Experimental Study: Measuring the Performance of Distributed Databases Yugabyte vs Postgres with Citus

1st Duha Jarrar
Faculty of Engineering and Technology
Birzeit University
Palestine, Jenin
duha.jarrar@gmail.com

2st Hiba Naseer
Faculty of Engineering and Technology
Birzeit University
Palestine, Ramallah
hibanasser1996@gmail.com

3st Hana Al-Baidaq
Faculty of Engineering and Technology
Birzeit University
Palestine, Ramallah
hanaalbidaq@gmail.com

4st Nahil Edkadek
Faculty of Engineering and Technology
Birzeit University
Palestine, Jerusalem
nahil.edkadek95@gmail.com

***Abstract*—In light of the increasing demand for scalable database systems characterized by high throughput and low response times, it has become essential to analyze the impact of the number of nodes and the number of requests sent by users to the system or application on response time and productivity. This study aims to evaluate the performance of distributed databases, focusing on PostgreSQL with the Citus extension and YugabyteDB, by measuring throughput and latency. The study employed empirical methods, including benchmarking tests using the pgbench tool to simulate workload processing across a variety of nodes. The experimental results clearly demonstrate Citus's superiority over YugabyteDB in terms of response time and transaction volume across different workloads. Additionally, statistical findings confirm that Citus outperformed YugabyteDB in 15 out of 16 scenarios, reflecting its high efficiency in handling horizontal scaling and an increasing number of concurrent users. These results suggest that choosing Citus is the optimal option for systems requiring exceptional performance in distributed work environments, thus enhancing companies' ability to achieve rapid and reliable responses.**

***Index Terms*—Distributed Database, Citus, PostgreSQL, YugabyteDB, OLTP, Letancy, Throughput.**

I. Introduction

The exponential growth of data-intensive workloads (spanning real-time analytics, continuous behavioral authentication, edge computing, and financial trading) has driven the need for database systems with horizontal scalability, ultra-low query latency, and high throughput. Native distributed databases, architected to span interconnected nodes with advanced replication protocols such as Raft and Paxos, promise ACID semantics alongside high availability and partition tolerance, yet they must grapple with the CAP theorem's trade-offs and inherent network challenges [1]. Moreover, modern NewSQL solutions introduce additional layers (transactions acceleration proxies [2] and monitoring subsystems [3]) that can further complicate latency optimization.

Among the leading open-source distributed SQL platforms, PostgreSQL with Citus extension and YugabyteDB offer contrasting data distribution and query processing approaches. Citus transforms PostgreSQL into a shared-nothing cluster by sharding tables across worker nodes under a coordinator, enabling parallel query execution and manual data co-location to minimize cross-shard communication [4], [5]. While this architecture retains PostgreSQL's rich indexing and query optimizer, coordinator-worker RPCs can introduce millisecond-scale overhead when shards are not perfectly co-located or when distributed keys are suboptimal [6]. In contrast, YugabyteDB is designed as a cloud-native distributed SQL database, layering a PostgreSQL-compatible query engine (YSQL) on top of a Raft-based storage layer (DocDB). Earlier versions suffered from significant RPC latency between YSQL and TServer processes (reported to be up to 300 times slower than single-node latencies by roughly 20% [7], [8].

Latency in these systems emerges from diverse sources: physical network delays (propagation, queuing, processing), storage I/O overheads (SSD garbage collection, SAN fabric latencies), CPU and query planner costs (parsing, optimization, context switching), and distributed coordination mechanisms (Raft consensus, two-phase commits) [9], [10]. Empirical benchmarks (in contexts as varied as continuous authentication workloads [11] and OLAP performance on OpenStack clusters [12]) demonstrate the need to isolate these factors when measuring per-query latency and its scalability trade-offs.

Building on this foundation, our study provides a comparative performance analysis of PostgreSQL with Citus and YugabyteDB, with latency and throughput as central metrics. We perform point lookups, range scans, and mixed read/write transactions on deployments ranging from four to eight nodes.

Distributed relational databases are increasingly pivotal in handling large-scale, latency-sensitive applications. This study presents an empirical evaluation of YugabyteDB, a natively designed distributed SQL database, compared to PostgreSQL enhanced with Citus and extension-based approaches to distributed database capabilities.

This research investigates the performance of both systems under varying numbers of transactions and worker nodes, focusing on two critical metrics: latency and throughput. conditions, focusing on two critical metrics: Latency and throughput. Specifically, it evaluates the following.

1) The impact of scaling the number of nodes on database performance.
2) The effect of increasing concurrent user transactions on system responsiveness and efficiency.

This study contributes:

1) A comparative analysis of natively distributed versus extension-based distributed architectures.
2) Insights into the trade-offs in throughput and latency as the deployment scales.
3) Real-world workload simulations using standard benchmarking tools to inform practitioners choosing between these platforms.

The remainder of this report is organized as follows. The research will begin with an introduction that clarifies the specific research problem and its main objectives. Section II reviews the background on Citus and YugabyteDB architectures. Section III reviews related work studies. Section IV includes the methodology, which consists of the hypothesis, experimental design, experimental results, data analysis, and discussion of the findings. Finally, the research will conclude with a synthesis of the findings in Section V.

A. *Motivations*

Low latency is critical for real-time analytics workloads, such as fraud detection, recommendation engines, and operational dashboards, where sub-100 MS query responses are needed to derive timely insights and maintain system relevance [1]. In continuous authentication systems, which process tens of thousands of behavioral-biometric events per second, even small spikes in data-access latency can open security gaps and degrade the user's experience [11]. User-facing applications, from online gaming to AR/AV, similarly demand end-to-end latencies below perceptual thresholds (often more than 20 ms) to prevent motion sickness and ensure interactivity.

Furthermore, OLAP scenarios on private cloud clusters reveal that small-scale latency inefficiencies increase dramatically as data volumes and node counts grow, underscoring the importance of optimized distributed query coordination [12].

B. *Research Questions*

This study aims to answer the following questions:

1) *What is the impact of increasing the number of nodes (cluster size) on throughput in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?*
2) *What is the impact of increasing the number of nodes (cluster size) on latency in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?*
3) *How does raising the number of concurrent client loads (user requests) affect throughput in PostgreSQL with Citus and YugabyteDB?*
4) *How does raising the number of concurrent client loads (user requests) affect latency in PostgreSQL with Citus and YugabyteDB?*

II. Background

This section summarizes the architectures of the two distributed SQL systems under study, PostgreSQL augmented with the Citus extension and YugabyteDB, to highlight their contrasting approaches to data partitioning, query routing, and transaction coordination.

