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

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

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

Commit: gitlab.rnl.tecnico.ulisboa.pt/esle/esle24-g1/-/tree/final-delivery?ref_type=tags
Demo video link: https://youtu.be/0sLP7ZoDMyQ

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.

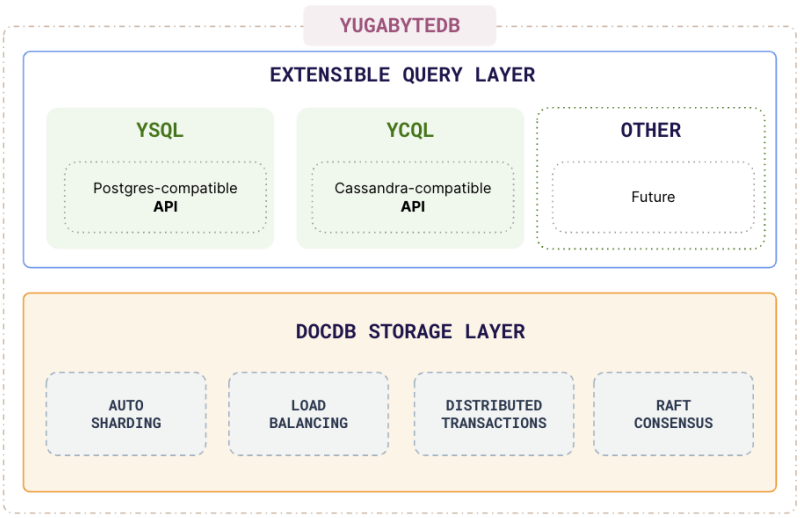

Fig. 1. YugabyteDB operation architecture

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**.

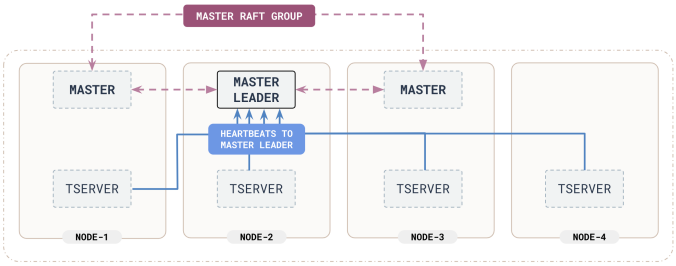

Fig. 2. YB-Master service architecture

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.

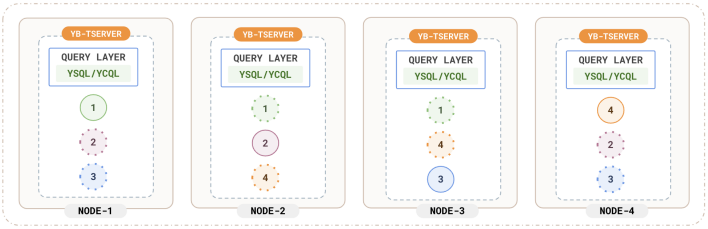

Fig. 3. YB-TServer service architecture

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.

3 Experimental Design

3.1 Factors and Levels

Before designing the experimental design, we needed to determine which factors, and levels of those factors, would be used in the experiments; this step is of the utmost importance, as certain factors can be either impractical to test or irrelevant to our experiment's objective.

Given the predefined structure of six factors, each with two levels, we conducted our experiments with the following factors and levels:

Factor	Level 1	Level 2
Number of Tservers	3	5
CPU/RAM by VM	Instance type: `n1-standard-2` vCPUs: 2 Memory: 7.5 GB	Instance type: `n2-standard-4` vCPUs: 4 Memory: 16 GB
Sharding Replication	False	True
Consistency Level	Read Committed	Snapshot
Workload Type	Read-Only	Write-Only
Operation Size	1 row per Operation	100 rows per Operation

Table 1. Factors and Levels Used in Experiments

The following is a detailed description of each factor and the rationale behind its inclusion in our experiments:

1. **Number of TServers**: The number of TServers (worker servers) deployed

2. **CPU/RAM by VM**: Here we vary the type of VMs that are used in our system's deployment; in this case, we use a more powerful VM type on Level 2. It is important to note that each server of our deployment is deployed on an individual VM (whether it is a Master or TServer).
   It would have been interesting to use an even more powerful VM in Level 2 (with 8 CPUs), as YugabyteDB relies a lot on the CPU, but unfortunately, it was not possible to do so, due to the imposed quotas on Google Cloud.

