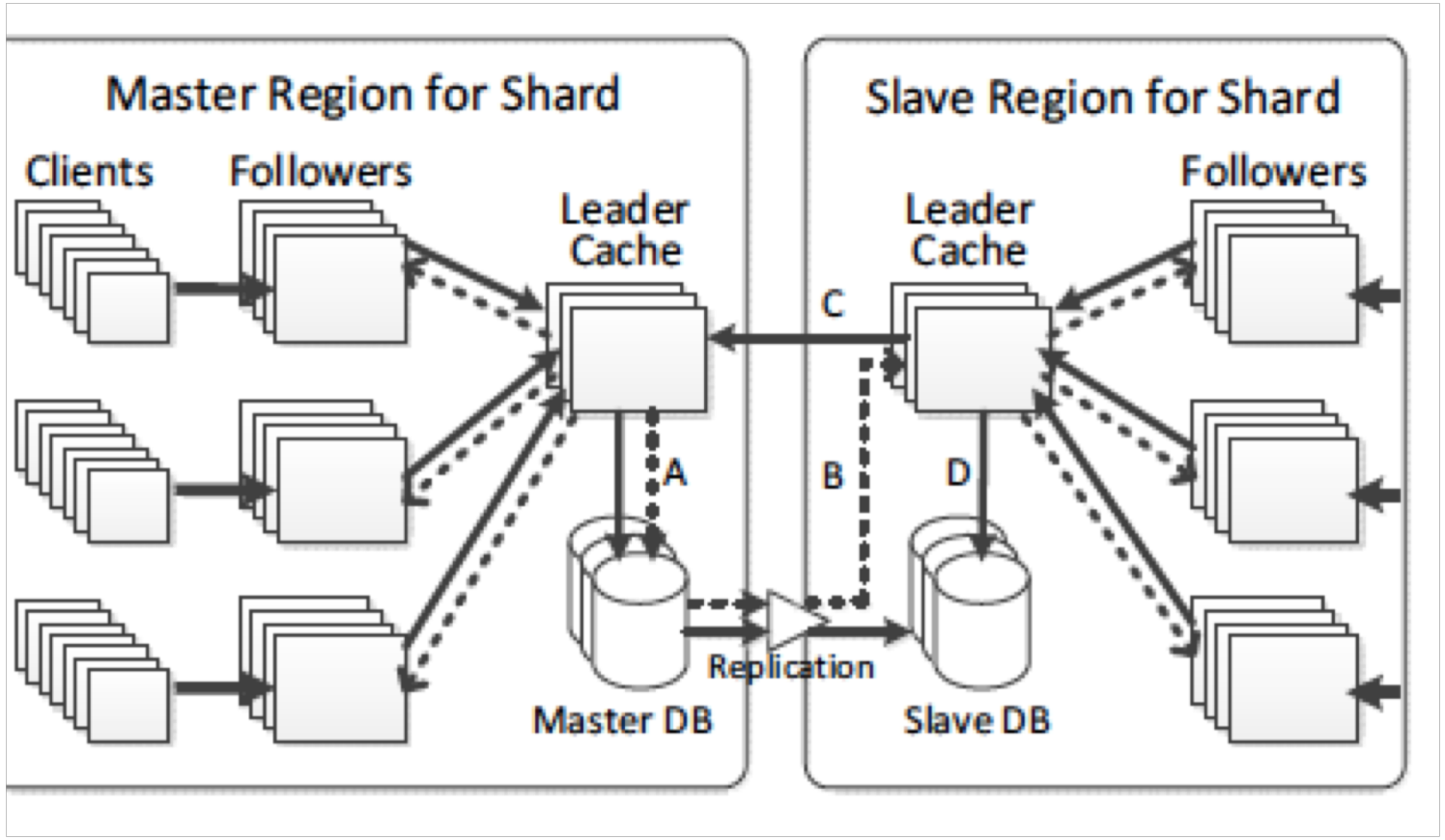
Query : *"Show me the most recent 50 comments on Alice's check-in"*	• This can be modeled as assoc_range(CHECKIN_ID, COMMENT, 0, 50)
Query : *"How many check-ins at the Golden Gate Bridge?"*	• This can be queried by assoc_count(CHECKIN_ID, CHECKIN)
Query : *"Show me the comments on Alice's check-in posted between 1 PM and 2 PM"*	• This is modeled as assoc_time_range(CHECKIN_ID, CHECKIN, 13, 14)

