Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 4, No. 2, April : 2024

A COMPARATIVE STUDY OF MEMCACHE, TAO, AND TAOBENCH END-TO-END BENCHMARK FOR SOCIAL NETWORK LIKE META

Rajiv Kumar Ranjan, Sankalp Sonu, Ankita Sinha, Assistant Professor, Department of Computer Science and Engineering, RashtrakaviRamdhari Singh Dinkar College of EngineeringBegusarai, State – Bihar (India), Pin-851134

Abstract.
Modern Web services rely extensively upon a tier of in-memory caches to reduce request latencies and alleviate load on backend servers. Memcached is a well known, simple, in-memory caching solution. Leveraging Memcached significantly enhanced the memory efficiency of caching the social graph, enabling scalable and cost-effective solutions. Nevertheless, the implementation complexity for product engineers in managing data storage and retrieval increased substantially. My paper gives a comparative study Meta utilizes Memcached as a foundational component to create and expand a distributed key-value store, facilitating the support of the world's largest social network. The drawbacks associated with its use and the introduction about the new technology TAO ("The Associations and Objects"). TAO, a geographically dispersed data storage system, offers Meta efficient and timely access to the social graph through a predefined set of queries, catering to its high-demand workload. Implemented at Meta, it supplants Memcached for numerous data types aligned with its framework. The ongoing rise of large-scale social network applications has brought about a level of data and query volume that pushes the boundaries of existing data storage capabilities. Than TAOBench, was introduced to emulate the social graph workload found at Meta.
Key words: Memcache, TAO, TAOBench, Social Network

1. Introduction

It is expensive always to fetch data from the source database due to resource constraints. The infrastructure challenges posed by widely-used and captivating social networking platforms are substantial. With hundreds of millions of daily users, these networks place demanding computational, network, and I/O requirements that traditional web architectures find challenging to meet. The infrastructure of a social network must enable (1) nearly instantaneous communication, (2) dynamic aggregation of content from various sources; (3) efficient access and updating of highly popular shared content, and (4) scalability to handle millions of user requests per second **Amazon SimpleDB (2023), Meta – Company Info (2023), Phillipe Ajoux et al. (2015), Audrey Cheng et al.(2022).**
Introducing a "caching layer" typically makes data access times faster and improves the ability for the underlying database to scale to accommodate consistently higher loads or spikes in data requests **P. Saab. Scaling memcached at Meta(2008), Gregory Chockler et.al (2010).** Memcached serves as a straightforward, exceptionally scalable cache based on keys, storing data and objects in available dedicated or surplus RAM for rapid access by applications. It is an open-source, distributed memory caching system designed to tackle today's web-scale performance and scalability challenges. Many of the largest and most heavily trafficked web properties on the Internet like Meta, Fotolog, YouTube, Mixi.jp, Yahoo, and Wikipedia deploy Memcached and MySQL to satisfy the demands of millions of users and billions of page views every month **Amazon SimpleDB (2023), Meta – Company Info (2023), Memcached (2023), P. Saab. Scaling memcached at Meta(2008).**
With over a billion active users engaging in recording relationships, sharing interests, and uploading diverse content, Meta initially had its web servers directly interact with MySQL for reading or writing the social graph. Memcache was extensively employed as a lookaside cache. In efforts to enhance the open-source Memcached, Meta transformed it into a fundamental element, constructing a distributed key-value store for the world's largest social network. Meta introduced TAO **Rajesh Nishtala et al. (2013), Meta – Company Info (2023)** a straightforward data model and API

uniquely designed to serve the social graph. TAO still relies on MySQL for persistent storage but manages access to the database through its specialized graph-aware cache. Deployed as a solitary geographically distributed entity at Meta, TAO boasts a streamlined API and prioritizes availability and machine-level efficiency over strict consistency. Its remarkable feature lies in its scalability, capable of supporting a billion reads per second on a dynamic dataset spanning many petabytes.

TAOBench provide open-source workload configurations along with a benchmark that utilizes these request characteristics to faithfully replicate production workloads and simulate emerging application behaviors. To ensure the reliability of TAOBench's workloads, it is validated against their production counterparts.

In this paper I have explained the working of Memcached and the problem associated with it, the new model TAO and TAOBench there after gives the comparison between them.

1.1. Using MySQL with Memcached

Throughout history, data caches have played a vital role in database services. Websites facing exceptionally high volumes of web requests opt for distributed memory object caching services like Memcached to ease the burden on their databases and enhance response times.