3. **Sharding Replication**: If there is or is not replication of shards. In order to control the impact that consistency algorithms have on the database, we may set this factor to true or false if we want to see all TServers with the same data (replication factor is the count of TServers) or each TServer with its own set (shard) of data with no replication.

4. **Consistency Level**: This factor determines the data consistency level. Level 1 uses "Read Committed" ensuring only committed changes are visible. Level 2, "Snapshot" provides higher consistency by isolating each transaction's view of the data.The choice of these two levels allows us to assess the trade-offs between consistency and performance under different workloads.

5. **Workload Type**: The workload used to test the system. These are equal to the workload types of Phase 1 of the project: read-only and write-only workloads.

6. **Operations Size**: The number of rows affected by a single operation in a test (the value used in the previous stage of the project was equal to Level 1, i.e. 1 row per Operation). In a read operation, this would mean that, for level 2, we would be fetching 100 rows with a single SQL `SELECT` statement.

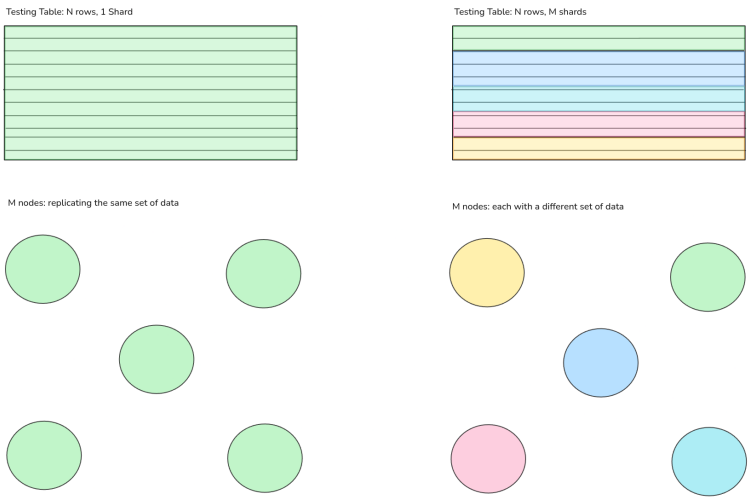

Fig. 4. Shard Replication Illustration

3.2 Fractional Factorial Design

With the factors and levels now clearly defined, we proceed to present the selected experimental design.

The number of experiments needed to conduct a **factorial design** is calculated by the expression $\boldsymbol{n^k r}$, where:

- $\boldsymbol{n}$: Number of levels
- $\boldsymbol{k}$: Number of factors
- $\boldsymbol{r}$: Number of replications (number of repeated tests done, to minimize experimental error)

In our scenario, we have $\boldsymbol{n} = 2$ and $\boldsymbol{k} = 6$, which results in a total of $\mathbf{64 \cdot r}$ individual experiments required. Even at $\boldsymbol{r} = 1$, this number remains excessively high.

To reduce the number of experiments while maintaining the integrity and reliability of our results, we opted for a **fractional factorial design**. In a fractional factorial design, some factors are **confounded**, meaning that the effects of certain factors are not estimated independently. This allows us to reduce the total number of experiments by only examining a subset of the possible combinations of factor levels.

The number of experiments of fractional factorial design is calculated by the following expression: $\boldsymbol{n^{k-p} \cdot r}$, where $\boldsymbol{p}$ is the number of factors that are confounded. We decided on a value of $\boldsymbol{p = 2}$, resulting in the number of experiments being equal to $\mathbf{16 \cdot r}$. We chose $\boldsymbol{p = 2}$ because it still maintains the integrity of the experiments when compared to a full factorial design, while $\mathbf{16 \cdot r}$ experiments are much more manageable.

Our choices for cofounding factors are depicted in the following table (we explain our choices after the table):

Operation Size	Workload Type	Nr. of TServers	Consistency Level	Sharding Replication	CPU/RAM p/ VM
A	B	C	D	E = B × C	F = A × B

Table 2. Factors and Corresponding Symbols

The table illustrates our approach to confounding interactions within the experimental design. Specifically, we have opted to confound the interaction between Workload Type and Nr. of TServers with Sharding Replication ($\mathbf{E} = \mathbf{B} \times \mathbf{C}$). Furthermore, we have confounded the interaction between Operation Size and Workload Type with CPU/RAM per VM ($\mathbf{F} = \mathbf{A} \times \mathbf{B}$).

The key to achieving a good fractional factorial design is to choose the right factors to be confounded; more specifically, try to combine two (or more) factors with a **low interaction effect**, and confound it with a factor with a **significant effect**.

