CS 591: Graph mining **Spring 2020**

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

Student: Athanasios Filippidis *Final Project*

1 Introduction

The subject of this project is a read-optimized graph data store, called TAO (The Associations and Objects), which was implemented by Facebook and it was published in 2013[1]. TAO is a system that relies heavily on caching to serve large numbers of low-latency queries. It implements a graph data model in which nodes are identified by fixed-length persistent identifiers (64-bit integers) and it encodes a specific mapping of its graph model to persistent storage and takes responsibility for persistence. In this data model, the main components are objects and associations. Associations naturally model actions that can happen at most once or record state transitions, such as the acceptance of an event invitation, while repeatable actions are better represented as objects. TAO also exposes a minimal API that does not support a complete set of graph queries, but provides sufficient expressiveness to handle most application needs while allowing a scalable and efficient implementation.

The motivation behind this paper involves a strict set of requirements behind the company's product. Assuming that the reader is familiar with Facebook, someone can easily imagine the volumes of data that are being received and sent every second. So, in order to effectively serve this tremendous amount of workload they had to move on from the Memcached[2] caching implementation they were using to something more sophisticated that would be a better caching solution for a social graph, that would provide a better distributed control logic and that would offer read-after-write consistency. One intriguing part behind this is the level on which Facebook's business model has affected the design of the application. What this means, is that due to the nature of Facebook's product, it is tolerable to have some inconsistencies in the content that is presented to the user while it is not tolerable to have high latency or unavailability. And those two are some of the main accomplishments TAO achieves. However, TAO is not useful only for this specific product. It follows some design principles that enable it to be capable of being used in a variety of other products that need to efficiently generate fine-grained user customized content from a highly interconnected data.

2 Related Work

The final suggested application by the authors touches upon various aspects of distributed database systems so there will be a short mention of each one of those and some related work of each of those aspects.

i) Eventual consistency: Eventual consistency is a relaxed form of consistency in distributed systems.

Bayou: The authors mention as a related example Xerox's Bayou[3] which was designed and published in 1995. One extremely interesting fact about Bayou is that it was designed to support a consistent storage between multiple mobile devices, many, many years before that was even a problem. It requires from the developers of the applications that utilize its mechanisms to provide automated conflict resolving rules. TAO, while it provides eventual consistency as well, it provides it by serializing the write transactions through a single leader for every shard of the data. There are two more pieces of related work that are relevant to this section.

DynamoDB: The first one is DynamoDB[4], Amazon's highly available key value store which utilizes consistent hashing and eventual consistency in order to provide an "always online" experience to the user. The main innovation behind DynamoDB is that even in the case of natural disasters the system is created in a way that will still be available for the user to create write transactions. This specification makes sense if someone sees that from Amazon's perspective. Being the biggest e-commerce platform, they want at any cost to ensure that when a user adds something to their shopping cart this write transaction will not be lost no matter what. The main way through which DynamoDB is related to TAO is that they both utilize a more relaxed consistency model in order to be able to provide availability, throughput and low latency. Moreover, TAO uses consistent hashing as well in order to effectively split its data in multiple shards.

Existential consistency: The second one is a paper on existential consistency[5], written by USC and Facebook that was published in 2015 - the authors mention that it had been deployed since 2012. What makes this paper really interesting is that it actually studies the consistency model of TAO. More specifically, the authors were able to prove that TAO, even though utilizing eventual consistency (which is one of the most relaxed consistency models), is still highly consistent (99.99% of the results were the same as the ones that would have been served under a strict consistency model).

ii) Geographically distributed data stores:

PNUTS : Yahoo!'s PNUTS[6] was one of the first systems that was able to achieve a low latency and a rich database functionality at a massive scale. It is a system that is more consistent than TAO but has a worse latency performance (TAO's average read latency is approximately 2.5ms and write latency is approximately 35ms while PNUTS's average latency is approximately 60ms). This is mainly due to the fact that PNUTS is not focused that much in caching as much it is in keeping a consistent "master" node for each of the records instead of each of the shards which is the case in TAO. One big innovation for their time was the message broker, a topic-based pub/sub system which is optimized to keep up to date geographically distributed replicas.