The advantages of utilizing Memcached encompass various aspects.

- Due to the storage of all information in RAM, the access speed surpasses that of fetching data from disk every time.
- The absence of data type restrictions in the "value" part of the key-value pair allows for caching a diverse range of data, including complex structures, documents, images, or a combination of such elements.
- In the scenario of using the in-memory cache for transient information or as a read-only cache for data also stored in a database, the failure of a Memcached server is not critical. For persistent data, an alternative lookup method employing database queries can be employed, reloading the data into RAM on a different server.

In this architecture, each client is set up to communicate with all servers. When a client initiates a request to store data, the key used for data referencing undergoes hashing, and the resulting hash is employed to designate one of the Memcached servers. This server selection occurs within the client before any interaction with the server, ensuring a streamlined process.

Upon subsequent requests for the same key, the identical algorithm is employed, resulting in the generation of the same hash and selection of the same Memcached server as the data source. This method ensures that cached data is evenly distributed across all Memcached servers, allowing access to cached information from any client. Consequently, a distributed, memory-based cache is established, facilitating the rapid retrieval of information, especially complex data and structures, compared to directly querying the database.

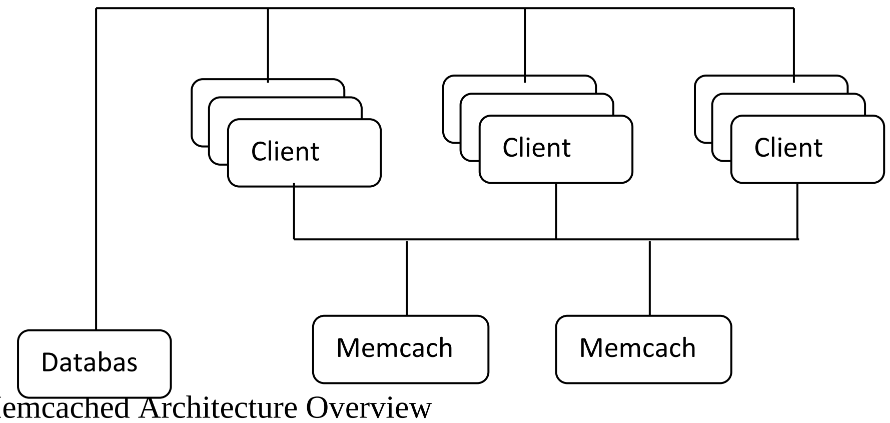


Figure 1 Memcached Architecture Overview

The RAM cache consistently draws data from the underlying store, which in this case is a MySQL database. In the event of a Memcached server failure, data retrieval can be seamlessly executed from the MySQL database to ensure uninterrupted service.

1.2. Memcache

Memcache functions as an in-memory data store for key-value pairs. Accessed through App Engine APIs, it serves as a shared service that is language-agnostic. Implementation varies depending on the programming language and the specific Memcache APIs utilized **Rajesh Nishtala et al. (2013).** This representation can be visualized diagrammatically as follows:
For example it can be implemented in JAVA by using the
followings ways:-

JCache APIs
GAE low level Memcache APIs
Objectify for Datastore

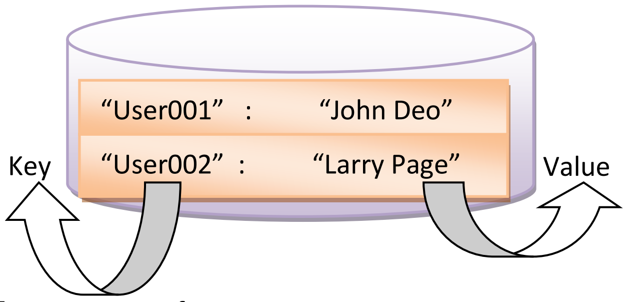


Figure 2 Java Implementation of API

1.3. General Memcache Usage Pattern

Coordinate data read with Datastore Check if Memcache value exists, if it does display/ uses cache value directly, otherwise Fetch the value from Datastore and write the value to Memcache Coordinate data write with Datastore Invalidate the Memcache value for this specific entry or entire Memcache Write the value to Datastore Optionally, Update the Memcache entry.