In the case of $\mathbf{E} = \mathbf{B} \times \mathbf{C}$, we chose to join **Workload Type** with **Nr. of TServers** because conceptually the interaction effect should be low. Our thought process was the following: if we do a read-only test and a write-only test (regardless of the number of TServers), the performance of the read-only test will be **higher** than the write-only test by **x** amount. If we now increase the number of TServers, and repeat the two tests, the performance of both tests will increase, but there will be no difference to the conclusions drawn from the tests done with fewer TServers (the read-only test will **still perform better by x amount**). Therefore, we can conclude that the interaction effect from these two factors is low. On the opposite end, factor E (Sharding Replication) has a **high effect** on the response variables (in our case, throughput and latency), as the way the tables and shards are set up can significantly impact performance. With this, we reached our goal: **confound a significant factor** (Sharding Replication) with an **insignificant one** (interaction effect from Workload Type and Number of TServers).

In the case of $\mathbf{F} = \mathbf{A} \times \mathbf{B}$, we chose to join **Operation Size** with **Workload Type**, because the difference in cost between querying 1 row (read) and inserting 1 row (write) is proportional to the difference in cost between querying 100 rows and inserting 100 rows. Then, the factor CPU/RAM p/VM obviously has a great impact in the overall throughput and latency of the system.

3.3 Sign Table

With our fractional factorial design explained, we can now proceed to present our **sign table**; we chose -1 for representing Level 1, and +1 for Level 2:

Runs	Operation Size (A)	Workload Type (B)	Nr. of TServers (C)	Consistency Level (D)	Sharding Replication (E = B × C)	CPU/RAM p/ VM (F = A × B)
1	-1	-1	-1	-1	1	1
2	-1	-1	-1	1	1	1
3	-1	-1	1	-1	-1	1
4	-1	-1	1	1	-1	1
5	-1	1	-1	-1	-1	-1
6	-1	1	-1	1	-1	-1
7	-1	1	1	-1	1	-1
8	-1	1	1	1	1	-1
9	1	-1	-1	-1	1	-1
10	1	-1	-1	1	1	-1
11	1	-1	1	-1	-1	-1
12	1	-1	1	1	-1	-1
13	1	1	-1	-1	-1	1
14	1	1	-1	1	-1	1
15	1	1	1	-1	1	1
16	1	1	1	1	1	1

Table 3. Sign Table

And this is the sign table with the corresponding levels replaced:

Runs	Operation Size (A)	Workload Type (B)	Nr. of TServers (C)	Consistency Level (D)	Sharding Replication (E)	CPU/RAM p/ VM (F)
1	1 row p/ Operation	Read-Only	3	Snapshot	True	`n1-standard-2`
2	1 row p/ Operation	Read-Only	3	Read Committed	True	`n1-standard-2`
3	1 row p/ Operation	Read-Only	5	Snapshot	False	`n1-standard-2`
4	1 row p/ Operation	Read-Only	5	Read Committed	False	`n1-standard-2`
5	1 row p/ Operation	Write-Only	3	Snapshot	False	`n2-standard-4`
6	1 row p/ Operation	Write-Only	3	Read Committed	False	`n2-standard-4`
7	1 row p/ Operation	Write-Only	5	Snapshot	True	`n2-standard-4`
8	1 row p/ Operation	Write-Only	5	Read Committed	True	`n2-standard-4`
9	100 rows p/ Operation	Read-Only	3	Snapshot	True	`n2-standard-4`
10	100 rows p/ Operation	Read-Only	3	Read Committed	True	`n2-standard-4`
11	100 rows p/ Operation	Read-Only	5	Snapshot	False	`n2-standard-4`
12	100 rows p/ Operation	Read-Only	5	Read Committed	False	`n2-standard-4`
13	100 rows p/ Operation	Write-Only	3	Snapshot	False	`n1-standard-2`
14	100 rows p/ Operation	Write-Only	3	Read Committed	False	`n1-standard-2`
15	100 rows p/ Operation	Write-Only	5	Snapshot	True	`n1-standard-2`
16	100 rows p/ Operation	Write-Only	5	Read Committed	True	`n1-standard-2`