Megastore and Spanner: Spanner[7], which is the evolution of Google's Megastore[8], is a globally distributed and synchronously replicated database (so it is related from that perspective to TAO) that offers externally-consistent distributed transactions (which is something that TAO sacrifices over performance). It was initially created to support Google's F1[9], the back-end of Google's advertising system and it was later evolved into a very useful tool for many other components as well. The main innovation behind it is TrueTime, an API on which the authors base their consistency model. TrueTime, in simple words, is the authors saying "we know that we can never have strictly synchronized clocks on the servers of all our data centers all over the world. However, we can build a reliable system based on the assumption that this clock skew will be always strictly bounded". TrueTime exposes exactly that fact and provides those "strict error bounds" which then Spanner utilizes to provide this novel product.

iii) Distributed hash-tables and key-values systems: The fact that TAO saves data in the caches as key-value tuples, is related strongly to previous work.

DynamoDB: One of the most famous examples of that is DynamoDB which was mentioned above. Two correlations between TAO and DynamoDB from this section's perspective is the high availability and the minimal API that is exposed to the developers in both systems. However, the most important one is an optimization that is briefly mentioned in the DynamoDB paper. This optimization adds a buffer layer on top of DynamoDB's storage which, according to the authors, can lead to a decrease in write latency by a

factor up to 5 (the 99.99th percentile latency with this optimization disabled is approximately 100ms, while with it enabled is approximately 50ms). While DynamoDB is write-oriented and TAO is read-oriented we can clearly see a pattern that leads to better latency performance by sacrificing durability in the first system and consistency in the second.

iv) Graph serving and processing: TAO, as mentioned before, is a product oriented towards Facebook's social graph. This fact leads to some correlations to graph databases.

Neo4j: One of the most famous databases in the category of graph databases is Neo4j[10]. Neo4j is designed to store fixed size vertices and edges. The main innovation of it is the direct pointers. This means that every vertex in the database stores some pointers to the actual physical locations of its neighbors. This leads to an important optimization because neighborhood queries and traversals can be executed without any kind of indexing which, in its turn, leads to their complexity being correlated to the size of the visited subgraph instead of the graph size.

FlockDB: Another graph store is Twitter's FlockDB[11]. FlockDB shares a lot of common attributes with TAO, although their goal is not the same. FlockDB is an in-storage system (in contrast to TAO which is in-memory) that is used to store the social graph of Twitter. It has an almost identical data model to TAO, which is reasonable considering how similar those two social network applications are. It also utilizes SQL for its storage, same as TAO. One difference is that TAO, from a design perspective, is an in-memory system that is built on top of a storage engine in order to provide a level of abstraction and better (mainly read) performance. FlockDB, on the other hand, is closer to being simply a storage engine, so someone could think of them as being in two different layers of design. An interesting future project could be to see whether TAO could work on top of FlockDB. One more noteworthy point regarding FlockDB is that Twitter actually utilizes some caching technologies on top of it as well. More specifically, they have developed **Twemcache**[12], a variation of Memcached that has been adjusted to Twitter's needs. The main difference between TAO and Twemcache seems to be the eviction policy. TAO uses LRU eviction while the official Twemcache blogpost mentions that they use random eviction. However, in their official code repository, there are multiple eviction modes options. Unfortunately, there is no published paper for Twemcache so it is not possible to compare its performance to TAO.

PeGaSus: PeGaSus[13] is a graph mining package. While TAO does not provide any kind of advanced graph processing, its simple API can still be categorized as such. The important idea behind PeGaSus that makes it special is the realization that many primitives behind different graph processing operations are common. The authors abstracted this common functionality, named it Generalized Iterative Matrix-Vector multiplication and added some optimizations on it which lead to a 5x speed improvement. This, in combination with the fact that they utilized Hadoop, provided the ability to perform graph mining in extremely large graphs in a timely manner.

Pregel: Google's Pregel[14] is an iterative computational model that is focused on the distributed, scalable, fault-tolerant processing of large graphs. Again, while not directly connected to TAO one may see some correlation since they both operate in a similar environment.

v) Other Facebook's related work:

Unicorn: Facebook's Unicorn[15] is a graph-based indexing system that provides a richer (from a semantics perspective) graph queries API. It is used in combination with TAO and Wormhole in order to answer queries on Facebook's social graph.

