Benchmarking Redis vs Memcached: An In-Depth Analysis of In-Memory Data Stores

Vonteri Harshith Reddy

2019CS50450

Introduction

In the realm of high-performance data storage, the choice of an in-memory data store can significantly impact the efficiency and responsiveness of applications. Among the notable contenders in this space, Memcached and Redis have emerged as open-source solutions, each offering distinct features and functionalities tailored to diverse use cases.

Memcached [3], renowned for its simplicity and speed, operates as an in-memory data store focusing exclusively on key-value pairs. Employing a client-server architecture, it excels in facilitating rapid data retrieval and storage. Memcached's multi-threaded design enhances concurrent access, making it particularly well-suited for scenarios where quick access to cached data is paramount. However, Memcached is constrained to storing only bit strings as values and lacks built-in support for data persistence, characteristics that may influence its suitability for certain applications.

In contrast, Redis [2], short for REmote DIctionary Server, extends the key-value paradigm by supporting a broader range of data types, transforming it into a versatile multi-model, including a document-store. Redis also adheres to a client-server architecture but diverges in its single-threaded design. This distinctive approach provides advantages in terms of simplicity and predictability, while Redis excels in handling complex data structures such as strings, hashes, lists, and sets. Furthermore, Redis distinguishes itself by offering on-disk persistence, providing durability and data recovery capabilities.

The choice between Redis and Memcached is a critical decision for organizations seeking optimal in-memory data storage solutions. According to database rankings reported in Table 1 for key-value stores, Redis holds the top position, with Memcached following closely behind in fourth place. In the context of in-premise data stores, where cloud deployment options like Azure and DynamoDB are not applicable, Redis claims the first position, and Memcached secures the second. This positioning underscores the competitive landscape these two platforms occupy in the domain of in-memory data stores.

The significance of this domain becomes evident when considering real-world applications such as gaming leaderboards, shopping carts, and user sessions. In these scenarios, the speed and reliability of in-memory data stores directly impact user experience and system performance. As Redis and Memcached stand out as top contenders in this space, a comprehensive benchmarking analysis becomes imperative to guide decision-makers in selecting the most suitable solution for their specific use cases. This paper endeavors to explore and compare the performance, scalability, and key features of Redis and Memcached, shedding light on their strengths and weaknesses to inform the decision-making process for optimal in-memory data storage.

Table 1: Ranking of various DBMS

Rank (Nov. 2023)	DBMS	Score (Nov. 2023)
1	Redis	160.02
2	Microsoft Azure	83.24
3	Amazon DynamoDB	34.11
4	Memcached	19.8

Experiments

To capture a comprehensive understanding of the performance characteristics of Redis and Memcached, the benchmarking experiments were conducted for the following key metrics:

- **Latency:** Measured to assess the time taken for the system to respond to individual requests, providing insights into the responsiveness of Redis and Memcached.
- **Throughput:** Evaluated to quantify the volume of requests processed per unit of time, shedding light on the overall efficiency of data retrieval and storage.
- **Scalability:** Investigated by varying workloads, thread counts, and operational parameters to understand how well Redis and Memcached can handle increased loads and adapt to changing demands.
- **Fault Tolerance:** Assessed to gauge the resilience of Redis and Memcached in the face of potential system failures or disruptions.
- **Memory Usage:** Evaluated the memory used by Redis and Memcached with varying number of keys to be stored.

These metrics collectively offer a comprehensive view of the comparative performance of Redis and Memcached under different conditions, enabling a nuanced understanding of their capabilities and limitations in real-world scenarios. The subsequent sections will delve into the experimental setup, specific experiments conducted, the measurement methodologies employed, and the results obtained for each of these critical performance metrics. The code to reproduce the experiments is available at: https://github.com/Harshithreddyvonteri/Benchmarking-Redis-and-Memcached. We modified the source code of memtier-benchmark to run the experiments. The modified source code of memtier-benchmark is available at https://github.com/Harshithreddyvonteri/Memtier_benchmark-modified. It is not required to compile this modified source code as the binary is included in the main repo, but the dependencies required by memtier-benchmark needs to be installed.

