Comparative Performance Analysis of Redis, Memcached, and Aerospike

Divya Sri Kaile†  
 Computer Science  
 University of Memphis

ABSTRACT

High throughput, low-latency applications are in high demand these days, and it is here that in-memory databases, and key-value stores including Redis, Memcached, and Aerospike, have been underlined as essentials. This paper shows a detailed comparative analysis of these databases using Yahoo Cloud Serving Benchmark (YCSB). It evaluates the performance of the various workloads that include read-heavy, write-heavy, and a balanced workload, on both single-node and clustered setups, measuring throughput, latency, and scalability as key metrics. The results indicate that Redis performs the best in low-latency scenarios due to its in-memory architecture, while Memcached is better-suited for caching workloads that involve lightweight operations and Aerospike prides nothing less than excellent scalability and reliability under high-load conditions. Issues such as memory constraints and replication latencies have been discussed as performance bottlenecks. These findings present insights on how and where to capitalize on each database's strengths and weaknesses, with actionable guidance on how to select a database for any use-case-specific requirements and system architectures.

KEYWORDS: Redis, Memcached, Aerospike, NoSQL, scalability, YCSB, latency, throughput

1. INTRODUCTION

The increase in the in-memory database is so because of the types of existence of DRAM devices getting improved and the ever-growing demands for high-speed access at data retrieval for high-performance applications. Among the premier key-value stores today, Redis, Memcached and Aerospike stand out primarily due to their special features. Redis possesses a versatile data structure for a variety of cases, Memcached offers performance ease in the best and most simplistic method for cache storage, and Aerospike yields the best balance for performance, as well as its scalability. Benchmarking these systems has certain complexities like the requirement of using consistent configurations across different architectures; resolution of philosophical design differences in the database, and bias-free assessment of throughput, latency, and memory consumption.

Benchmarking problems include:

1. Consistent configurations across architectures.
2. Inherent differences due to the philosophy of construction of database systems.
3. Maintaining an unbiased measurement of the evaluation such as throughput, latency, and memory usage.

This discussion is meant to provide the best choices for cases like caching and session management or for some distributed system support.

1. RELATED WORK

Rick Cattell, whose work of 2011 in the ACM SIGMOD Record, has conducted comparative research on various SQL and NoSQL data stores based on data models, consistency mechanisms, durable guarantees, and other dimensions [3].

Some studies analyze the throughput of Voldemort DB, Redis, and other databases for their APM activities for big data applications [5]. Such studies mostly involve scanning operations to measure the efficiency of these databases [2].

Benchmarking efforts focused specifically on Redis and Memcached have also been extensively documented. An article by Alamin examined performance differences by varying key and value sizes. In 2010, Salvatore Sanfilippo, the founder of Redis, released comparative results to throw light on such findings, involving situations with multiple clients and contrasting Memcached (with 2 threads on two cores) with Redis (with two servers on two cores) [3]. While Dormando provided more insights comparing the two databases in different client configurations [6], it demonstrated the varied performance concerning workload and setup.

Many other advances, such as the review of Memcached in a multi-threaded access situation, have extended the investigations to particular use cases and performance issues [4][6].

EVALUATION METHODOLOGY

**3.1 Experimental Setup**

*3.1.1* *Single-Node Setup*

For testing a smaller deployment, it assumes that all database transactions take place on one node.

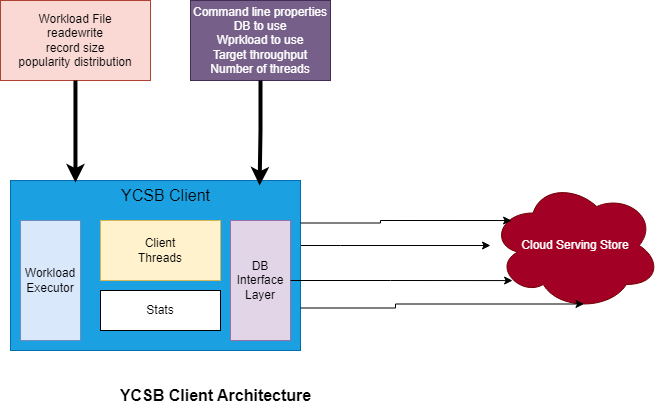
*3.1.2* *Cluster Setup*

This represents a shared database environment in which multiple machines are running instances of the database.

