ESLE Group 1 Project (Stage 1): YugabyteDB Scalability Analysis and Benchmark

Beatriz Barbosa Esteves Seara Matias[1 [95538]], João Henriques Sereno[1 [99249]], and Simão Santos Silva[1 [99329]]

Instituto Superior Técnico
{beatriz.matias,joaohsereno,simao.santos.silva}@tecnico.ulisboa.pt

Commit: https://gitlab.rnl.tecnico.ulisboa.pt/esle/esle24-g1/-/tree/project-stage-1?ref_type=tags
Demo video link: https://youtu.be/JVMTzLiMVtU

1 Introduction

YugabyteDB is a distributed SQL database designed to provide high performance, scalability, and reliability for modern applications. YugabyteDB advertises to be horizontally scalable, strongly consistent, compliant to ACID properties, and fully SQL compatible. Per the CAP theorem, it focuses in the Consistency - Partition Tolerance guarantees. It supports two interfaces - YSQL (for SQL workloads) and YCQL (for Cassandra-like workloads). In this project, we'll focus on the YSQL interface. The consistency level is tunable in order to achieve higher levels of availability, but we will be using Single-row linearizable transactions and Multi-row snapshot isolated transactions (equivalent to PostgreSQL's repeatable read isolation), the default consistency model offered by YugabyteDB with YSQL interface.

Using our configuration, this system is well-suited for financial applications that require high performance, reliability, and strong consistency, for example. However, this is not the best configuration to choose when the target application has very high throughput demands on writes, for example, a meteorological system gathering data from multiple, geographically distributed sources.

Putting YugabyteDB to analysis and benchmarking will let us gain a deeper understanding of its scalability characteristics, understand how it handles bottlenecks, and compare to what would be expected given our analysis.

2 System Description

2.1 Analysis

Operations in YugabyteDB are split in two logical layers [1]: Yugabyte Query Layer (YQL), that handles SQL and NoSQL queries, and DocDB Layer, that manages distributed storage and replication.

YugabyteDB has two different kinds of instances.

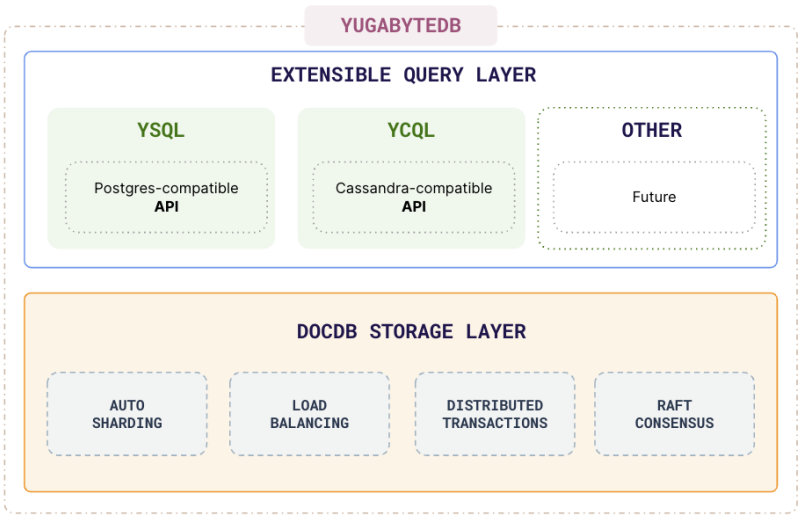


Fig. 1. YugabyteDB operation architecture

YB-Master [2] server acts as a catalog manager and cluster orchestrator. It manages many background tasks. When replicated, they follow a leader-follower distributed architecture. The leader forms a Raft group with its peer to ensure consistency and leader election between replicas. This service is responsible to keep system metadata, records (tables and location of the tablets), users, roles, and so on. It's also responsible for load-balancing and initiation replication of under-replicated data. **This service is not in the critical path of normal I/O operations**.