Experimental Setup

Benchmarking Tool: The benchmarking tool selected for this comparative analysis is Mem-tier [1]. Developed by Redis, Mem-tier stands out as an open-source utility designed to facilitate both comprehensive testing and performance measurement of in-memory data stores. What sets Mem-tier apart is its compatibility with both Memcached and Redis, making it an ideal choice for our comparative study. With a multithreaded architecture, Mem-tier is capable of simulating concurrent access scenarios, providing a realistic assessment of system performance under varying loads.

Mem-tier offers a high degree of flexibility with customizable parameters, allowing fine-tuning of test conditions to replicate real-world scenarios accurately. The tool's statistical analysis capabilities enable the extraction of meaningful insights from the benchmarking results, aiding in the identification of performance bottlenecks and comparative analysis between Redis and Memcached.

The versions of the software under evaluation are pivotal to the accuracy and relevance of the benchmarking results. For this study, we have utilized Redis 7.0.12 and Memcached 1.6.22. All the experiments are ran on a 8-core CPU with 8GB RAM.

Latency Benchmark

The objective of this latency benchmark was to evaluate the time taken by Redis and Memcached to execute Set and Get operations under minimal concurrency. To assess the latency performance of Redis and Memcached, we conducted a series of benchmarks using Mem-tier with the following parameters:

- **# Threads:** 1
- **# Clients per thread:** 1
- **Data size:** 32 bytes

The benchmark results from Fig. 1 reveal that Memcached exhibits slightly better performance than Redis for both Set and Get operations. Analyzing the trends, it becomes apparent that Get operations are slightly faster than Set operations for both Redis and Memcached. This observation implies that the retrieval of data from the in-memory stores is more expedient than the storage of new data. The Redis-Set operation was found to be 5.5% slower than the Memcached-Set operation. Similarly, the Redis-Get operation exhibited a

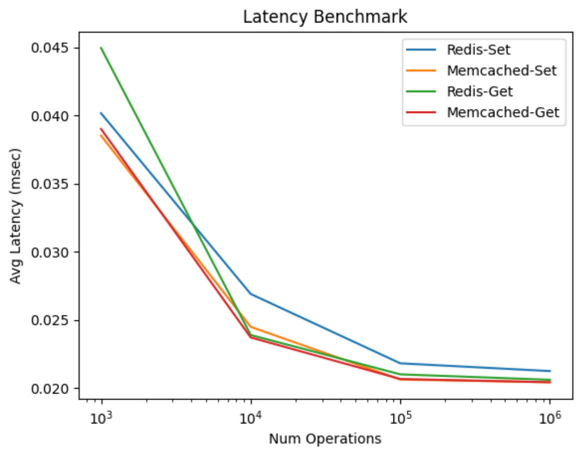


Figure 1: Latency Benchmark

6% delay compared to the Memcached-Get operation. These distinctions provide insights into the relative latencies of the two systems for different operations.

Throughput Benchmark

In this throughput benchmark, we aimed to assess the system-level performance of Redis and Memcached under a more substantial workload using Memtier. The key parameters for this benchmark were as follows:

- **# Threads:** 4
- **# Clients per thread:** 50
- **Data size:** 32 bytes
- **Requests per client:** 1000000