2. Actual Scenario in the Social Graph like Meta

Meta imposes an exceptionally demanding workload on its data backend. With over a billion active users accessing Meta via desktop browsers or mobile devices, each user encounter hundreds of pieces of information sourced from the social graph. This includes News Feed stories, associated comments, likes, and shares, as well as photos and check-ins from their connections. Due to the extensive customization of user output and the frequent updates to a user's News Feed, pre-generating views for users becomes unfeasible. Consequently, the dataset must be dynamically retrieved and rendered within a few hundred milliseconds to meet user expectations.
The complexity of this challenge is exacerbated by the fact that the dataset is not readily partitionable, and certain items, like photos of celebrities, often experience substantial spikes in request rates. When multiplied by the millions of times per second that this highly personalized dataset must be delivered to users, it creates a continuously evolving workload dominated by reads, posing significant efficiency challenges **Nathan Bronson et al. (2013), Rajesh Nishtala et al. (2013).**

3. Drawback of Using Memcache

The inefficiency of performing operations on lists in Memcached, such as updating the entire list, coupled with the complexity of clients managing cache and the challenge of providing read-after-write consistency, posed significant obstacles to Meta's "move fast" development philosophy. This slowed down the change-debug-release cycle, prompting the development of TAO.
Given Meta's requirement to aggregate and filter hundreds or thousands of items from the social graph on a single page, the necessity for an efficient, highly available, and scalable graph data store becomes evident. This is crucial for serving the dynamic, read-heavy workload while ensuring that each user receives personalized content that undergoes privacy checks.

TAO, despite being a large-scale implementation of the graph approach, continues to rely on MySQL for persistent disk storage. However, it prioritizes ensuring eventual consistency of data across multiple data centers; thereby ensuring users receive up-to-date information. The TAO service operates across a network of server clusters, strategically distributed geographically and structured in a logical tree formation. Distinct clusters are designated for the persistent storage of objects and associations, as well as for caching data in both RAM and Flash memory. This division enables independent scaling of different cluster types and efficient utilization of server hardware resources.

4. TAO data model and API

The TAO data model is elucidated through a straightforward scenario **Nathan Bronson et al. (2013).** In this scenario, Alice performs a check-in at The Red Fort and tags Bob, while Cathy adds a comment to the check-in and David expresses his appreciation by liking it.

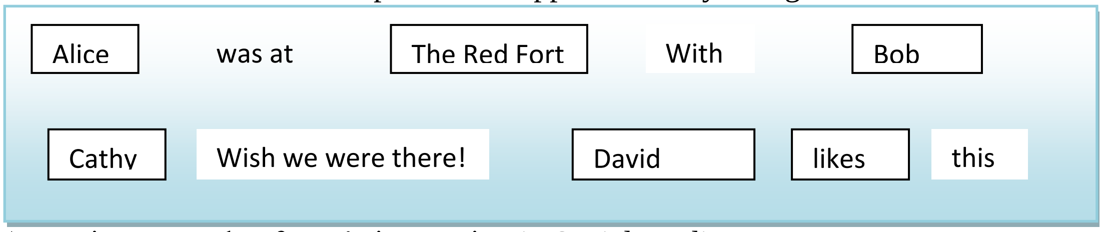


Figure 3 A running example of user's interaction in Social Media

In this straightforward illustration, we observe a sub graph of objects and associations formed within TAO following the culmination of all events. Each data entity, such as a user, check-in, or comment, is depicted as a typed object containing a set of named fields. Relationships between these objects, like "liked by" or "friend of," are represented by typed edges (associations) organized into association lists based on their origin. Multiple associations can link the same pair of objects, provided the types of these associations remain distinct. Collectively, these objects and associations constitute a labeled directed multigraph.

For each association type, TAO allows for specification of an inverse type. When an edge of the direct type is established or removed between objects with unique IDs id1 and id2, TAO automatically generates or deletes a corresponding edge of the inverse type in the opposite direction (from id2 to id1). This feature aims to assist application developers in preserving referential integrity for relationships inherently reciprocal, such as friendships, or where efficient graph traversal in both directions is crucial for performance, as seen in scenarios like "likes" and "liked by."