**3.2** **Benchmark Setup**

*3.2.1* *Yahoo! Cloud Serving Benchmark (YCSB)*

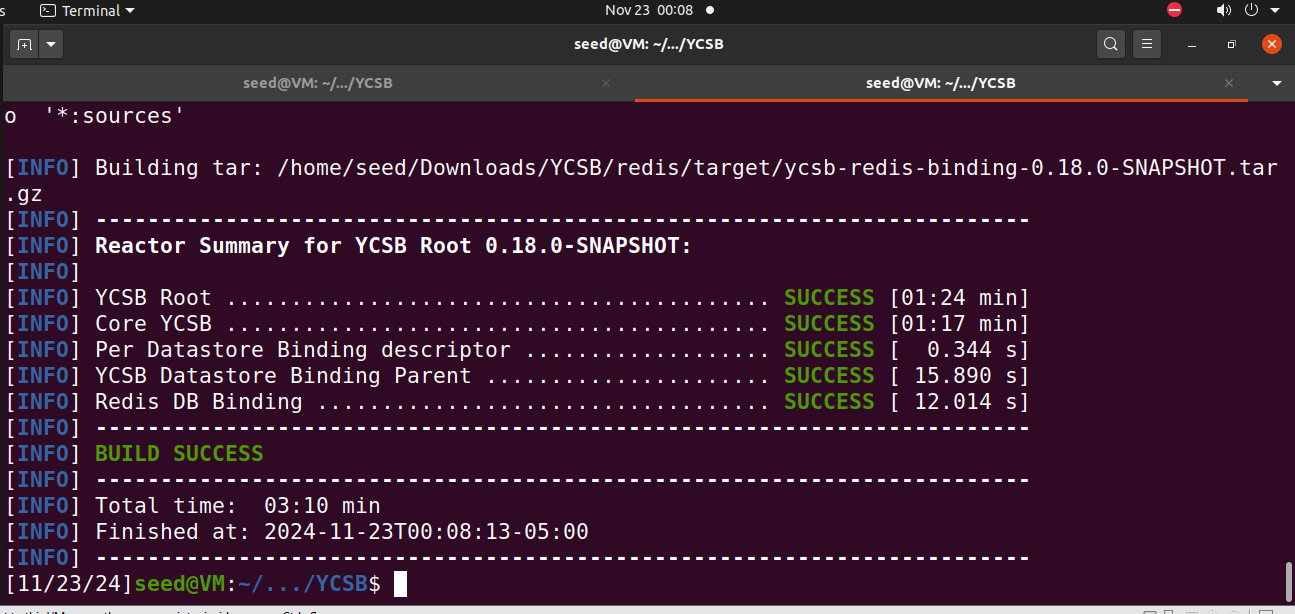
In short, it is an effective platform for simulating as well as benchmarking numerous workloads on databases. Using YCSB, end-users can easily test crucial performance metrics such as throughput, latency, and memory utilization under diverse operational scenarios.

**Figure 1: YCSB Client Architecture**

**3.3** **Databases**

*3.3.1* *Redis*

Designed to facilitate applications with the capabilities of using a lot of different types of data structures, including hashes, sets, and sorted sets, among others.



*3.3.2* *Memcached*

Completely focus on lightweight high-performance caching in multiprogramming architecture thread optimized under read-heavy workloads.

*3.3.3* *Aerospike*

Hybrid in-memory and persistent storage model, very much according to the scalability and strong consistency across the operations.

**3.4** **Workloads**

*3.4.1* *Read Intensive*

Simulate load scenarios whereby the predominant activity involves data retrieval, such as cache usage in web applications [9].

*3.4.2* *Write Intensive*

This stresses the database under largely write-dominant actions, such as transactional use cases like logging or analytics ingestion.

*3.4.3* *Balanced (50-50 Reads and Writes)*

Generic workloads **and** all-purpose databases have read as well as write operations with at least 50% of traffic allocated to either [9].

**3.5** **Evaluation Metrics**

Redis, Memcached, and Aerospike were studied concerning their performance effects with the parameters; throughput, read latency, write latency, and memory footprint, given various workloads with different numbers of concurrent clients.

The measurements for throughput and latency were taken down by the clients to assess responsiveness under load before comparison, while the memory footprint data was collected from the server machines the database was hosted on. In a way, this makes the complete picture of the efficiency and the resource consumption level of each database known, so that an informed choice could be made concerning which particular high-performance application requirements could be addressed by any of them.

1. RESULTS AND OBSERVATIONS

**4.1** **Single-Node Setup**

*4.1.1* *Throughput*

Redis and Aerospike performed similarly, with Aerospike leading in write-heavy scenarios. Memcached excelled in heavy read workloads due to its multi-threaded architecture.

*4.1.2* *Latency*

Memcached exhibited the lowest latency in read-heavy workloads. Redis showed consistent performance but struggled under high concurrency.

*4.1.3* *Memory Efficiency*

Memcached consumed the least memory, followed by Aerospike. Redis exhibited higher memory usage due to its rich data structures.