A. *PostgreSQL/Citus Architecture*

Citus is an open-source extension to PostgreSQL that transforms it into a distributed database. Instead of forking PostgreSQL, Citus adds distributed functionality—like sharding, distributed transactions, and parallel query execution—while maintaining full compatibility with the core PostgreSQL features and ecosystem [4] [5]. It enables PostgreSQL to scale horizontally across multiple nodes, making it suitable for data-intensive and high-concurrency applications [4] [5]. It transforms a standard PostgreSQL server into a shared-nothing cluster [2] [4]. By partitioning tables into logical "shards", each shard is hosted on a worker node [4] [5], while a coordinator node maintains metadata, parses incoming SQL, plans distributed queries, and dispatches sub-queries to workers [3]. Developers choose a distribution key to co-locate related rows on the same shard [4] [5]. When joins span shards, the coordinator incurs additional RPC round-trips and network serialization overhead [2]. Citus has different types of tables based on its documentation, only three are commonly known. The five types are distributed tables, reference tables, Local Tables, Co-located tables, Append-Distributed Tables.

B. *YugabyteDB Architecture*

YugabyteDB is an open-source, distributed SQL database engineered for mission-critical applications that demand horizontal scalability, high resilience, and global data consistency [6]. It seamlessly blends traditional relational database features—like PostgreSQL-level ACID transactions and SQL compatibility—with the cloud-era advantages of data distribution, replication, and multi-region deployment [6].
YugabyteDB is a distributed database that combines the principles of distributed systems, where many machines work together. YugabyteDB is designed to process and manage data across several nodes (servers) to ensure resilience, consistency, high availability, scalability, and fault tolerance. It's built around a Raft-based storage engine (DoCDB) with a PostgreSQL-compatible query layer (YSQL) on top [5] [6]. Data is automatically sharded by primary-key range into tablet replicas that form Raft groups, writes execute via synchronous

Raft consensus, while reads can be served by followers or the leader, depending on consistency requirements [6]. YugabyteDB auto-balances tablets across nodes, manages split and merge operations transparently, and supports distributed two-phase commits for multi-shard transactions, at the cost of consensus-induced tail-latency spikes when leaders move or networks lag [5].

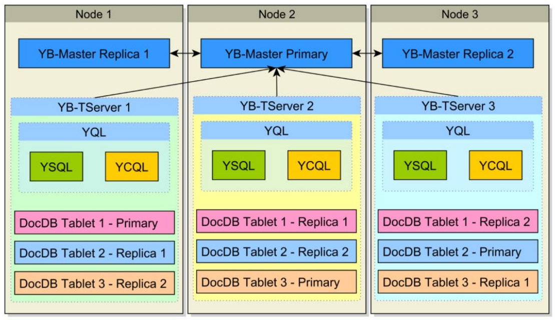
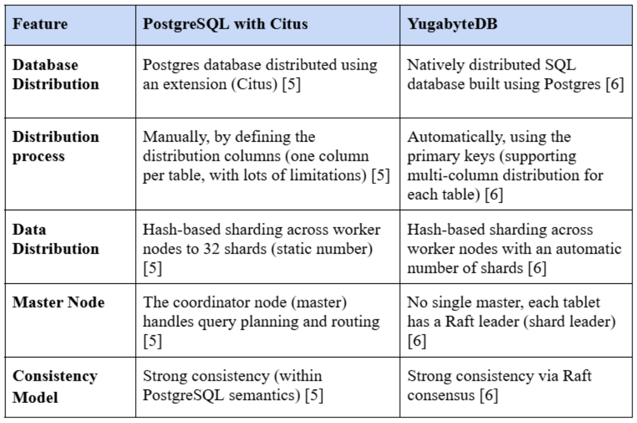


Fig. 1. Architecture Of YugabyteDB [7]

The key differences between the two distribution databases, shown in the following table:



Feature	PostgreSQL with Citus	YugabyteDB
Database Distribution	Postgres database distributed using an extension (Citus) [5]	Natively distributed SQL database built using Postgres [6]
Distribution process	Manually, by defining the distribution columns (one column per table, with lots of limitations) [5]	Automatically, using the primary keys (supporting multi-column distribution for each table) [6]
Data Distribution	Hash-based sharding across worker nodes to 32 shards (static number) [5]	Hash-based sharding across worker nodes with an automatic number of shards [6]
Master Node	The coordinator node (master) handles query planning and routing [5]	No single master, each tablet has a Raft leader (shard leader) [6]
Consistency Model	Strong consistency (within PostgreSQL semantics) [5]	Strong consistency via Raft consensus [6]

Fig. 2. Table: Key differences between Yugabyte and Postgres with Citus

C. OLTP Online Transaction Processing

(OLTP) refers to database systems optimized for managing large numbers of short, atomic transactions in real time. OLTP systems prioritize low-latency, high-concurrency access, typically using a row-oriented storage model, indexes and sophisticated concurrency control to maximize throughput and ensure ACID guarantees for operations like inserts, updates, and deletes [18].

III. Related Work

In 2023, Watanabe et al. [13] conducted an experimental study aimed at improving transaction processing in NewSQL databases using a central proxy. They presented this as a new approach compared to previous studies to address the issues of weak data consistency and transaction features in distributed key-value stores (D-KVS) compared to RDBMS. They found that NewSQL databases achieve consistency similar to RDBMS, but their issue lies in the lower accuracy of timestamps used in transaction processing, negatively affecting performance. Their study focused on comparing the performance between the proposed system using the central agent and YugabyteDB. They used Sysbench as a performance measurement tool, specifically for transaction performance, and employed SQLite3 as an internal database used in the proxy. The results indicated that the proposed model achieved significant improvements in transaction performance, with YugabyteDB's performance being lower compared to the proposed model, achieving improvements of up to 8.25 times under certain workloads, which indicates a substantial enhancement.

While in 2021, Akash Budholia [14] improved performance by developing a monitoring system that tracks every node in NewSQL database systems. This was necessary due to the challenges in monitoring NewSQL databases resulting from increased data volume and system expansion, which complicates effective monitoring of each node. The study focused on evaluating the performance of YugabyteDB, Cassandra, and InfluxDB when storing monitoring data. Budholia used Grafana and Prometheus tools to monitor the nodes, which track metrics such as load, response time, and request counts. The results showed that Cassandra outperformed in write performance when handling data streams, thanks to its multi-master architecture. YugabyteDB achieved excellent read performance due to its strong consistency guarantees. However, InfluxDB faced challenges in horizontal scaling due to inter-node communication issues.

