

Birzeit University

Faculty of Engineering and Technology

Computer Science Department

SWEN7308: Distributed Systems

Project Report:

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

Prepared by:

Duha Jarrar 1225272

Hana AL Baidaq 1225365

Nahil Dkidek 1205359

Hiba Naser 1225406

Instructor:

Prof. Adel Al-Taweel

Date: 30 June 2025

Birzeit University

**Table of contents**

[**1. Introduction 2**](#_a1uch9xpsf3b)

[1.1. Motivation 3](#_t69a0s3lsdwt)

[1.2. Background 3](#_8f8rro6ohbtp)

[1.2.1. PostgreSQL with Citus Extension 3](#_z85nylyvrz7w)

[1.2.2. YugabyteDB 4](#_yv8w5gp8vn0d)

[1.2.3. Containerized Deployment Using Docker 5](#_zfyqk062dpxb)

[1.2.4. Online Transaction Processing (OLTP) 6](#_nqkcgxxrxh91)

[**2. Research Questions & Hypotheses 6**](#_aaik6c75rkte)

[**3. Experiment Design & Execution 7**](#_h7yketgov395)

[3.1. Selection of Database Systems 7](#_v9pczvmptvr)

[3.2. Dataset and Schema 8](#_1wtzadj9c0js)

[3.3. Data Distribution Strategy 8](#_6ecqc9rwgfon)

[3.4. Experiment Variables 8](#_xe74l9ui001z)

[3.5. Environment Setup 9](#_sv529b3t5idm)

[3.6. Workload Simulation and Benchmarking 9](#_gcpz7kcrhw9b)

[3.7. Experiment Execution 9](#_lizmgobo6sy5)

[3.8. Experiment Result 10](#_mpt1fqqapsnr)

[3.3.1. Throughput 10](#_esbtbxdjx4ki)

[3.3.2. Latency 11](#_8za48m7w03zp)

[**4. Data Analysis 11**](#_qiawtmpy3385)

[4.1. Throughput trends 11](#_q0a3ogbb4ojf)

[4.2. Latency Behaviour 16](#_1ghksx3k2mhv)

[**5. Discussion 21**](#_7kba4ux81jnf)

[5.1. Study Limitations 22](#_dtpy6841522x)

[5.2. Future work 22](#_e7oxtjz0sd79)

[**6. Conclusion 23**](#_7q27tdkah5mp)

[**7. References 24**](#_4q0ul8gvszxw)

# 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 as up to 300x 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 necessity of isolating these factors when measuring per-query latency and its scalability trade-offs.

Building on this foundation, our study delivers a comparative performance analysis of PostgreSQL with Citus and YugabyteDB, with latency and throughput as the central metrics. We execute point lookups, range scans, and mixed read/write transactions across 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, in comparison with 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.

Specifically, it evaluates:

* The impact of scaling the number of worker nodes on database performance.
* The effect of increasing concurrent 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.

## Motivation

Low latency and high throughput are 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<20 MS) to prevent motion sickness and ensure interactivity.

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

## Background

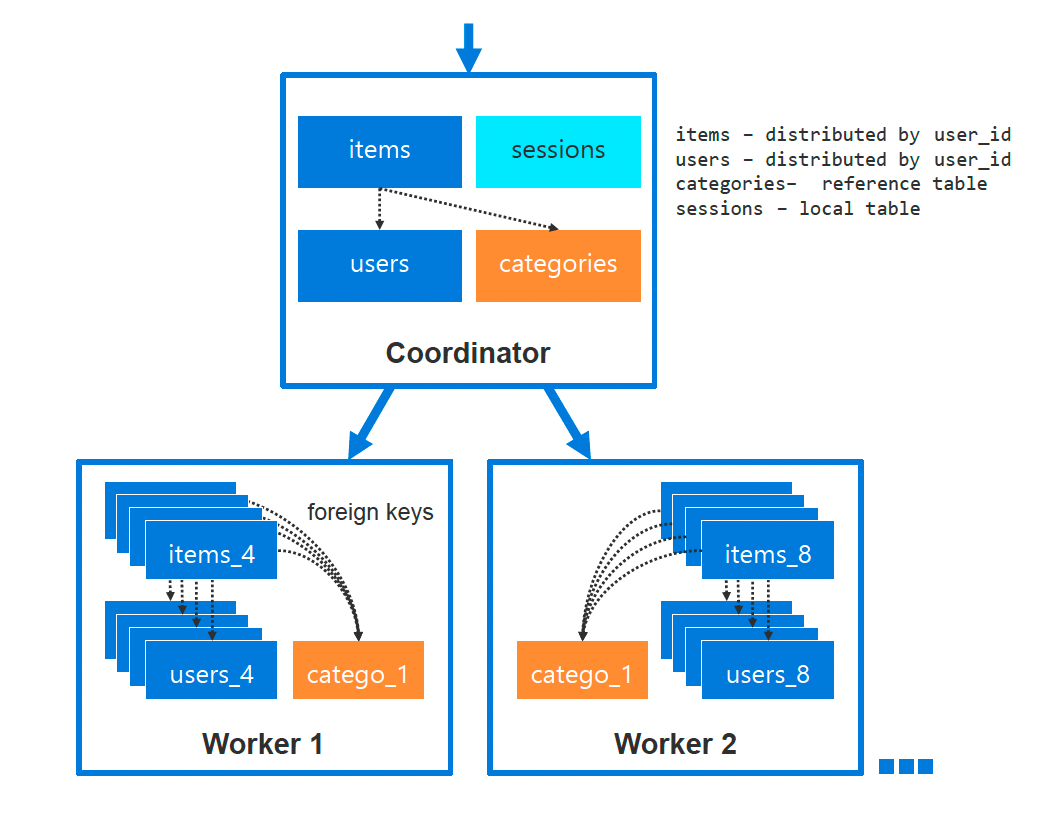
This section summarizes the architectures of the two distributed SQL systems under study, PostgreSQL augmented with the Citus extension and YugabyteDB, to highlight their contrasting approaches to data partitioning, query routing, and transaction coordination.