Table 4. Sign Table with replaced level's values

4 Results

Runs	Operation Size (A)	Workload Type (B)	Nr. of TServers (C)	Consistency Level (D)	Sharding Replication (E)	CPU/RAM p/ VM (F)	Throughput ceiling
1	1 row p/ Operation	Read-Only	3	Snapshot	True	`n1-standard-2`	3561
2	1 row p/ Operation	Read-Only	3	Read Committed	True	`n1-standard-2`	3568
3	1 row p/ Operation	Read-Only	5	Snapshot	False	`n1-standard-2`	3479
4	1 row p/ Operation	Read-Only	5	Read Committed	False	`n1-standard-2`	3489
5	1 row p/ Operation	Write-Only	3	Snapshot	False	`n2-standard-4`	2846
6	1 row p/ Operation	Write-Only	3	Read Committed	False	`n2-standard-4`	2789
7	1 row p/ Operation	Write-Only	5	Snapshot	True	`n2-standard-4`	2847
8	1 row p/ Operation	Write-Only	5	Read Committed	True	`n2-standard-4`	2753
9	100 rows p/ Operation	Read-Only	3	Snapshot	True	`n2-standard-4`	2985
10	100 rows p/ Operation	Read-Only	3	Read Committed	True	`n2-standard-4`	2922
11	100 rows p/ Operation	Read-Only	5	Snapshot	False	`n2-standard-4`	2931
12	100 rows p/ Operation	Read-Only	5	Read Committed	False	`n2-standard-4`	2953
13	100 rows p/ Operation	Write-Only	3	Snapshot	False	`n1-standard-2`	2931
14	100 rows p/ Operation	Write-Only	3	Read Committed	False	`n1-standard-2`	1033
15	100 rows p/ Operation	Write-Only	5	Snapshot	True	`n1-standard-2`	994
16	100 rows p/ Operation	Write-Only	5	Read Committed	True	`n1-standard-2`	878

Table 5. Sign Table with results

We define the throughput ceiling to be the maximum average throughput that was observed in each run with any number of concurrent clients. We focused our attention in this rather than the latency since the throughput is a more critical property in YugabyteDB.

With this defined metric, we constructed table 5, and we are able to discover what are the main effects of this metric, and what are the interaction effects between factors.

4.1 Main effects

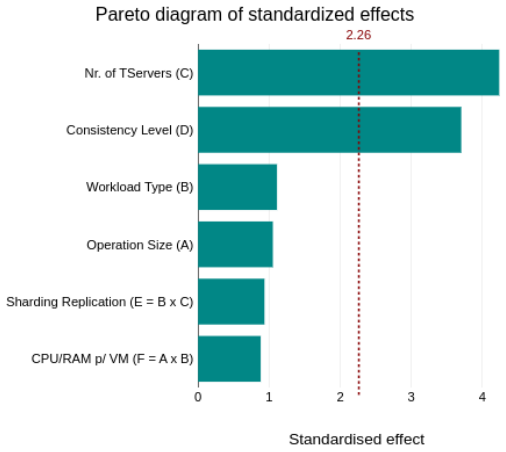

Fig. 5. Pareto diagram of standardized effects

By examining the Pareto diagram in Figure 5, we observe that the most influential factors affecting the throughput ceiling are the number of TServers and the selected Consistency Level.

This observation aligns with expectations, as distributed algorithms designed to ensure strong consistency are highly network-intensive, which can significantly degrade system performance. Both Factors C and D directly impact these properties.

The number of TServers directly contributes to increased network demand. As the number of nodes required for data replication grows, so does the need for communication among these nodes.

Selecting a Snapshot consistency level introduces additional network constraints, as every operation—whether a read or a write—requires confirmation by a quorum of nodes, increasing network overhead. In contrast, the Read Committed consistency level has less stringent requirements, which helps to alleviate network load. Both levels cause great impact on YugabyteDB's performance

4.2 Interaction effects

One interesting aspect to analyze too is the interaction effect between factors, in particular, the ones we used to confound with the single factor (Workload Type $\times$ Operation Size and Workload Type $\times$ Nr. of TServers). This analysis will allow us to assess if we made a good choice or not in which factors to confound.