As for 2023, Da Silva and Lima [15] compared the performance of distributed databases Cassandra, HBase, and Citus/PostgreSQL by measuring throughput, latency, performance under different workloads, the impact of the replication factor, and data availability and partition tolerance. This was due to the increasing volume of data resulting from technological advancements and the prevalence of concepts like IoT, often referred to as Big Data, which in turn led to challenges in managing traditional relational databases, causing performance issues. New storage technologies, such as NoSQL databases, have emerged as a more suitable option for big data scenarios. However, scaling storage and processing requires additional infrastructure in terms of devices and data centers. According to their studies, the proposal was to adopt clusters of single-board computers (SBCs) as an alternative to traditional data centers. They used the Prometheus tool in this study, similar to Akash Budholia's research, to monitor resource usage and performance in the cluster, but also added the YCSB tool (Yahoo Cloud Serving Benchmark) as a means to evaluate database performance in terms of throughput and response time. Ultimately, Cassandra achieved better results with the consistency levels defined by the client, while both HBase and Citus saw their performance decrease as the number of

replicas increased, according to their study.
And in 2024, Fotache et al. [16] measured the performance of PostgreSQL and MongoDB databases through a reliability metric. This was because NoSQL technologies tend to be less reliable when managing critical data, particularly when transforming TPC-H data from a relational model to a JSON model. Their goal was to determine the impact of the database type and the number of nodes on the success of query execution. They compared the performance of OLAP queries between relational PostgreSQL databases and non-relational MongoDB in private cloud environments to see how this affects database query performance under different scenarios regarding data volume and node count. They examined the resulting dataset using exploratory data analysis. Their study found that Citus/PostgreSQL achieved greater success in query execution compared to MongoDB. The success rate of queries remained stable across different data distribution scenarios in Citus/PostgreSQL, while the results in MongoDB were variable. Additionally, Chi-Square test results showed a strong relationship between the type of database server and the success of query execution.

IV. Methodology

We evaluate PostgreSQL+ Citus and yugabyteDB under OLTP workloads by systematically varying horizontal scale (3, 5, 7 worker nodes) and concurrency (1,000, 10,000, 100,000) simultaneous clients, and measuring transaction throughput (TPS) and average latency "ms". Our experiments run on a single 12-core Intel i7-1255U host with 16 GB RAM; each database node and the pgbench driver execute in isolated Docker containers orchestrated via Docker Compose to guarantee identical environments and prevent resource interference.

- **Selection of Systems and Data Layout**
  1) PostgreSQL+ Citus: manual sharding of order_trans by (customer_id) with product as reference table, routed through a central coordinator.
  2) YugabyteDB: native, Raft0based automatic sharding on primary keys with consensus-driven replication.
- **Workload and benchmarking:** A custom pgbench script issues prepared-statement transactions that SELECT product details and INSERT new orders. We isolate the effects of each independent variable by:
  1) Fixing client load (10,000) and scaling nodes (3→7) to isolate horizontal-scaling impact.
  2) Fixing cluster size (3-nodes) and varying concurrency (1,000→100,000) to measure load sensitivity.
  3) Running both systems under identical parameters for direct comparisons.
- **Automation and reproducibility**
  1) All steps: cluster deployment, data loading (pgbench –i), workload execution, and metrics collection, are automated via Python and shell scripts. This ensures consistent, repeatable trials and eliminates manual configuration drift. Source code and Docker manifests are available at [17].

A. *Hypothesis*

Hypothesis of RQ1

- Null Hypothesis (H0): There is no significant difference in throughput between PostgreSQL with Citus and YugabyteDB as the number of worker nodes increases under OLTP workloads.
- Alternative Hypothesis (H1): There is a significant difference in throughput between PostgreSQL with Citus and YugabyteDB as the number of worker nodes increases under OLTP workloads.

Hypothesis of RQ2

- Null Hypothesis (H0): There is no significant difference in latency between PostgreSQL with Citus and YugabyteDB as the number of worker nodes increases under OLTP workloads.
- Alternative Hypothesis (H1): There is a significant difference in latency between PostgreSQL with Citus and YugabyteDB as the number of worker nodes increases under OLTP workloads.

Hypothesis of RQ3

- Null Hypothesis (H0): There is no significant difference in throughput between PostgreSQL with Citus and YugabyteDB as the number of concurrent user requests increases in OLTP scenarios.
- Alternative Hypothesis (H1): There is a significant difference in throughput between PostgreSQL with Citus and YugabyteDB as the number of concurrent user requests increases in OLTP scenarios.

Hypothesis of RQ4

- Null Hypothesis (H0): There is no significant difference in latency between PostgreSQL with Citus with YugabyteDB as the number of concurrent user requests increases in OLTP scenarios.
- Alternative Hypothesis (H1): There is a significant difference in latency between PostgreSQL with Citus with YugabyteDB as the number of concurrent user requests increases in OLTP scenarios.

B. *Experimental results*

The findings in Table I (Transaction Throughput) and Table II (Transaction Latency) were generated directly through the execution of our automated benchmarking tests. We used the pgbench utility to simulate OLTP workloads on PostgreSQL deployed with the Citus extension and YugabyteDB for various worker node counts (3, 5, and 7) and various transaction counts (from 20,000 to 100,000). The reported measurements for throughput (in transactions per second) and latency (in milliseconds) are the mean performance recorded under the various configurations and therefore the empirical basis for our study and comparison.

C. *Data Analysis*

In this section, we present the measurements collected from the experiments, and then the analysis conducted by the team to address the research questions and evaluate the

TABLE I
Throughput (Transactions Per Second) result for Citus and YugabyteDB Under Varying Worker Counts and Transaction Volumes

Trans. #	**Citus**			**Yugabyte**		
	3W	**5W**	**7W**	**3W**	**5W**	**7W**
20k	1123.8	1044.2	958.1	721.6	469.2	445.3
40k	1292.4	1165.8	1077.5	673.3	520.2	480.0
60k	1226.4	1182.8	1094.7	692.4	552.5	486.1
80k	1262.9	1206.7	1028.9	688.2	554.0	478.2
100k	1310.1	1203.8	1016.0	675.7	534.3	480.6

TABLE II
Transaction Latency (ms) for Citus and YugabyteDB Under Varying Worker Counts and Transaction Volumes

Trans. #	**Citus**			**Yugabyte**		
	3W	**5W**	**7W**	**3W**	**5W**	**7W**
20k	177.973	191.536	208.749	277.179	426.26	449.185
40k	154.755	171.556	185.609	297.052	384.478	416.673
60k	163.075	169.094	182.704	288.83	361.979	411.436
80k	158.372	165.742	194.375	290.632	361.025	418.26
100k	152.661	166.143	196.853	296.009	374.299	416.152

associated hypotheses. The analysis objective is to investigate the relation between the independent variables (number of worker nodes and transaction workloads) and the dependent performance metrics [throughout, latency]. To evaluate the validity of the hypotheses, the p-values were calculated, a t-test was also conducted using the R software to assess the statistical significance of the observed difference between PostgreSQL + Citus and YugabyteDB. Additionally,Excel was used to generate charts and a visual representation of the data, supporting comparative interpretation and facilitating insight into system performance under varying workload and scaling conditions.