TAO Architecture – Storage Layer

- MySQL for persistent storage to consider data accesses that don't use the TAO API like backup, bulk import, migrations, etc.
- Other systems like LevelDB didn't fit their needs in this regard.
- To handle the large volume of data, data is divided into logical shards.
- One server to many shards.
- Tune the shard to server mapping for load-balancing.
- All object types – stored in one table and all association types separate in another table.
- Objects – shard_id & association is stored on shard of its id1(originating object). This ensures better locality and helps with retrieving objects and associations from the same host.
- Objects data – serialized and stored against id. Association – key : id and data being serialized and stored into one column.

TAO Architecture – Caching Layer

- Client requesting information connects to cache which implements the complete API for all communications with databases.
- Cache misses & write requests are requested from other caches/databases.
- In-memory cache holds objects, association lists and association counts.
- LRU policy to fill and evict cache on demand.
- Inverse association-write operation may involve two shards of caches.



Client Communication Stack

- Each cache belongs to a tier consisting of multiple such caches and the database.
- Adding more caching servers to a tier will make it fatter and hence prone to hot spots.
- Communication cost grows quadratically.

TAO Architecture Leaders & Followers

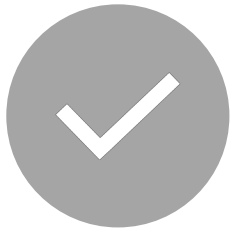
- A two level hierarchy divides the region into multiple follower tiers but only 1 leader tier.
- Read misses & writes are always handled by Leader.
- Read hits by follower, where request is first landed or by another follower tier.
- Consistent hashing alleviates the hotspots which makes addition of tiers easier without rebalancing caches too much.
- Followers can offload read workload for popular celebrities(objects) to the client and clients can cache them for longer.
- Single leader ensures consistency and by mediating all the requests from id1, it also protects the database from thundering herds.

Geographical Scaling

- High workload is handled by the leader-followers configuration.
- **Assumption** – Network latencies from follower to leader and leader to database are low.
- For data center located in China and North America, network round trip can become a bottleneck.
- Master-slave architecture with write to master & reads handled locally.
- TAO follower must be local to a tier holding a complete copy of social graph – infeasible(expensive)
- **Solution** : Choose data center locations that are clustered into a few regions (reducing latency)
- Writes forwarded by the local leader to the leader with the master database.
- All reads can be satisfied with the compromise of returning stale data to the clients. Querying the same follower will give the user a consistent view of TAO

Implementation – Caching Servers

Layer between clients & databases



Memory Management : Slab allocator, thread-safe hash table, LRU eviction among items of equal size, dynamic slab rebalancer (maintains LRU ages consistent throughout)

Partition RAM into arenas by association or object types

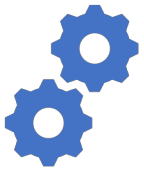
Partitioning of RAM extends cache lifetime of important types

To prevent memory overhead of pointers for small items(in the main hash table), these items are stored separately using direct-mapped 8-way associative caches with no pointers. Sliding the entries down tracks the LRU order in each bucket.

Optimization : Additional table for mapping active ‘atype’ allows to hold 20% more items in cache by recording absence of id2 and only taking 10 bytes instead of 14.

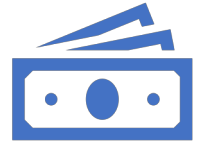
MySQL Mapping

Fields of an object table allocated in a shard are stored in a single ‘data’ column, serialized.



Objects benefitting from separate data management policies are stored in separate custom tables.

Associations – similar to objects, but their tables have additional indexes to support range queries.



Saving on the expensive “SELECT COUNT” queries, association counts are stored in a separate table.

Cache Sharding & Hot Spots

Consistent hashing used to map shards to cache servers within a tier

Load imbalance on followers might get created due to semi-random assignment of shards.

TAO rebalances followers with shard cloning (reads to a shard served by multiple followers in a tier)

Cloning can also help distribute the load of a popular object queries.

If access rate of an object exceeds a threshold, TAO client caches the data and version.

The access rate can also be used to throttle client requests for very hot objects.