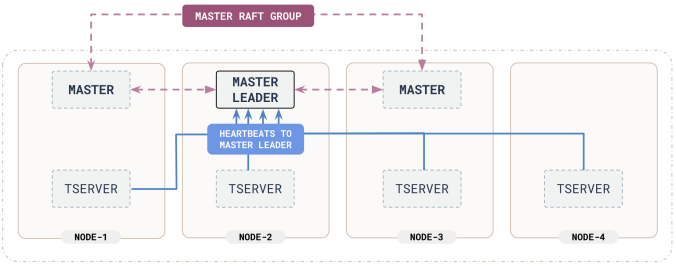


Fig. 2. YB-Master service architecture

YB-TServer [3] splits table data into tablets. Each tablet is composed of one or more tablet peers, depending on the replication factor. These tablets are maintained and managed on each node by the TServer. Each tablet is synchronously replicated using the Raft algorithm.

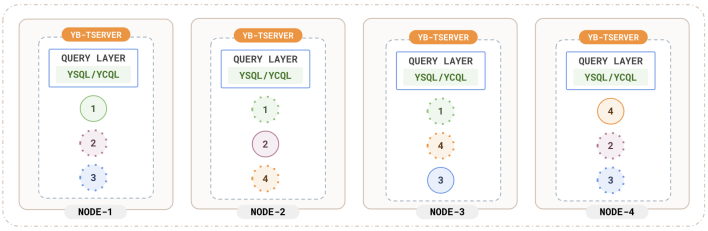


Fig. 3. YB-TServer service architecture

There are various factors that can turn out to be bottlenecks for systems like YugabyteDB. We will benchmark the system to find which factors are the bottleneck, and under which workloads can we that shown.

Specifically:

1. **Network latency**: Since YugabyteDB replicates data using the Raft protocol, which involves communication between nodes, stressing the network. Latency between nodes could become a bottleneck, specially in geographically distributed setups.
2. **Disk I/O**: The DocDB layer manages the underlying storage using a Log-Structured Merge Tree (LSM-tree). While they are efficient for write-heavy workloads, they can introduce performance degradation in read-heavy workloads due to the need to merge data across multiple levels of the tree.

2.2 Workloads

To assess the behavior of the system, we devised a benchmark that contains two separate workloads:

1. **Write-Only Workload**: In this workload, we sent several write requests to the system; these requests consisted of inserts of a key and a value to a key-value table.
2. **Read-Only Workload**: In this workload, we performed several read requests to the system; each request consisted of reading the value associated with a particular key (the keys that are used are in the range of keys that was written in a previous Write-Only workload)

Each of the two workloads was submitted to the system using an increasing number of clients; we started our benchmark by applying each of the two workloads using only 1 thread (each thread represents a client), and after the set of tests for each workload is completed, we would increment the number of threads by one, and repeat the tests. We would stop when we observed that the throughput of the system wasn't increasing (this effectively meant that the system was saturated).

For each workload, and for each number of client threads, two tests were done in total, for 60 seconds each; then the results of these two tests were averaged out, to obtain the effective throughput and latency of that workload, for that number of clients. This was done to minimize the effects of outliers and random fluctuations in the system's performance, which were common because these tests were conducted using our personal laptops.

Here is the highest number of client threads that were required to be used, for each workload:

	Write-Only Workload	**Read-Only Workload**
Max number of Client Threads - 3 Replicas	35	35
Max number of Client Threads - 5 Replicas	45	45

Table 1. Comparison of the maximum number of client threads for each workload

We decided to separate read and write operations completely into two different workloads, due to the known fact that write operations have a higher cost in terms of performance, when compared to read operations in a database. This difference in cost is aggravated in distributed databases, like YugabyteDB, therefore we believed it would be best to separate the two types of operations, as we knew we would have different results in terms of overall throughput and latency. We also did not think it would be useful to include a third workload, that included read and write operations, because the performance profile would be a weighted combination of the other two workloads.

3 Results

All tests were performed with similar available computing power. The computer that executed them has a AMD Ryzen 5 4500U CPU - with up to 4.0GHz and 6 cores, a M.2 2280 SDD - with a maximum of 3350/1350 (MB/s) read/write speeds.

Throughout this report, all mentions of the number of clients are intended to represent concurrent clients, with each client actively interacting with the system during the test period.

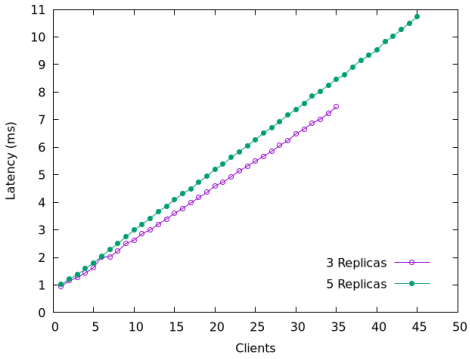
3.1 Read-only workload

In the read-only workload, we observed that the latency increased linearly with the number of clients. As shown in 4(a), when the number of clients increased, the average latency per read operation grew accordingly. This is a typical behavior in systems where an increasing load results in longer wait times for operation completion.