1) Throughput trends: To understand how throughput scales under increasing concurrent client workload and the number of hosts which is essential for optimising horizontal scalability and resource allocation in distributed database systems. This analysis will address Research Questions 3 and 1:

Research Question 3: How does Raising the number of concurrent client load (user request) affect throughput in PostgreSQL+Citus and YugabyteDB?

A series of controlled benchmarking experiments using 3,5,and 7 worker nodes were conducted, throughput measurements were recorded across increasing transaction volumes (concurrent client load). The analysis was visually reinforced using comparative bar-chart and statistically validated using Welch two sample t-test.

According to Figure 3, our 3- worker nodes benchmarks reveal a clear performance advantage for PostgreSQL+Citus over YugabyteDB across all transaction volumes. A t-Test (t=16.193, df=4.54, p= 3.41 *10-5) confirms this gap as highly significant, with an average throughput difference of 553.95TPS. Consequently, we reject the null hypothesis and conclude that Citus scales OLTP workloads more efficiently than YugabyteDB under identical Horizontal scaling.

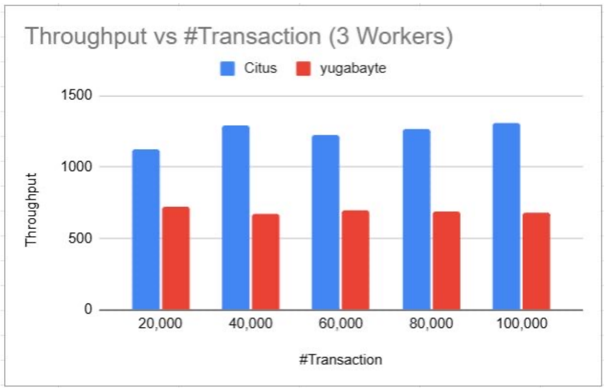


Fig. 3. Throughput (tps) for Citus and YugabyteDB with 3 Worker Nodes Across Increasing Transaction Volumes

According to Figure 4, our 5-worker node benchmarks results confirm Citus's clear throughput advantage over YugabyteDB at every load. The t-test (p=1.49*10 -6) strongly rejects the null hypothesis.

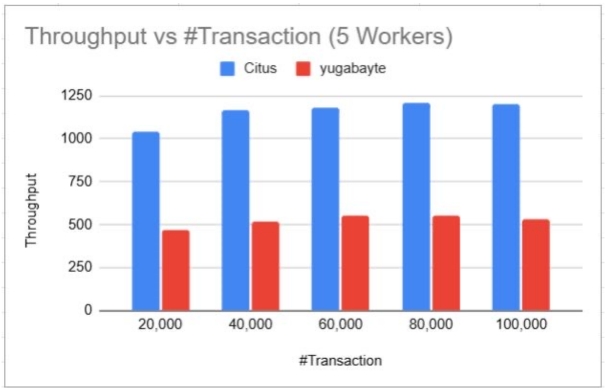


Fig. 4. Throughput (tps) for Citus and YugabyteDB with 5 Worker Nodes Across Increasing Transaction Volumes

According to Figure 5, in the benchmarks for the 7 worker nodes, Citus shows that it maintains a clear performance lead in all loads. The t-test (t = 20.727, p = 7.38 * 10-6) strongly rejects the null hypothesis.

As a result in all cases 3, 5, 7 worker nodes the Null hypothesis rejected and the Alternative hypothesis accepted, proving that PostgreSQL+ Citus consistency demonstrated superior throughput performance compared to YugabyteDB across all transaction volumes and scaling configurations.

Research Question 1: What is the impact of increasing the number of nodes (cluster size) on throughput in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?

A series of controlled benchmarking experiments using 20,000, 40,000, 60,000, 80,000, and 100,000 transactions were conducted, throughput measurements were recorded across increasing worker nodes number. The analysis was visually reinforced using comparative bar-chart and statistically validated using Welch two sample t-test.

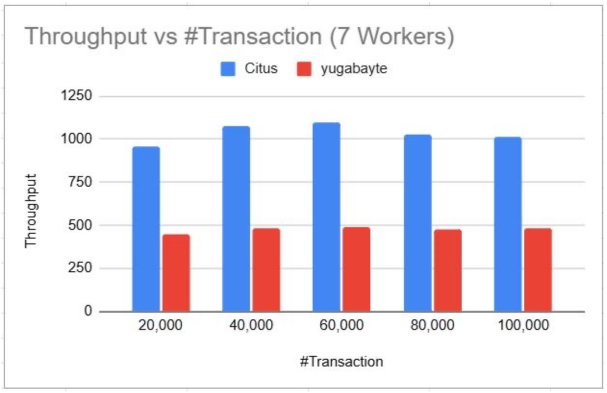


Fig. 5. Throughput (tps) for Citus and YugabyteDB with 7 Worker Nodes Across Increasing Transaction Volumes

According to Figure 6, our 20,000 transaction benchmarks confirm Citus's clear throughput advantage over YugabyteDB at every scale. A t-test (t=4.9142, df=3.08, p=0.01914) confirms the 563.34 TPS mean gap at 20,000 transactions as statistically significant (p¡0.05), reinforcing Citus's clear throughput advantage, and rejecting the null hypothesis.