High-degree Objects

- TAO doesn’t cache the complete association list for objects that have more than 6000 associations with the same atype.
- For an assoc_get query returning empty result, high-degree objects almost always go to the database as id2 could be in the uncached tail.
- TAO’s solution :
  - Choose assoc_count to determine direction of the query and also leveraging application-domain knowledge to improve cacheability.
  - Limit the search to associations who time is greater than object’s creation time.

Consistency

- Eventual consistency : After a write, TAO guarantees eventual delivery of an invalidation and a refill to all tiers.
- Replication lag < 1 second
- Synchronous update to the cache by returning a changeset from a master leader when write is successful. All associated shards(inverse and ones with Slave leader) should be updated before returning to the caller.
- The race condition while applying changeset to follower's cache might arise the risk of cache being stale. TAO resolves this by keeping a version number(both in persistent storage & cache) that is updated during each update.
- Rare but can happen? – Slave region's storage server to receive an update takes longer than it does to evict. (risk of evicted value being reloaded)
- Not strong consistency to single source of truth, MySQL. It gives us the chance to give consistency for smaller subset of requests.

Failure Detection & Handling

Database failures : If the master goes down, one of the slaves automatically comes up. Global configuration maintained to mark down crashes. When slaves are down, misses are redirected to TAO leaders hosting the DB master. Additional binlog trailer is run on the Master database, and the refills & invalidates are delivered inter-regionally.

Leader failures : Followers reroute read misses directly to the database. Writes are routed to a random replacement leader performs the operations and also enqueues asynchronous invalidation to the original leader.

Failure Detection & Handling

Refill & invalidation failures : Problem of permanent failure of leader is solved by sending bulk invalidation operations that invalidates all objects and associations from a shard_id.

Follower failures : TAO follows a primary and backup follower tier. Followers in other tiers share the responsibility.

Production Workload

Multi-tenancy :	Amortization in operational costs Multi-tenancy enables new applications to link to existing data Allows 64-bit id of an object to be handled uniformly without a step to resolve otype
Observations on 6.5 mn requests :	Reads more frequent than writes Most edge queries have empty results Query frequency, node connectivity and data size have distributions with long tails

Production Workload Statistics

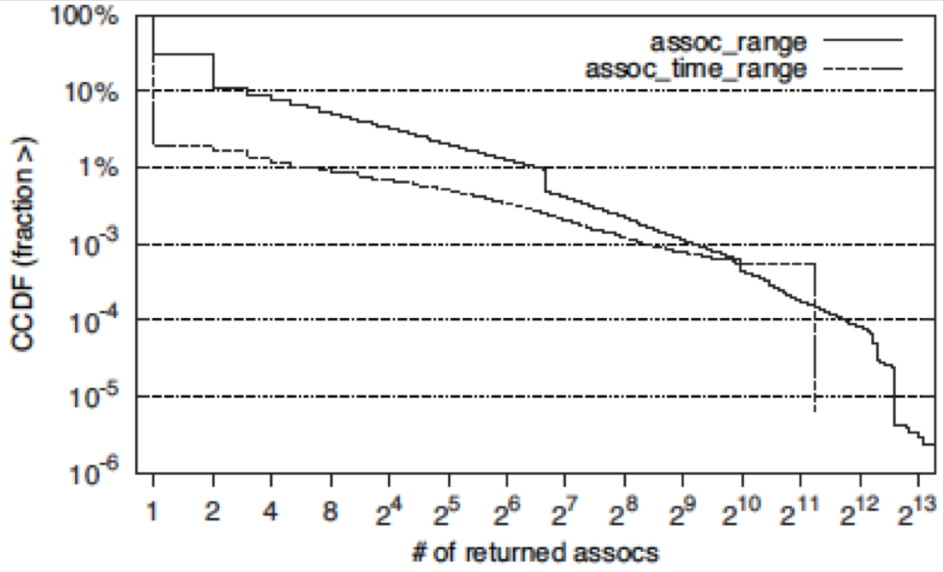


Figure 5: The number of edges returned by assoc_range and assoc_time_range queries. 64% of the non-empty results had 1 edge, 13% of which had a limit of 1.