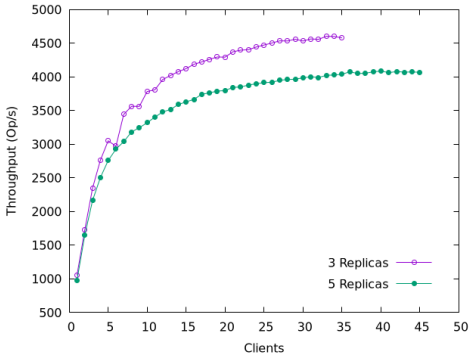
On the other hand, the system's throughput followed a different trend. Initially, as the number of clients increased, the throughput also rose, reaching a peak of approximately 4500 operations per second for three replicas and around 4000 operations per second for five replicas, both at approximately 15 clients. After this point, as depicted in 4(b), throughput plateaued, since increasing the number of clients did not result in any significant increase in throughput, indicating that the system had reached its saturation point.

This behavior aligns with predictions from the Universal Scalability Model (USM), as shown in 4(c). The model highlights how the system's capacity, which factors in performance along with the serial fraction and crosstalk factor, reaches its maximum at roughly 4500 operations for three replicas and 4000 for five replicas. Both configurations hit their peak at around 15 parallel workers. Beyond this, the system's capacity plateaus, indicating that adding more resources yields no payoff and no significant performance gains. Theoretically, the lines should match since increasing resources while holding other factors constant should result in similar behavior. However, they don't match, likely due to the increasing sharding factor with more replicas, which demands higher levels of network communication, as well as variations in our testing environment. Despite efforts to maintain consistent conditions, some variability in the tests was inevitable.

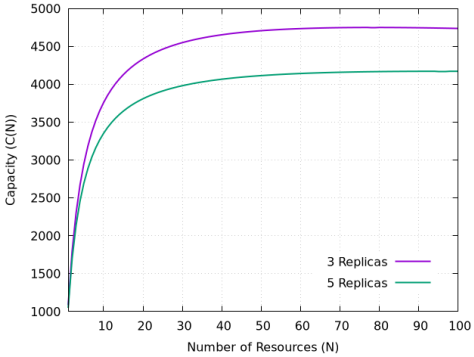
4(d), shows the relationship between latency and throughput. As expected, there is a trade-off: as throughput increases, latency begins to rise, reflecting the cost of handling additional clients. Once the system reaches its throughput peak, the latency increases at a higher rate without corresponding gains in throughput. This indicates that the system has reached its saturation point, beyond which it can no longer efficiently process additional load.



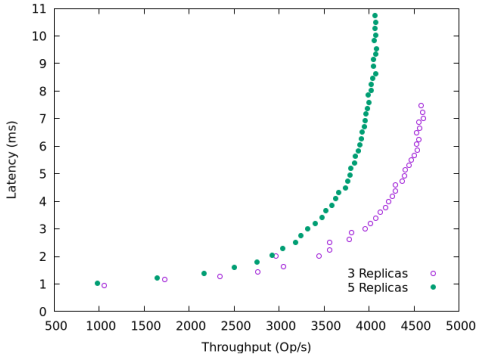
(a) Average read latency for N clients



(b) Average read throughput for N clients



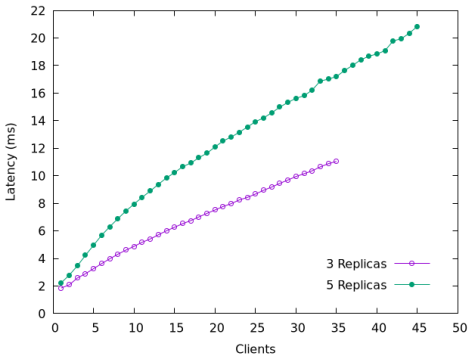
(c) Universal Scalability Model of reads