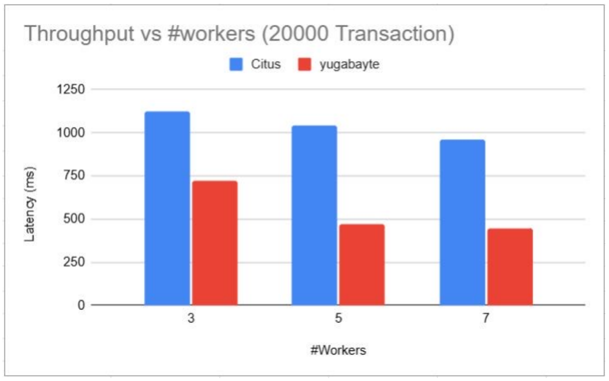


Fig. 6. Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 20,000 Transactions

According to Figure 7, our 40,000 transaction benchmarks confirm Citus's clear throughput advantage over YugabyteDB at every scale. A t-test (t= 7.2384, df= 3.987, p= 0.001957) reveals a 557.82 TPS mean gap (p¡0.05), confirming Citus significantly higher throughput at 40,000 transactions. Which led to rejecting the Null hypothesis.

According to Figure 8, our 60,000 transaction benchmarks confirm Citus's clear throughput advantage over YugabyteDB at every scale. A t-test (t=8.1948, p= 0.002073) reject null hypothesis (p¡0.05), confirming a statistically significant approximately 579 TPS mean advantage for Citus.

According to Figure 9, our 80,000 transaction benchmarks confirm Citus's clear throughput advantage over YugabyteDB at every scale. Testing the null hypothesis against alternative hypothesis with t- test (t=6.3404, p=0.00375) leads to rejecting the null hypothesis.

According to Figure 10, our 100,000 transaction bench-

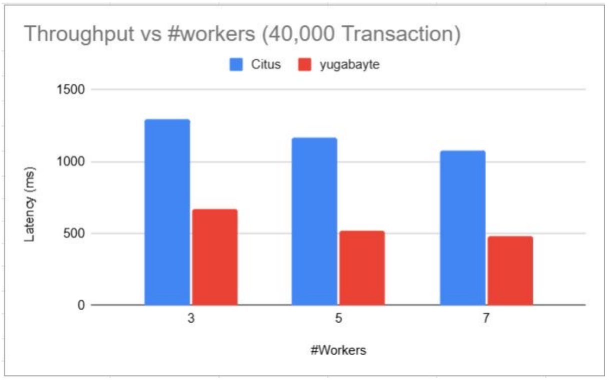


Fig. 7. Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 40,000 Transactions

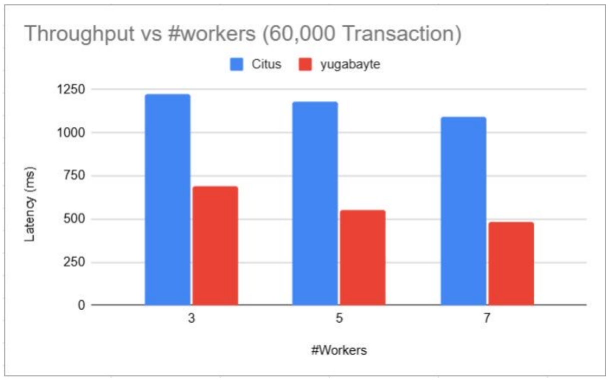


Fig. 8. Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 60,000 Transactions

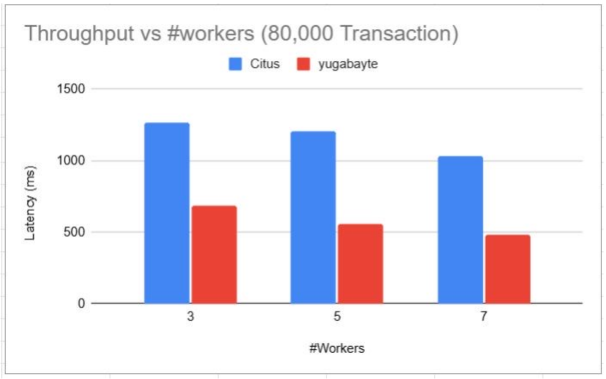


Fig. 9. Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 80,000 Transactions

marks confirm Citus's clear throughput advantage over YugabyteDB at every scale. To the hypothesis testing null(no throughput difference) against the alternative (significant difference) using the t-test (t=5.9059, df=3.51, p=0.01062) this mean rejecting the null hypothesis. As a result in all cases (20000, 40000, 60000, 80000, 100000) transactions the Null hypothesis rejected and the Alternative hypothesis accepted, proving that PostgreSQL+ Citus consistency demonstrated superior throughput performance compared to YugabyteDB across all workers.

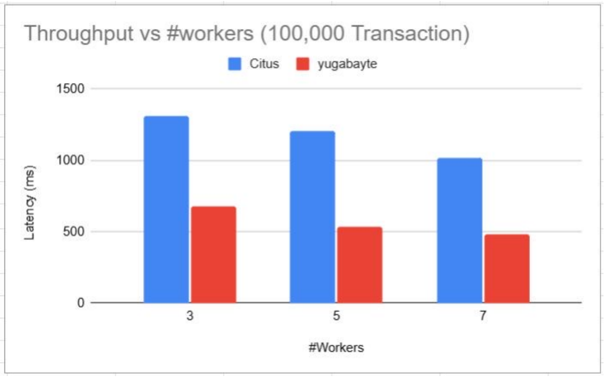


Fig. 10. Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 100,000 Transactions

2) Latency Behaviour: To understand how latency scales under increasing concurrent client workload and the number of hosts which is essential for optimising horizontal scalability and resource allocation in distributed database systems. This analysis will address Research Questions 4 and 2 :

Research Question 4: How does raising the number of concurrent client load (user requests) affect latency in PostgreSQL with Citus and YugabyteDB?

According to the barchart Figure 11, shows that Citus delivers consistently lower latencies than YugabyteDB at every load level. We confirmed from this with a t-test which returned t= -22.307 and p=3.35 *10-8 (p¡0.05). Accordingly we reject the null hypothesis of equal mean latencies and accept the alternative.

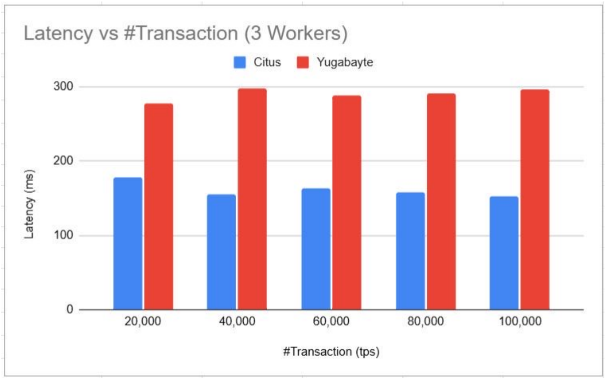


Fig. 11. latency (ms) for Citus and YugabyteDB with 3 Worker Nodes Across Increasing Transaction Volumes

According to the Figure 12, shows that Citus delivers lower latencies than YugabyteDB at every load level. We confirmed from this with a t-test which returned t= -16.194 and p = 1.009* 10-5 (p¡0.05). Accordingly we reject the null hypothesis of equal mean latencies and accept the alternative.