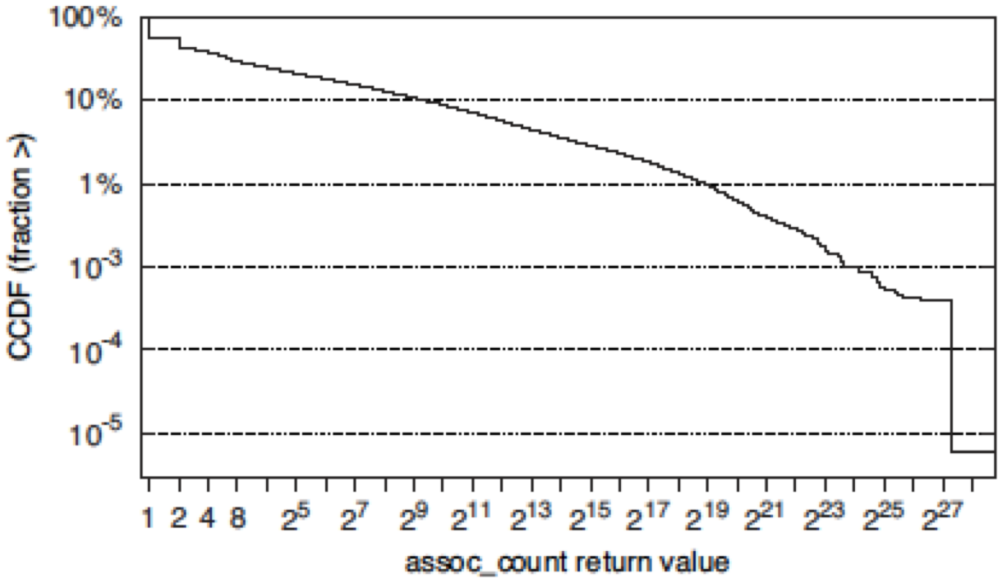


Figure 4: assoc_count frequency in our production environment. 1% of returned counts were ≥512K.

Distribution of the Data Sizes for TAO query results

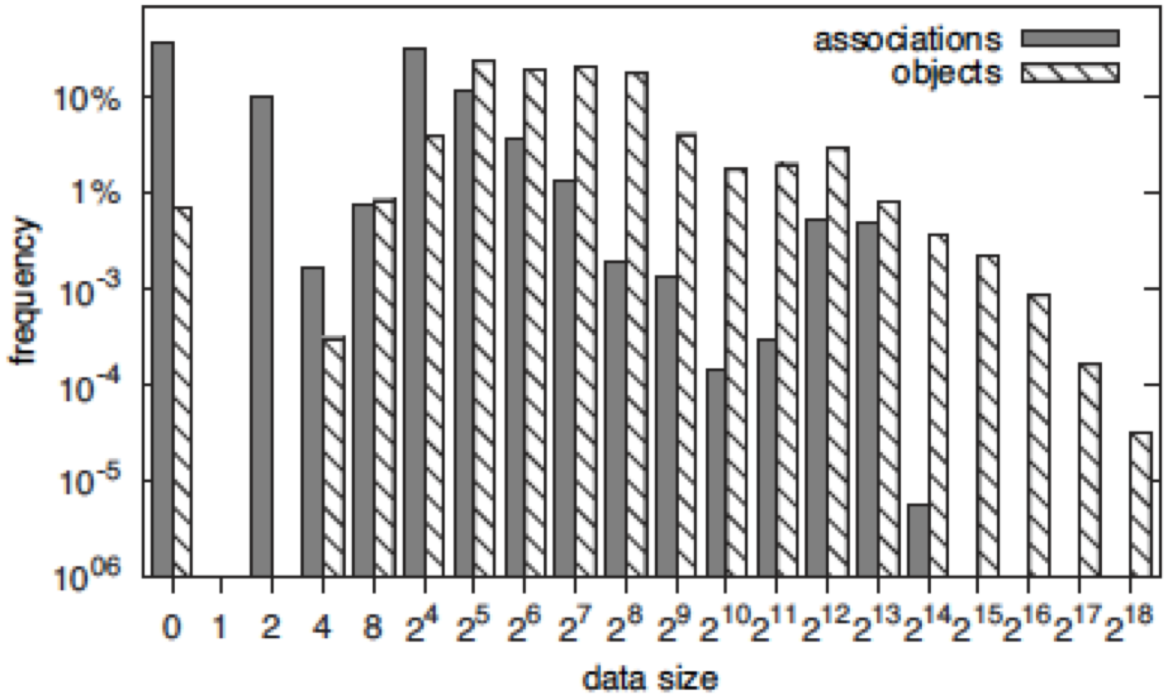


Figure 6: The size of the data stored in associations and objects that were returned by the TAO API. Associations typically store much less data than objects. The average association data size was 97.8 bytes for the 60.5% of returned associations that had some data. The average object data size was 673 bytes.