Workload Type × Operation Size

The interaction effect between these two factors can be observed in the following graph:

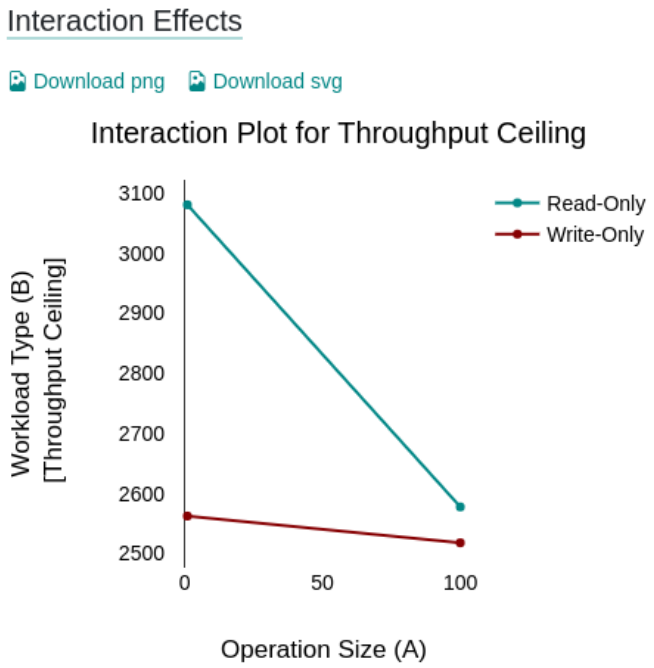

Fig. 6. Workload Type and Operation Size Interaction Effect

To evaluate whether there is a lot of interaction effect or not, we need to look at the **slope** of both lines: if the slope is similar, there is a low interaction effect; if it is not similar, then probably there is a higher interaction effect.

In this case, there is a significant slope difference. Read-only throughput suffers much more with a higher operation size than write-only throughput; this may be due to how YugabyteDB processes a SQL `SELECT` statement with many keys to read, versus a `INSERT` with many keys.

This tells us that maybe Workload Type × Operation Size is not the ideal factor to be confounded with another factor.

Workload Type × Nr. of TServers

The interaction effect between these two factors can be observed in the following graph:

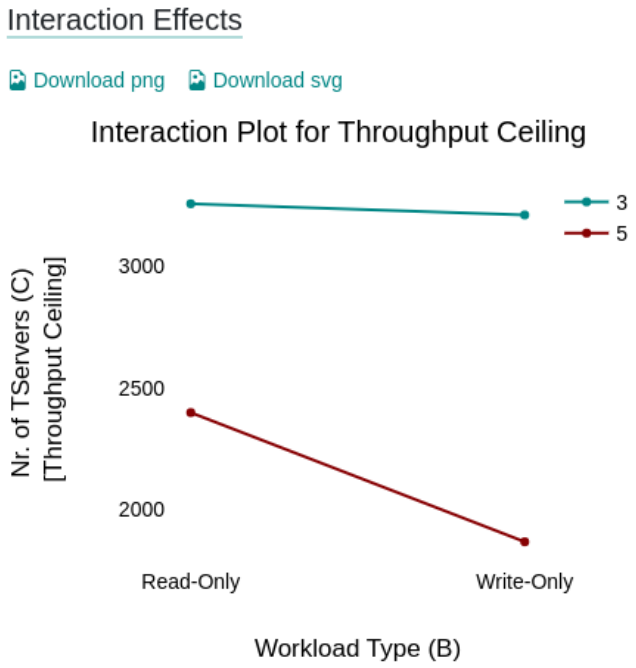

Fig. 7. Workload Type and Nr. of TServers Interaction Effect

In this case, we can see that the difference between the slopes of the two lines is much smaller when compared to the previous interaction effect. Whilst there is some difference in the slopes, we can confidently say that this was a better choice of factors to be used in the confounding of factors.

Another interesting fact that is clearly visible in this graph is that the throughput with 3 TServers is higher than with 5 TServers.

5 Conclusion

In conclusion, our performance and scalability analysis of YugabyteDB reveals that key factors influencing system behavior are the number of TServers and the transaction isolation level. The analysis demonstrates that both Read Committed and Snapshot isolation levels impact transaction processing significantly, with higher isolation levels generally incurring more overhead. Throughput performance, surprisingly, improved with fewer nodes, highlighting YugabyteDB's strong consistency model and the associated cost of replicating data across multiple nodes. This effect suggests that scaling out with additional nodes may not yield linear performance improvements in strongly consistent workloads due to increased coordination requirements.

On a small note: CPU and RAM were found to be the lowest influencing factors in this analysis, but Google Cloud quotas limited our ability to test configurations with sufficiently contrasting levels.

Our analysis further shows that the interaction between workload type and operation size had a noticeable impact on system behavior, suggesting that we should've chosen other pair to confound in order to get a more accurate experience. However, the interaction effect between workload type and the number of TServers was low, validating our choice to control for them without substantial impact on the overall conclusions. Overall, this study illustrates the complexities of scaling YugabyteDB, with implications for optimizing node configurations, isolation levels, and workload distributions in strongly consistent, distributed environments.