Wormhole: Facebook's Wormhole[16] is a publish-subscribe system. Its main functionality is to identify new writes in Facebook's system and to publish those updates to all the different components of Facebook's back-end. The main innovation here is twofold. The first part is its performance; it achieves a throughput of 50 million messages per second which is something that no other pub-sub system was able to achieve by that time. The second part is its adaptability. Facebook uses multiple storage systems as MySQL, HDFS and RocksDB as well as multiple systems that have to communicate and receive updates from those storages – one of them being TAO. Wormhole is created in a way that it is not oriented towards a specific product and it is able to be used with any kind of new products. While not mentioned in TAO's paper, since it was published before Wormhole, TAO is strongly based and related to Wormhole in order to receive new write transactions in a timely manner.

3 Proposed - Implemented functionality and Critical discussion

Implemented functionality: The implementation of this project has two main goals. The first one is to show that there exists an in depth understanding of the system and its purpose. The second one is to provide to a future reader a basic, easy-to-read implementation of TAO that can be used to create a real-life, open-source implementation of it. In order to enhance the aspect of readability the language that was used is Python. Following the initial project proposal, this implementation is focused on the storage, the cache and the API aspect of TAO and not in the distributed system aspect of it.

Critical Discussion: After studying and finishing the implementation of it there are some key points that can be discussed further. The main motivation behind TAO is Facebook's realization that Memcached was not the ideal caching option for their product and that they could benefit more from a caching system that represented the social graph using the actual primitives of it – objects and association or simply - vertices and edges. Then, considering the continuous growth of the company, they had to specify the set of features that this system should provide. This is a problem that was approached from many different aspects. One aspect was that of the workload. They knew that 99.8% of the transactions they receive are reads and only the 0.2% is writes. That clearly proved the need for TAO to be heavily read-optimized. Another aspect that they had to think of was the trade-off between consistency and performance. This is a problem that distributed systems designers have to face always. Ideally, one would want to have both. However, it is proven that while there exist state-of-the-art systems that perform amazingly in either of the two features, it is almost impossible to provide both of them, especially when someone thinks of the volume of the data such a system is expected to handle nowadays. Reasonably, the authors chose performance because they understood that their average client will usually not notice and always will not care if a like comes in the wrong order or if two persons in two different sides of the planet do not see the exact same post simultaneously. In the same manner, they chose availability before consistency since, again, someone would care significantly more if they entered Facebook and they got stuck in a loading screen instead of just noticing some inconsistencies.

TAO is using a typical distributed systems leader-followers combination where every follower can answer read queries while the write transactions have to go through the leader who is responsible to update the storage with the new value and to invalidate the followers - caching servers in order to avoid serving stale values. This is a technique used by many other systems as well. A very interesting innovation that is mentioned in this paper regarding those caches is the partitioning of RAM in segments. Considering that TAO is above all a caching system, there should be a quite novel discussion about the actual caching policy. Unfortunately,

the only two things that are mentioned regarding caching are the following. Firstly, the authors mention that they partition their RAM in order to extend the cache lifetime of important objects or associations. For example, a post that is observed to be accessed by many clients at once periodically (for example that could be a post that provides the link to a daily White House report – as someone may imagine almost nobody will visit that in the middle of the night, however it may be useful to keep it in cache instead of reloading it every day from storage) should remain in cache since it is expected to be heavily read by the clients. This is a very interesting idea and with the modern machine learning trends this partitioning could be performed automatically instead of manually, as mentioned that is done in the paper. The second thing that is mentioned is the actual caching policy that is being used, the Least Recently Used (LRU) policy, which simply evicts the item that has not being accessed for the longest time when it gets full. This is a reasonable choice since this way the most "popular" objects are kept in cache which leads to the biggest "saving" in terms of storage I/Os. However, from the operating systems perspective, it is known that there exist multiple other policies as well that can be proven useful under the right circumstances. Considering how important part is that of the caching policy there could have been a further analysis on it.

At this point it should be mentioned that there is a wide design space for in-memory databases and caches that has been explored by Chasseur and Patel[17], and that the authors could have provided a more thorough analysis on that, but the lack of it might indicate corporate privacy issues.

4 Experiments

Setup:

- OS: macOS Catalina 10.15.4
- Python: 3.7
- CPU: 2.2 GHz 6-Core Intel Core i7
- RAM: 16 GB 2400 MHz DDR4
- GPU: Radeon Pro 555X 4 GB

Findings: This was intended to be a reference implementation (on which future implementations can be based for better understanding of the requirements and the components of TAO) rather than an experimental implementation (and that was one of the reasons that Python was used instead of C/C++, which would have led to a finer grained control of the memory and the resources, and to faster execution times). For that reason, I believe that this implementation's performance results may be misleading and I did not perform extensive experimentation. However, the reader can get an abstract notion of the performance from the following figure.