Performance

- Availability : Over a period of 90 days, the fraction of queries failed from the web server was fairly small but even a small fraction might have dynamic data dependence on the rest of the social graph.
- Follower Capacity : Peak throughput dependent on its hit rate (Figure 7)

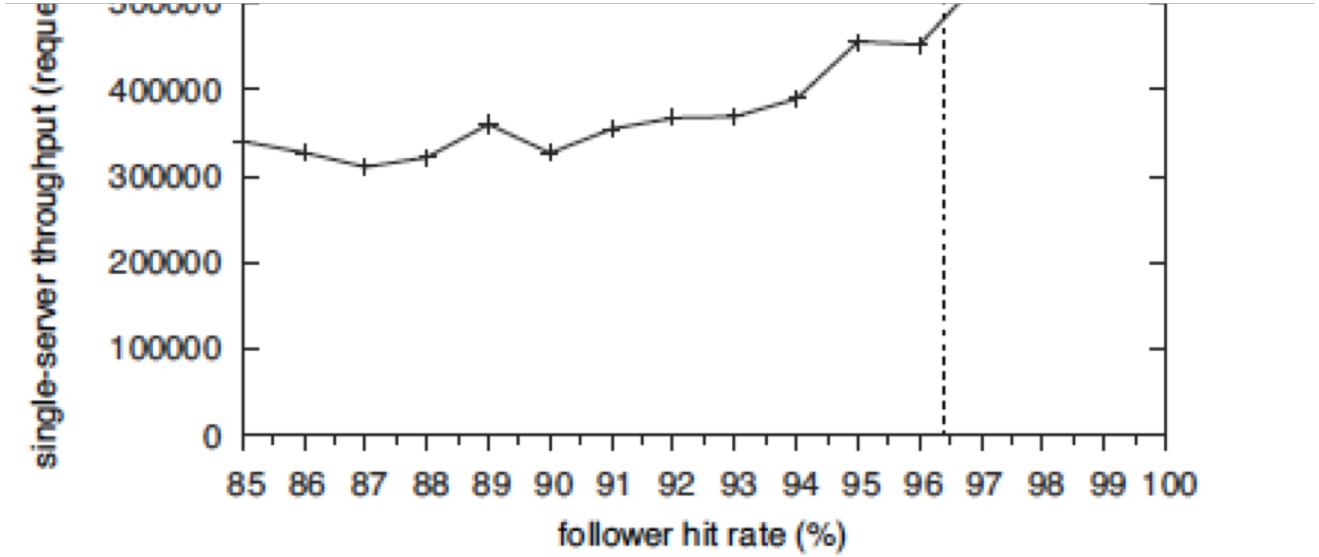


Figure 7: Throughput of an individual follower in our production environment. Cache misses and writes are more expensive than cache hits, so the peak query rate rises with hit rate. Writes are included in this graph as