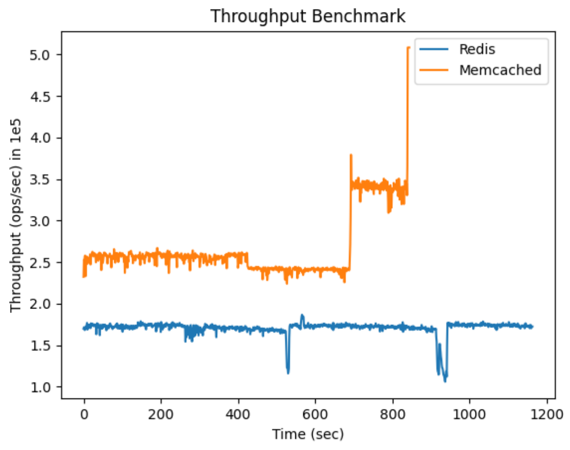


Figure 2: Throughput Benchmark

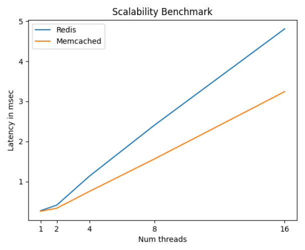
The benchmark results from Fig. 2 show that Memcached outperforms Redis in terms of throughput. This difference in performance can be attributed to Memcached's utilization of multi-threading for request serving, allowing it to efficiently handle a larger number of concurrent requests. Both Redis and Memcached set a threshold to control dictionary expansion. When this threshold is exceeded, a double-size hash bucket is created, and data gradually moves from the existing bucket to the new one. Two distinctive dips, or ”dents,” in the throughput plot for Redis indicate that the server experienced dictionary expansion twice during the benchmark. The impact of dictionary expansion on Redis is evident in the observed throughput degradation, which is approximately 38%. This suggests that the dictionary expansion process in Redis introduces a notable overhead, leading to a decrease in throughput.

In contrast, Memcached handles dictionary expansion more efficiently. Utilizing multi-threading, Memcached dedicates one exclusive expanding thread that is always ready to perform dictionary expansion. This approach ensures that the process does not significantly impact the overall throughput. Consequently, Memcached demonstrates minimal dent in throughput even during dictionary expansion. The fundamental difference in the architecture of Redis (single-threaded) and Memcached (multithreaded) becomes evident in this benchmark. Redis’s single thread must manage both request serving and dictionary expansion, leading to a more pronounced impact on throughput during expansion. Conversely, Memcached’s use of multithreading enables it to separate these responsibilities, contributing to more stable throughput. An interesting observation is the increase in throughput for Memcached, which may be attributed to a reduction in the number of active clients over time. This reduction could be linked to the efficiency of Memcached’s multithreading approach in managing concurrent requests.

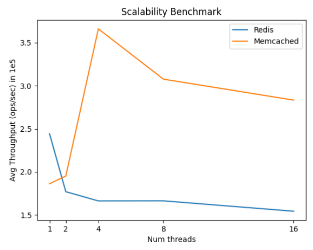
Scalability Benchmark

In the scalability benchmark, we aimed to evaluate how well Redis and Memcached scale in terms of both average latency and throughput under increasing thread counts. The key parameters for this benchmark were as follows:

- **# Clients per thread:** 50
- **Data size:** 32 bytes
- **Requests per client:** 10000
- **Set:Get Ratio** 1:10



(a) Latency



(b) Throughput

Figure 3: Scalability

The benchmark results from Fig. 3a clearly demonstrate that Memcached exhibits superior scalability in average latency as the number of threads increases. This scalability advantage can be attributed to Memcached’s utilization of multiple threads to concurrently serve requests, allowing it to efficiently handle increased loads. The average latency of Redis is observed to be 48.3% higher than that of Memcached. This difference highlights the impact of Redis being single-threaded, as it struggles to manage increased concurrency effectively.

The throughput of Memcached is significantly higher than Redis, even as the number of threads increases as can be seen from Fig. 3b. This result is consistent with the scalability advantage observed in average latency. The constant drop in throughput with an increase in the number of threads for Redis is a characteristic of its single-threaded nature, limiting its ability to efficiently handle concurrent requests. Memcached’s multi-threaded architecture contributes to the initial increase in throughput up to four threads (which is the number of threads memcached uses by default). Beyond this point, there is a gradual decrease, possibly due to the saturation of system resources or contention among threads. Nevertheless, the overall throughput of Memcached remains notably higher than that of Redis across different thread counts.

Fault Tolerance and overhead of Persistance Models

Memcached, by design, lacks native support for on-disk persistence, rendering it non-fault-tolerant in scenarios where data durability is a critical consideration. Memcached relies solely on the in-memory storage paradigm, making it a suitable choice for use cases prioritizing rapid data access over persistent storage. Redis, in contrast, addresses fault tolerance through the implementation of two persistence models:

RDB (Redis Database) Persistence: RDB performs point-in-time snapshots of the dataset at specified intervals, creating a snapshot of the entire dataset. One notable aspect of RDB is its negligible overhead. Redis forks a child process using copy-on-write during background saves, allowing data loading and snapshotting to run concurrently.