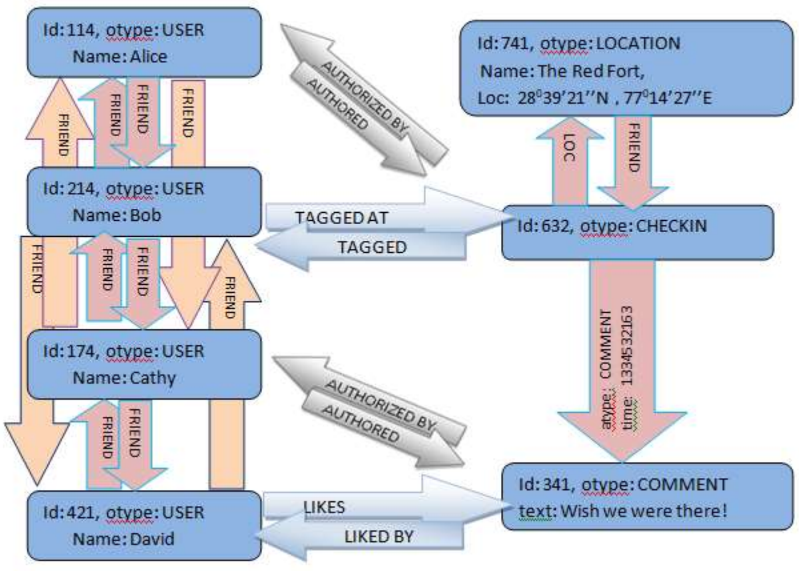


Figure 4 Data Model including Objects and Associations

Associations are managed as individual edges, created and deleted independently. When an association type has a defined inverse type, an inverse edge is automatically generated. To leverage workload locality at creation time, the API mandates that each association includes a dedicated time attribute, typically used to denote its creation time. TAO utilizes this association time value to enhance cache performance and boost hit rates by optimizing the working set.

TAO partitions the dataset into hundreds of thousands of shards, with each shard persistently storing all objects and associations in the same MySQL database. These are cached on identical server sets within each caching cluster. Optionally, individual objects and associations can be allocated to designated shards upon creation. Managing data collocation has proven crucial for optimizing performance by minimizing communication overhead and preventing congestion in high-traffic areas.

T AOBench stands as the inaugural open-source benchmark crafting end-to-end transactional request patterns drawn from a vast social network, bridging the gap of representative workloads in a critical application domain. Through our benchmark, we render Meta's social graph workload accessible to the database community, offering insight into the genuine challenges entailed in accommodating such workloads. TAOBench can faithfully replicate the existing TAO workload while also offering enough flexibility to assess novel scenarios effectively.

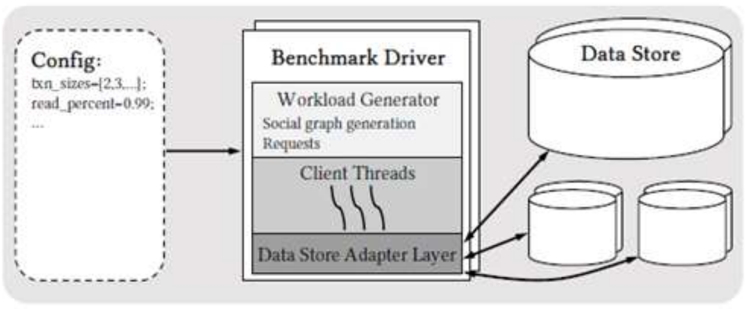
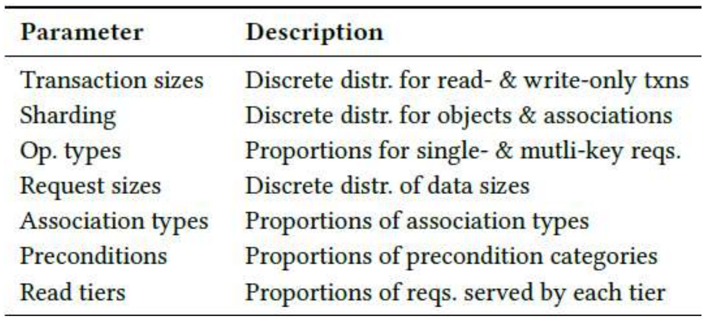


Figure 5 TAOBench architecture (Source:- [16])

Table 1 Parametrize TAO's workload with this set of features (Source:- [16])



Parameter	Description
Transaction sizes	Discrete distr. for read- & write-only txns
Sharding	Discrete distr. for objects & associations
Op. types	Proportions for single- & mutli-key reqs.
Request sizes	Discrete distr. of data sizes
Association types	Proportions of association types
Preconditions	Proportions of precondition categories
Read tiers	Proportions of reqs. served by each tier

Audrey Cheng et al. (2022) categorize these parameters into two sets: those applicable to both OLTP and graph databases, and those specific to TAO. A concise selection of parameters proves ample to thoroughly define the workload of the social network.