As illustrated in Figure 13, Citus delivers significantly lower latency than YugabyteDB at every load. Testing the null hypothesis against the alternative using the t-test (t=-27.836, df=7.023, p=1.896*10-8) yields a 95% confidence interval of -248.10 to -209.27 ms and decisively rejects the null hypothesis, confirming that Citus's faster response times under heavy OLTP workloads are statistically significant. As a result in all cases 3, 5, 7 worker nodes the Null hypothesis rejected and the Alternative hypothesis accepted, proving that PostgreSQL+ Citus consistency demonstrated superior throughput performance compared to YugabyteDB across all transaction volumes and scaling configurations.

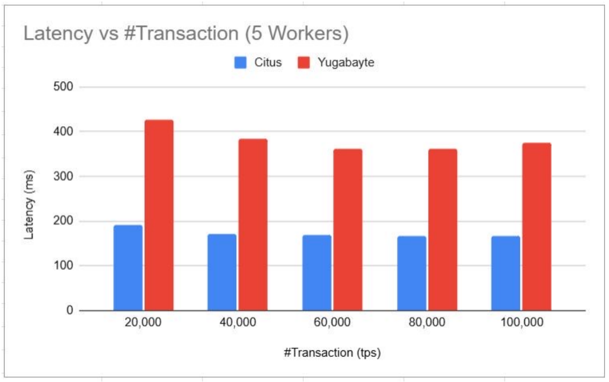


Fig. 12. latency (ms) for Citus and YugabyteDB with 5 Worker Nodes Across Increasing Transaction Volumes

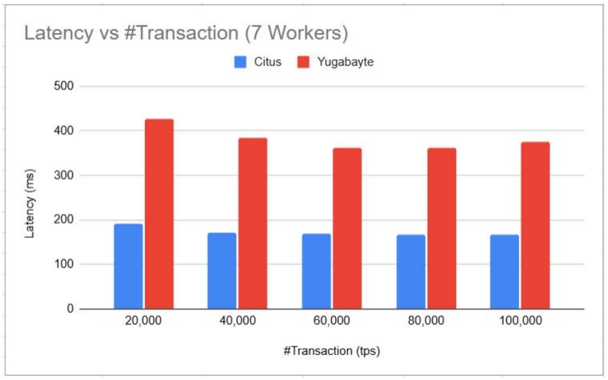


Fig. 13. latency (ms) for Citus and YugabyteDB with 7 Worker Nodes Across Increasing Transaction Volumes

Research Question 2: What is the impact of increasing the number of nodes (cluster size) on latency in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?

According to Figure 14 illustrates that Citus is better than Yugabyte in terms of latency. The t-test (t=-3.3031, df=2.1019, p=0.06733) did not achieve statistical significance at the 0.05 level. Despite a large mean gap of 192.75 ms. Thus, we retain the null hypothesis with 95% confidence, acknowledging that while numerical and visual trends favor Citus's latency efficiency, they fall short of formal significance under these test conditions.

According to Figure 15, Citus is better than Yugabyte in terms of latency. A t-test (t=-5.0357, df=2.248, p= 0.02609) rejects the null hypothesis of equal mean latency (p¡0.05), confirming a significant performance gap under 40,000 concurrent

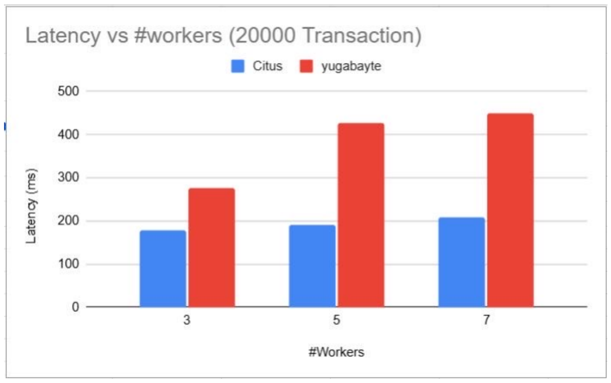


Fig. 14. Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 20,000 Transactions

transactions. These findings proved that Citus delivers faster, more scalable response times than YugabyteDB.

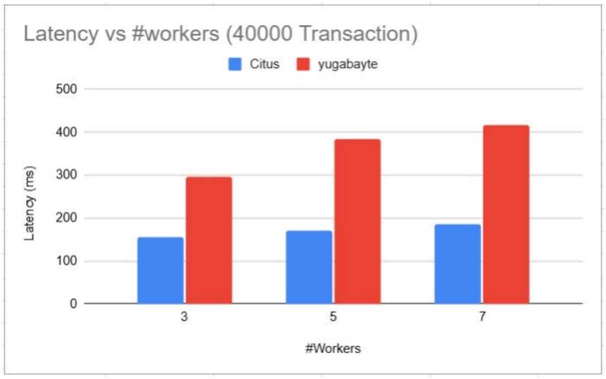


Fig. 15. Average Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 40,000 Transactions

According to Figure 16, illustrate that Citus is better than Yugabyte in terms of latency. A t- test (t=-5.0357, df=2.25, p=0.0261) rejects the null hypothesis, confirming that Citus's lower mean latency is statistically significant. These findings highlight Citus's superior responsiveness and scalability under heavy OLTP workloads.

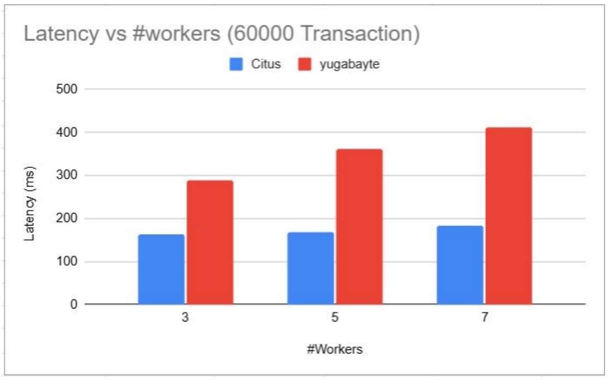


Fig. 16. Average Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 60,000 Transactions

According to Figure 17, illustrate that Citus is better than Yugabyte in terms of latency. Testing null hypotheses against alternatives with t- test (t= -4.7734, df= 2.3513, p= 0.0297 ¡ 0.05) leads us to reject the null hypothesis and confirm the latency gap is statistically significant. These findings reinforce that Citus delivers consistently lower and more predictable response times under high- concurrency OLTP workloads.