AOF (Append Only File) Persistence: AOF persistence logs every write operation received by the server, creating a log that can be replayed at server startup to reconstruct the original dataset. AOF introduces a more significant overhead, particularly in the ”Always” (immediate) syncing mode. This mode ensures that every write operation is immediately synchronized to the AOF file. Redis provides three modes for syncing: Always (Immediate), Every Sec, and No.

- **Always:** Ensures immediate synchronization of every write operation, maximizing data durability.
- **Every Sec:** Synchronizes data to disk every second, balancing durability with performance.
- **No sync:** OS flush the output buffer when it wants. Leaves the syncing to the operating system, providing minimal guarantees about data persistence.

Redis allows for the complete disabling of persistence, a configuration sometimes used in caching scenarios where data durability is of lesser concern.

To understand the overhead of persistence models in Redis, we conducted experiments comparing the performance of three Redis configurations: RDB, AOF-Always (Immediate), and No Persistence. The key parameters for this benchmark were as follows:

- **# Threads:** 4
- **# Clients per thread:** 50
- **Requests per client:** 100000

Table 2: Overhead of Persistence models

Model	Avg. Latency (msec)	Avg. Throughput (ops/sec)
AOF-Always	7.144	28021.47
RDB	1.136	176008.06
No Persistance	1.120	176351.48

From Table 2, we can see that the overhead of RDB persistence is minimal, thanks to the concurrent execution of data loading and snapshotting by leveraging copy-on-write mechanism during background saves. Significant overhead is observed in AOF-Always (Immediate) sync, reflecting the continuous and immediate synchronization of every write operation. The choice of persistence model in Redis involves a trade-off between data durability and performance. RDB offers a more lightweight solution with minimal overhead, while AOF, especially in the ”Always” syncing mode, introduces a more substantial performance impact.

Memory Usage

In the memory usage analysis, the focus is on understanding how Memcached and Redis manage memory when subjected to a specific workload. The experiment was designed with the following parameters:

- **# Threads:** 1
- **# Clients per thread:** 1
- **Operation Type:** Only set

Table 3: Memory Usage comparision

Num Keys	10000	100000	1000000
Redis	0.78	7.46	72.4
Memcached	0.67	6.87	49.6

The memory usage results from Table. 3 demonstrate that Memcached exhibits more efficient memory utilization compared to Redis. Memcached consumes less memory (in MB) even with increase in number of key-value pairs. Memcached’s advantage in memory usage is attributed to its ability to use less metadata to store keys and values. This efficiency becomes particularly noticeable as the size of the database increases.

Limitations and Future Work

The experiments focused on specific workloads, such as single-threaded and low-concurrency scenarios, which may not fully represent the complexity of real-world applications with diverse and dynamic workloads. While the study touched upon fault tolerance, it did not extensively explore recovery scenarios under system failures or assess the impact of persistence models on data recovery times. This can be a potential direction to work-on in future. Availability and Consistency benchmarks are not studied here.

Conclusion

In this comprehensive benchmarking study, we delved into the performance characteristics of two prominent in-memory data stores, Redis and Memcached, across various dimensions such as latency, throughput, scalability, fault tolerance, persistence overhead, and memory usage. While both redis and Memcached exhibit similar latency, Memcached outperformed Redis in throughput, scalability owing to its multithreaded nature. On contarary, Redis offers persistance, replication and can store complex datastuructures, features absent in Memcached. Organizations can leverage the findings of this benchmarking study to make informed decisions based on their specific use cases, balancing factors such as latency, throughput, scalability, fault tolerance, and memory efficiency. The insights gained from these experiments enable decision-makers to optimize resource allocation based on their priorities, whether it be minimizing latency, maximizing throughput, or efficiently managing memory.

References

[1] Memtier-Benchmark, https://github.com/RedisLabs/memtier_benchmark/tree/master
[2] Redis, https://redis.io/
[3] Memcached, https://memcached.org/
[4] Redis Vs Memcached comparison https://scalegrid.io/blog/redis-vs-memcached-2021-comparison/