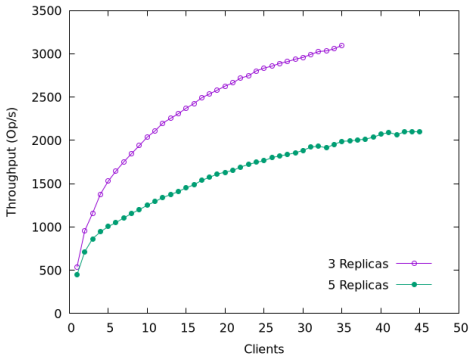
(d) Latency-Throughput relation of reads

Fig. 4. Read-only workload results

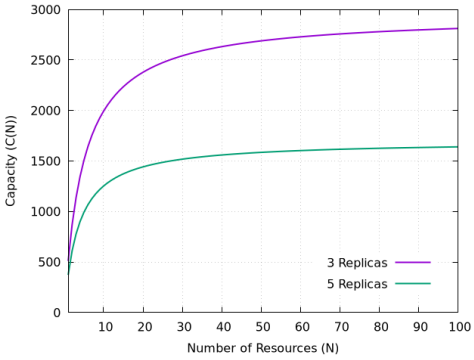
3.2 Write-only workload



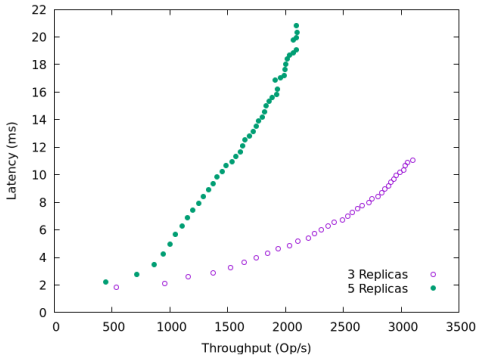
(a) Average write latency for N clients



(b) Average write throughput for N clients



(c) Universal Scalability Model of writes



(d) Latency-Throughput relation of writes

Fig. 5. Write-only workload results

The write-only workload followed similar patterns to the read-only workload, with some notable differences. As shown in 5(a), the latency per write operation grew linearly as well, however, it grew significantly faster compared to the read-only workload. This can be attributed to the higher resource cost of write operations in a distributed environment like YugabyteDB, where data must be replicated across nodes to maintain consistency.

In terms of throughput, as seen in 5(b), the system managed to increase throughput until it plateaued at approximately 2000 operations per second, for five replicas, and 3000 operations per second, for three replicas, indicating the system had reached its maximum capacity for handling write operations, much like with reads, but at a lower throughput level due to the additional complexity and resource demand of writes.

When applying the USM to writes, the calculator returned a negative crosstalk factor, which is impossible. Despite running various tests on different machines, we were unable to determine the cause without inspecting the source code. As a result, for the graph shown

in 5(c), we set the crosstalk factor to zero. This adjustment was necessary because accurately measuring the impact of communication in a local deployment is challenging. After this change, 5(c) exhibits a similar trend to the USM for reads. The system reaches its maximum capacity at around 2500 operations for three replicas and 1500 for five replicas, with both configurations peaking at roughly 15 parallel workers. Beyond this point, capacity plateaus. The same issue of the lines not matching, as seen with reads, also occurs with writes.

In 5(d), the latency-throughput curve for writes mirrors that of the read workload but with a steeper incline. However, we did not observe a vertical asymptote, which suggests that we had not fully saturated the system.

4 Conclusion

In this project, we evaluated the scalability characteristics of YugabyteDB by analyzing both read and write operations under varying levels of client load. Through benchmarking, we found that YugabyteDB demonstrates solid scalability for read-only workloads, with throughput increasing up to a saturation point of around 4500 operations per second for three replicas and 4000 for five replicas. However, write operations exhibited lower throughput and a faster increase in latency due to the higher resource demands of replication and consistency mechanisms in a distributed environment.

These findings illustrate YugabyteDB's strengths and limitations. While it is well-suited for applications requiring strong consistency and moderate throughput, especially in read-heavy scenarios, its performance can degrade under high write loads or geographically distributed setups where network latency becomes a bottleneck.

4.1 Future Work

In the future, we aim to develop more comprehensive benchmarks that include larger read scans and write targets, to better evaluate YugabyteDB's performance under diverse workloads. Additionally, testing in production-like deployment environments will provide more realistic insights, helping us understand the system's behavior under real-world conditions.