Hit rates & latency

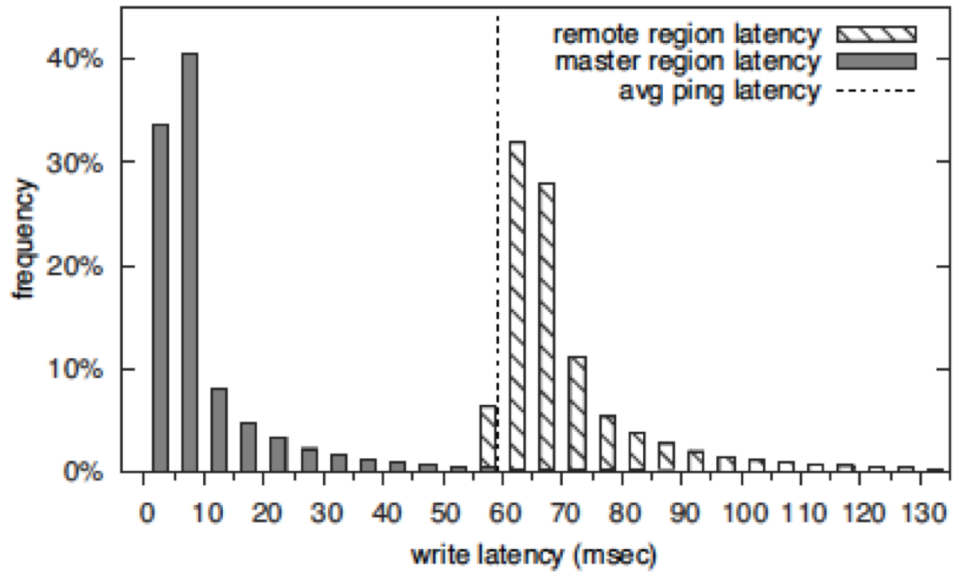


Figure 9: Write latency from clients in the same region as database masters, and from a region 58 msec away.

	hit lat. (msec)			*miss lat. (msec)*		
operation	*50%*	*avg*	*99%*	*50%*	*avg*	*99%*
assoc_count	1.1	2.5	28.9	5.0	26.2	186.8
assoc_get	1.0	2.4	25.9	5.8	14.5	143.1
assoc_range	1.1	2.3	24.8	5.4	11.2	93.6
assoc_time_range	1.3	3.2	32.8	5.8	11.9	47.2
obj_get	1.0	2.4	27.0	8.2	75.3	186.4

Figure 8: Client-observed TAO latency in milliseconds for read requests, including client API overheads and network traversal, separated by cache hits and cache misses.

Replication lag & Failover

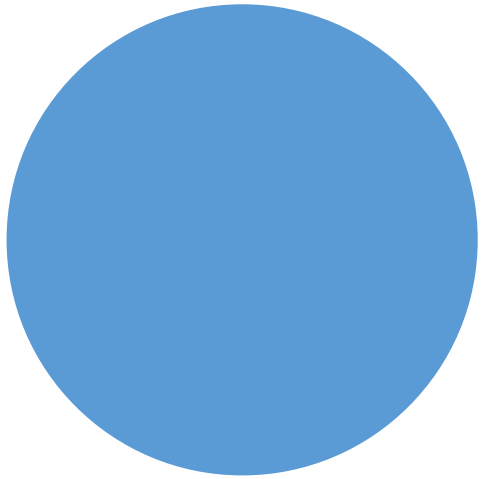
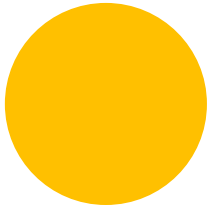
TAO's slave storage servers lag their master by less than 1 second during 85% of the tracing window.

Less than 3 seconds – 99% time and less than 10 seconds – 99.8%

Failover caches contacted the leader 0.15% of follower cache misses over our sample.

Related Work

- Eventual Consistency
- Geographically distributed data stores
- Distributed hash tables and key-value systems
- Hierarchical connectivity
- Structured storage
- Graph serving
- Graph Processing

Conclusion

Questions

- RAM arenas are configured manually to address specific caching problem, will automating it add to cost-overhead?
- Lack of atomicity between two updates of write operations might result into hanging associations if failure occurs. Is there a better way to resolve this than
- Challenges of graph databases.
- Will this model work if Facebook starts behaving like Twitter and transactions are equally write-heavy as well?
- Why is TAO API not suitable for other tasks such as replication, backup and migration?
- Is eventual consistency good enough?

References

- N. Bronson, et al., Tao: Facebook's Distributed Data Store For The Social Graph, *Proc. USENIX Annual Technical Conference*, pages 49-60, 2013
- D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent Hashing and Random trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In Proceedings of the 29th annual ACM Symposium on Theory of Computing, STOC, 1997
- R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation, NSDI, 2013.
- W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don’t settle for eventual: scalable causal consistency for widearea storage with COPS. In T. Wobber and P. Druschel, editors, Proceedings of the 23rd ACM Symposium on Operating System Design and Implementation, SOSP. ACM, 2011.

- Neo4j. http://neo4j.org/.
- https://medium.com/coinmonks/tao-facebooks-distributed-database-for-social-graph-c2b45f5346ea
- Facebook Research. https://research.fb.com/