Utilizing the workload parameters outlined above, TAOBench utilizes a workload configuration file containing discrete or piecewise linear probability distributions. Additionally, TAOBench incorporates various benchmark parameters (such as duration, target load, and warm-up period) specific to each execution. The benchmark driver, capable of being distributed across multiple machines, utilizes these parameters to generate requests. Each driver spawns multiple client threads, which autonomously execute requests through the data store adapter layer. Each thread monitors throughput and latency, with these metrics being aggregated and reported after each run. Presently, the benchmark generates steady-state workloads, with future iterations aiming to capture temporal variations and periodic trends.

5. Conclusions

TAO, a user-friendly and robust distributed data store created at Meta, has emerged as a cornerstone of the company's infrastructure. Its graph capabilities effectively support the intricate and evolving social demands placed on Meta's systems. TAOBench introduces a novel benchmarking tool that generates workloads mirroring those encountered in real-world social networks. TAOBench also uncovered bugs and highlighted optimization prospects for YugabyteDB. TAOBench's focus centers on Meta's social graph, making these workloads accessible to the broader research community. Notably, TAOBench model can be effortlessly adapted to accommodate other systems. TAOBench aspires to serve as a comprehensive platform for social network evaluation, and invite other social networks to contribute their workloads to this open-source framework.

References

[1] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, et al. "TAO: Meta's Distributed Data Store for the Social Graph". In: Proceedings of the 2013 USENIX Annual Technical Conference. San Jose, California USA, 2013, pp. 49–60
[2] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, et al. "Scaling Memcache at Meta". In: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI). Lombard, IL, 2013, pp. 385–398
[3] 2023. Amazon SimpleDB. http://aws.amazon.com/simpledb/.
[4] 2023. Meta – Company Info. http://newsroom.fb.com.
[5] 2023. Memcached - Project Homepage. http://memcached.org/ .
[6] 2008. P. Saab. Scaling memcached at Meta. http://www.Meta.com/note.php? note_id= 39391378919.
[7] 2022. TAOBench. https://github.com/audreyccheng/taobench
[8] Phillipe Ajoux, Nathan Bronson, Sanjeev Kumar, Wyatt Lloyd, Kaushik Veeraraghavan Challenges to Adopting Stronger Consistency at Scale. University of Southern California, 15th workshop on hot topics in operating systems, May 18-20, 2015
[9] Gregory Chockler, Guy Laden, Ymir Vigfusson, Data Caching as a Cloud Service, LADIS '10: Proceedings of the 4th International Workshop on Large Scale Distributed Systems and MiddlewareJuly 2010 Pages 18–21 https://doi.org/10.1145/1859184.1859190
[10] Pankaj Deep Kaur, Gitanjali Sharma, "Architectures for Scalable Databases in Cloud – And Application Specifications". Procedia Computer Science 58 (2015) 622 – 634
[11] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, Venkateshwaran Venkataramani "Scaling Memcache at Facebook", 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)
[12] Matthew Burke, Audrey Cheng, and Wyatt Lloyd. 2020. Gryf: Unifying Consensus and Shared Registers. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20). 591–617.
[13] Zhichao Cao, Siying Dong, Sagar Vemuri, and David HC Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Meta. In 18th USENIX Conference on File and Storage Technologies (FAST '20). 209–223
[14] Audrey Cheng, Xiao Shi, Lu Pan, Anthony Simpson, Neil Wheaton, Shilpa Lawande, Nathan Bronson, Peter Bailis, Natacha Crooks, and Ion Stoica. 2021. RAMP-TAO: Layering Atomic Transactions on Meta's Online TAO Data Store. Proceedings of the VLDB Endowment 14, 12 (2021), 3014–3027
[15] Xiao Shi, Scott Pruett, Kevin Doherty, Jinyu Han, Dmitri Petrov, Jim Carrig, John Hugg, and Nathan Bronson. 2020. FlightTracker: Consistency across ReadOptimized Online Stores at Meta. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, 407–423.
[16] Audrey Cheng, Xiao Shi, Aaron Kabcenell, Shilpa Lawande, Hamza Qadeer, Jason Chan, Harrison Tin, Ryan Zhao, Peter Bailis, Mahesh Balakrishnan, Nathan Bronson, Natacha Crooks, Ion Stoica. TAOBench: An End-to-End Benchmark for Social Network Workloads. PVLDB, 15(9): 1965-1977, 2022.