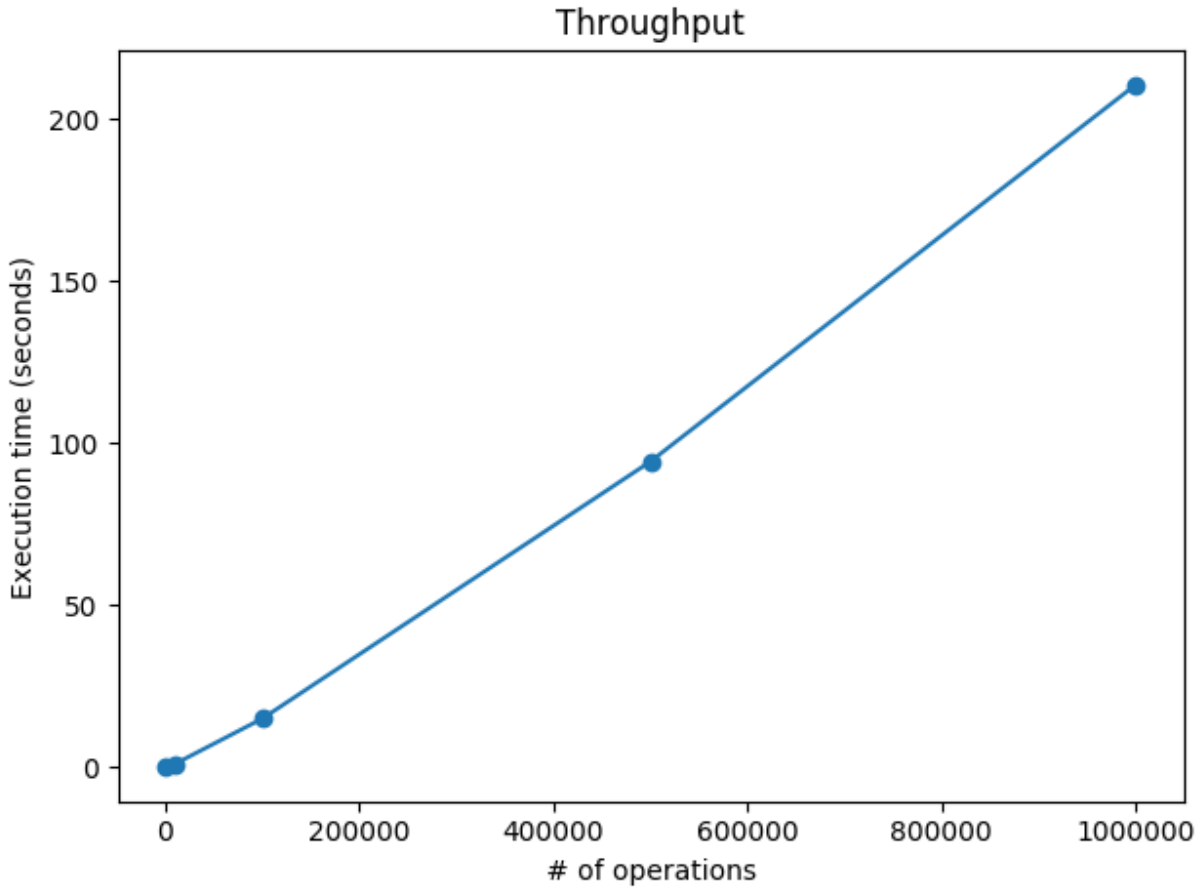

Figure 1: Operations throughput. This test was made with a fixed size of caches equal to 100 element each. As someone may easily notice the throughput rate stays approximately the same as the number of operations increases.

5 Conclusion

Summary: This project had as a goal to study in depth Facebook's TAO and its underlying mechanics. TAO is a read-optimized data store that was developed to replace Facebook's previous caching solution – Memcached. It was designed to fit exactly Facebook's needs – which, however, does not exclude it from being useful in other similar applications as well. Those needs were availability, heavily read-oriented performance and scalability. The main feature that was sacrificed to achieve those in the tremendous levels of workload Facebook has to deal with is consistency. Nevertheless, later work has proven that TAO's eventual consistent model provides a noteworthy level of consistency too. This makes TAO one of the few distributed systems that were ever created to provide such a rich set of features in such a big scale.