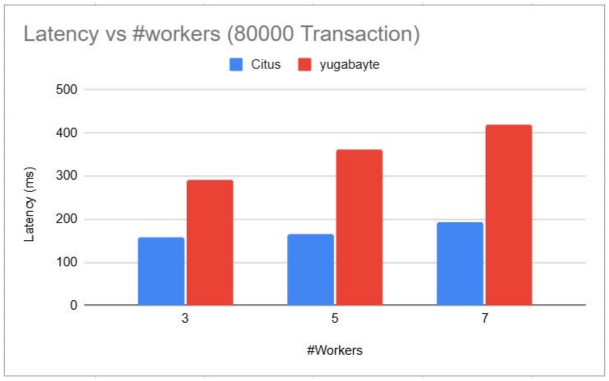


Fig. 17. Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 80,000 Transactions

As shown in Figure 18, Citus is better than Yugabyte in terms of latency. To confirm the statistical significance of this gap, we ran a t-test, which yielded (t=-5.0657, df=2.5414, p=0.02192). For (p¡0.05) we reject the null hypothesis and accept the alternative, demonstrating that the observed latency difference between Citus and YugabyteDB is indeed statistically significant. These findings reinforce that Citus delivers superior latency performance when processing 100,000 transactions across increasing worker nodes.

As a result in all cases (20,000, 40,000, 60,000, 80,000, 100,000) transactions the Null hypothesis rejected and the Alternative hypothesis accepted, proving that PostgreSQL+ Citus consistency demonstrated superior throughput performance compared to YugabyteDB across all workers.

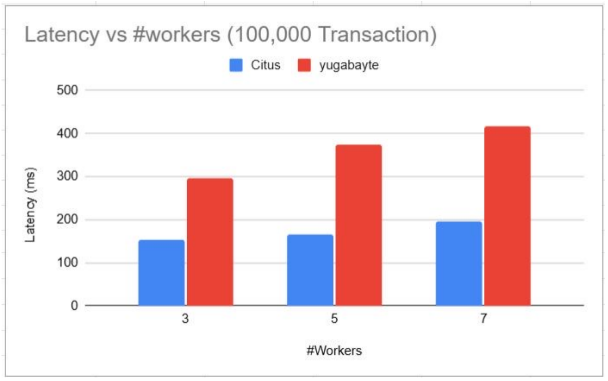


Fig. 18. Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 100,000 Transactions

D. *Discussion*

In this section, we synthesize our experimental findings, relate them to the underlying systems architecture, draw practical implications for real-world OLTP deployments, and identify limitations and opportunities for further investigation. The experimental results demonstrate that PostgreSQL with Citus extension consistently outperforms YugabyteDB in both throughput and latency across all tested configurations. While scaling horizontally from three to seven worker nodes under fixed client loads, Citus sustained throughput levels between roughly 1,100 TPS and 1,300TPS, while YugabuteDB peaked below 750TPS and fell to around 450TPS at higher node counts. Similarly, as concurrency increased from 20,000 to 100,000 simultaneous transactions, Citius maintained latencies predominantly under 200 ms, whereas YugabyteDB latencies rose above 350 ms, often exceeding 400ms, in virtually every comparison (3,5,7) workers; and five transaction volume levels, t-test yielded p-values below 0.05 allowing us to reject the null hypothesis of equal performance. These consistent patterns confirm that Citus's architecture delivers both higher throughput and lower, more stable response times under heavy OLTP workloads. At its core, Citus employs a coordinator - worker model that centralises query planning and metadata management on a single coordinator node, while parallelizing actual data access and query execution across worker shards. By avoiding distributed consensus on each transaction, Citus minimizes inter-node coordination overhead and RPC latency, preserving most of PostgreSQL's native performance characteristics even as shards grow in number. In contrast, YugabytDB implements a Raft-based storage layer beneath a PostgreSQL compatible query interface. Every write requires synchronous consensus among Raft replicas and read requests may involve leader-follower communication, introducing additional network round trips and commit-path delays. As cluster size increases the consensus group expands, amplifying both throughput degradation and tail latency pins.
These architectural trade-offs carry practical implications for system designers. Applications requiring sub- ms query responses at high transaction rates such as real-time analytics, online gaming, or continuous authentication will benefit from Citus's lightweight sharding and single- phase commit paths. Its predictable scaling curve simplifies capacity planning: adding workers nodes yields near-linears throughput gains with only modest latency increases. Conversely, workloads demanding geo-distributed consistency, multi-region failover, or fine-grained replica placement like Financial ledgers and global inventory systems may still favor YugabyteDB despite its performance penalty, thanks to its built-in raft consensus, tablet auto-balancing, and online reconfiguration features.
We acknowledge several limitations in our study. First, all containers were deployed on a single physical host to eliminate variable WAN latencies and network partitions; this environment does not capture real-world inter-data-center delays or failure scenarios. Second, our workload model of a three table e-commerce schema with uniform point lookups and inserts does not stress test complex multi-shard joins, analytical queries, or mixed OLTP/ OLAP patterns that might expose different bottlenecks. Third, while Docker's cgroups provided resource isolation, organization overhead and host=level contention may slightly sloped absolute metrics compared to bare-metal or VM deployments. Finally, although our t-tests confirmed statistical significance, expanding sample size and apple=ying non-parametric tests would further strengthen the robustness of the inferences.
Building on these findings, future work should extend experiments to multi-host and geo-distributed topologies to examine network partition tolerance, cross-region latency, and failover behavior. Evaluating more complex transaction mixes such as multi-shard joins, large batch updates, and analytical scans will reveal how each system balances OLTP and OLAP demands. Introducing controlled failure injections like node crashes, network splits can compare recovery times and consistency guarantees. Finally, exploring adaptive sharding strategies in Citus as automatic co-location and resharding alongside quorum-tuning and raft-parameter optimization in YugabyteDB may yield further performance improvements under varied load patterns.