A graph of different colored bars

Description automatically generated

**4.2** **Cluster Setup**

*4.2.1* *Throughput*

Memcached outperformed Redis and Aerospike in read-heavy and balanced scenarios but lagged in write-heavy tasks. Aerospike demonstrated superior scalability, handling up to 32 concurrent clients efficiently.

*4.2.2* *Latency*

Memcached's latency was consistent in read-heavy workloads but increased significantly under write-heavy conditions. Redis and Aerospike maintained the lowest average latency in most scenarios, achieving approximately 1,034 µs in Workload A. However, latency increased in more complex workloads, such as Workload F, where it showed signs of scalability challenges.

*4.2.3* *Scalability*

Aerospike demonstrated the best scalability, followed by Redis, which leveraged its clustering capabilities effectively.

A graph of different colored bars

Description automatically generated

Figure 2: Throughput comparison across workloads

1. DISCUSSION

***5.1*** ***Redis***

Redis's single-threaded design prioritizes atomicity and simplicity, making it ideal for applications requiring complex data structures [7]. However, due to its limited multi-threading capabilities, it may not be the best choice for highly concurrent environments.

***5.2*** ***Memcached***

Memcached excels in lightweight caching scenarios, offering the highest throughput in read-dominant workloads. Its simplicity, however, limits its performance in write-intensive or complex use cases [1][7].

***5.3*** ***Aerospike***

Aerospike provides a balanced solution for mixed workloads. Its hybrid storage model and focus on scalability make it a strong candidate for distributed applications requiring consistent performance​ [8].

A graph with different colored bars

Description automatically generated

Figure 3: Average latency comparison across workloads.

1. CONCLUSION

Selecting a suitable in-memory key-value store is becoming increasingly important. Reducing request latency, and allowing the access of entire databases in memory, also gives credibility to such systems. This study thoroughly compares the three most popular in-memory key-value stores, Redis, Memcached, and Aerospike, under three workout loads: read-heavy, balanced, and write-heavy, with varying concurrence degrees using YCSB. Single-node and cluster configurations were available for benchmarking.

According to our findings, Memcached, specifically for read-heavy and balanced workloads, can be the best choice for cache because it performs best under this given workload type. Almost everything else is equal between Redis and Aerospike, but since Redis has a heavy memory footprint, Aerospike is deemed a good second option for general usage.

However, for single-client applications where there is a need to deal with complex value types or with bigger key-value sizes, Redis appears to be more favorable over both Memcached and Aerospike.

REFERENCES

[1] Al-Allawee, Ali, Pascal Lorenz, Abdelhafid Abouaissa, and Mosleh M Abualhaj. “A Performance Evaluation of In-Memory Databases Operations in Session Initiation Protocol.” *Network* 3, no. 1 (December 28, 2022): 1–14. <https://doi.org/10.3390/network3010001>.

[2] Alamin, Mohamed. “POWER CONSUMPTION ESTIMATION USING IN-MEMORY DATABASE COMPUTATION,” 2016. <https://acikbilim.yok.gov.tr/bitstream/handle/20.500.12812/638416/yokAcikBilim_10123674.pdf?sequence=-1>.

[3] Stonebraker, Michael, and Rick Cattell. “10 Rules for Scalable Performance in ‘Simple Operation’ Datastores.” *Communications of the ACM* 54, no. 6 (June 2011): 72–80. <https://doi.org/10.1145/1953122.1953144>.

[4] Sangat, Prajwol, Maria Indrawan-Santiago, David Taniar, Beng Oh, and Paul Reichl. “Processing High-Volume Geospatial Data: A Case of Monitoring Heavy Haul Railway Operations.” *Procedia Computer Science* 80 (2016): 2221–25. <https://doi.org/10.1016/j.procs.2016.05.385>.

[5] Neves, Ricardo, and Jorge Bernardino. “Performance and Scalability of Voldemort NoSQL,” June 1, 2015. <https://doi.org/10.1109/cisti.2015.7170484>.

[6] Dormando. “Memcached - a Distributed Memory Object Caching System.” memcached.org. Accessed December 5, 2024. <https://memcached.org/>.

[7] Bonisteel, Steve. “Memcached vs Redis: Choose Your In-Memory Cache.” Kinsta®. Kinsta, June 16, 2023. <https://kinsta.com/blog/memcached-vs-redis/>.

[8] Aerospike Inc. “Aerospike | Real-Time Data Platform - Cloud Managed Database Service.” Aerospike. Accessed December 5, 2024. <https://aerospike.com/>.

[9] RAHUL CHANDEL. “Performance and Scalability Analysis of Redis and Memcached.” dzone.com. DZone, July 2, 2024. <https://dzone.com/articles/performance-and-scalability-analysis-of-redis-memcached>.