What I learned: I can happily say that this project widened my research interests. More specifically, I started going through some of the related work in order to get a better understanding on how significant the features offered by TAO are, and I ended up reading multiple papers varying from distributed systems to graph databases. Particularly graph databases seem to be a very promising field, especially considering how

some of the most demanding modern applications have to deal with social graphs and I plan on continuing studying them. I also studied and learned every small piece of TAO in order to be able to reproduce its expected behavior in my non-distributed implementation of it that can be found here[18]. Moreover, I spent time realizing the design decisions behind all of those pieces in order to be able to successfully translate them in Python and to apply any available optimizations in various pieces.

Open Problems - Future Work: The main area in which I believe there might be some space for future improvement for this project (which however might have been done already and not been published yet) is on the actual implementation of the caches. While the authors briefly mention the model that is being used, there is no mention of the underlying structure. There are many options at this point, from the row-stores and column-stores perspective (which however might not be so useful in this case considering that the data is kept in key-value pairs), from the block-based and file-based organization perspective as well as from the compression perspective. It would be extremely interesting to see a future paper addressing how those different options work in a caching model and how they can be connected to TAO.

References

[1] Nathan Bronson et al, TAO: Facebook's Distributed Data Store for the Social Graph,
`https://www.usenix.org/system/files/conference/atc13/atc13-bronson.pdf`

[2] Rajesh Nishtala et al, Scaling Memcache at Facebook,
`https://www.usenix.org/system/files/conference/nsdi13/nsdi13-final170_update.pdf`

[3] Douglas B. Terry et al, Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System,
`https://people.eecs.berkeley.edu/~brewer/cs262b/update-conflicts.pdf`

[4] Giuseppe DeCandia et al, Dynamo: Amazon's Highly Available Key-value Store,
`http://courses.cse.tamu.edu/caverlee/csce438/readings/dynamo-paper.pdf`

[5] Haonan Lu et al, Existential Consistency: Measuring and Understanding Consistency at Facebook,
`http://sigops.org/s/conferences/sosp/2015/current/2015-Monterey/printable/240-lu.pdf`

[6] Brian F. Cooper et al, PNUTS: Yahoo!'s Hosted Data Serving Platform,
`http://www.cs.bu.edu/~jappavoo/jappavoo.github.com/451/papers/cooper-pnuts.pdf`

[7] James C. Corbett et al, Spanner: Google's Globally-Distributed Database,
`https://static.googleusercontent.com/media/research.google.com/el//archive/spanner-osdi2012.pdf`

[8] Jason Baker et al, Megastore: Providing Scalable, Highly Available Storage for Interactive Services,
`http://cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf`

[9] Jeff Shute et al, F1: A Distributed SQL Database That Scales,
`https://static.googleusercontent.com/media/research.google.com/el//pubs/archive/41344.pdf`

[10] Neo4j,
`https://neo4j.com/`

[11] FlockDB,
`https://blog.twitter.com/engineering/en_us/a/2010/introducing-flockdb.html`

[12] Manju Rajashekhar et al, Caching with Twemcache,
`https://blog.twitter.com/engineering/en_us/a/2012/caching-with-twemcache.html`

[13] U Kang et al, PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations,
`https://www.cs.cmu.edu/~ukang/papers/PegasusICDM2009.pdf`

[14] Grzegorz Malewicz et al, Pregel: A System for Large-Scale Graph Processing,
`https://www.cs.cmu.edu/~pavlo/courses/fall2013/static/papers/p135-malewicz.pdf`

[15] Michael Curtiss et al, Unicorn: A System for Searching the Social Graph,
`http://www.vldb.org/pvldb/vol6/p1150-curtiss.pdf`

[16] Yogeshwer Sharma et al, Wormhole: Reliable Pub-Sub to Support Geo-replicated Internet Services,
`https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-sharma.pdf`

[17] Craig Chasseur , Jignesh M. Patel, Design and Evaluation of Storage Organizations for Read-Optimized Main Memory Databases,
`http://www.vldb.org/pvldb/vol6/p1474-chasseur.pdf`

[18] Athanasios Filippidis, Project's Github repository,
`https://github.com/tflpd/TAO`