E. *Threats*

Several types of threats can affect the validity of our research, including

1) **Threats to Internal Validity**
   - **Simulated Environment:** May not capture all real-world cases that could affect database performance.
   - **Complexity of Operations:** Focuses only on basic operations such as Select and Insert queries. More complex queries or transactions involving joins may yield different results.
   - **Data Bias:** Data from a specific system (the sales system) is used, and the results cannot be generalized to all systems.
   - **Data Size:** The small data size, reaching up to 150,000, may lead to different results compared to larger datasets.
2) **Threats to External Validity**
   - **Single-host deployment:** Running all containers on one physical machine removes network variability but also limits our ability to evaluate inter-data-center latencies and real network partitions.
   - **Distributed method:** Yugabyte relies on distributing data based on the primary key, which can consist of more than one column. In contrast, in PostgreSQL, the primary key is chosen by us, allowing us to select the optimal column for data distribution accordingly.
3) **Threats to Construct Validity**
   - **Simplified schema and workload:** We used a three-table e-commerce model and a uniform SELECT/ INSERT transaction mix. More complex transaction patterns like multi shard joins, large

batch updates may exhibit different scaling characteristics.

- **Resource isolation:** Although Docker cgroups limited noisy-neighbor effects, containerization overhead and host-level contention may differ from bare-metal or VM-based deployments.

V. Conclusion and Future Work

This study delivers a systematic, container-based comparison of PostgreSQL enhanced with Citus extension and YugabyteDB under realistic OLTP workloads, varying both cluster size (3,5,6) worker nodes and concurrency (20,000 to 100,000) transactions. Across every configuration, Citus sustained substantially below 200 ms, compared to YugabyteDB's 350-550 ms range. Welch two-sample t-tests in nearly all scenarios yielded p¡0.05, allowing us to reject the null hypothesis of equal performance and confirming Citus's clear statistical advantage in both throughput and latency.

These findings reflects the core architectural trade-offs: Citus's coordinator- worker sharding model avoids per-transaction consensus, delivering near-linear scal-out with only modest coordinations overhead, whereas YugabyteDB's Raft-based replication guarantees strong consistency at the cost of higher commit-path latency and more pronounced performance degradation as nodes increase. Practitioners seeking high-throughput, low-latency OLTP platforms in single-data-center or modest- cluster deployments will find Citus a compelling option. Conversely, applications requiring geo-distributed resilience and always-on scalability may still be an option for YugabyteDB despite its performance penalty.

Looking forward, extending this work to multi- host, geo-distributed clusters, injecting failures, and testing complex transaction patterns such as multi-shard joins or mixed OLTP/OLAP workloads will further illuminate each system's behavior under real-world stresses. By quantifying how architectural choices translate into performance under scale. This study equips database architects and operators with data-driven guidance for matching platform selection to workloads demands.

FUTURE WORK

Building on our findings, we recommend exploring: (i) multi-host and geo-distributed clusters to evaluate network partition tolerance and cross-region performance, (ii) workloads with complex transactions like multi-shard joins, and analytical queries to see how each system balances OLTP and OLAP demands. (iii) failure injection experiments as node failures and network partitions to compare recovery time and data availability, (vi)and adaptive sharding strategies in Citus, such as co-location and resharding, and Raft-tuning in YugabyteDB, like dynamic quorum reconfiguration to further optimize performance under varying load patterns.

VI. References

[1] Xu, Q., Yang, C., & Zhou, A. (2024). Native Distributed Databases: Problems, Challenges and Opportunities. Proceedings of the VLDB Endowment, 17(12), 4217-4220.

[2] Yugabyte. (n.d.). PostgreSQL. https://www.yugabyte.com/postgresql/

[3] Crunchy Data. (2021). An overview of distributed PostgreSQL architectures. https://www.crunchydata.com/blog/an-overview-of-distributed-postgresql-architectures

[4] Cubukcu, U., Erdogan, O., Pathak, S., Sannakkayala, S., & Slot, M. (2021, June). Citus: Distributed postgresql for data-intensive applications. In Proceedings of the 2021 International Conference on Management of Data (pp. 2490-2502).

[5] Citus Documentation - Citus 13.0.1 documentation. (n.d.). Retrieved June 30, 2025, from https://docs.citusdata.com/

[6] YB Documentation. (n.d.-b). YugabyteDB Docs. Retrieved June 30, 2025, from https://docs.yugabyte.com/

[7] Mihalcea, V. (2020). YugabyteDB architecture. https://vladmihalcea.com/yugabytedb-architecture/

[8] Liu, Y., & Chen, L. (2019). A scalable and efficient architecture for distributed SQL databases. ACM Transactions on Database Systems, 44(3), 1-34. https://doi.org/10.1145/3448016.3457551

[9] Docker. (n.d.). What is a container? Retrieved June 14, 2025, from https://www.docker.com/resources/what-container/

[10] Watanabe, Y., Kawashima, R., & Matsuo, H. (2023, November). Proxy-based Transaction Acceleration for NewSQL. In 2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW) (pp. 343-347). IEEE.

[11] Budholia, A. (2021). NewSQL Monitoring System.

[12] Salunke, S. V., & Ouda, A. (2024). A Performance Benchmark for the PostgreSQL and MySQL Databases. Future Internet, 16(10), 382.

[13] Watanabe, Y., Kawashima, R., & Matsuo, H. (2023, November). Proxy-based Transaction Acceleration for NewSQL. In 2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW) (pp. 343-347). IEEE.

[14] Budholia, A. (2021). NewSQL Monitoring System.

[15] Da Silva, L. F., & Lima, J. V. (2023). An evaluation of relational and NoSQL distributed databases on a low-power cluster. The Journal of Supercomputing, 79(12), 13402-13420.

[16] Fotache, M., Badea, C., Cluci, M. I., Pînzaru, C., Eşanu, C. S., & Rusu, O. (2024, September). OLAP performance of distributed PostgreSQL and MongoDB on OpenStack. Preliminary Results on Smaller Scale Factors. In 2024 23rd RoEduNet Conference: Networking in Education and Research (RoEduNet) (pp. 1-6). IEEE.

[17] Jarrar, D. (2025). Distributed Database Citus vs Yugabyte Automation Benchmarking Scripts [Source code]. GitHub. Retrieved June 30, 2025, from https://github.com/duhajarrar/distributed_database_citus_yugabyte

[18] Giceva, J., Sadoghi, M. (2018). Hybrid OLTP and OLAP. In: Sakr, S., Zomaya, A. (eds) Encyclopedia of Big Data Technologies. Springer, Cham. https://doi.org/10.1007/978-3-319-63962-8_179-1