### PostgreSQL with Citus Extension

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 joining span shards, the coordinator incurs additional RPC round-trips and network serialization overhead [2].

The main Citus table types used in our experiment are:

* **Distributed Tables**: distribution is achieved by shredding across worker nodes using a chosen distribution column [4][5]. This type is ideal for large datasets and parallel query execution [4][5].
* **Reference Tables**: In this type, a selected reference table content concentrated in one shard is fully replicated on all workers [4][5]. This type is useful for small lookup tables that are frequently joined with distributed tables [4][5]. The figure below describes the usage of the reference table and distributed table [4]:

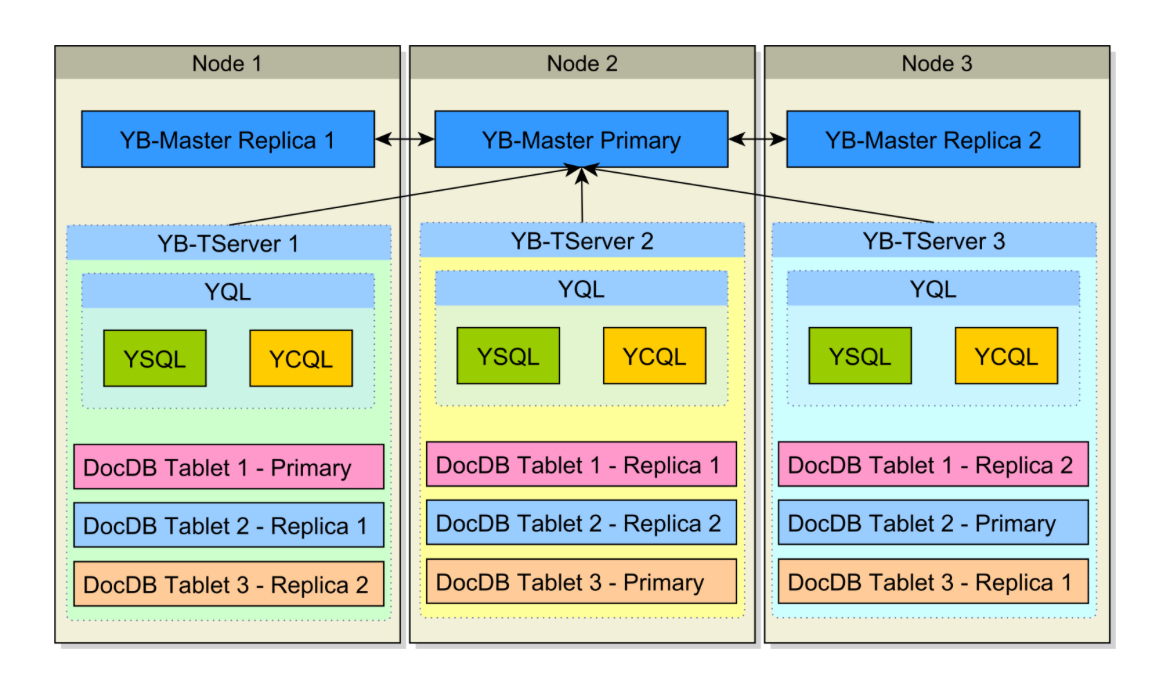


*Figure 1:Distributed Data with Reference Table [4]*

### YugabyteDB

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].



*Figure2: Distributed Data with Reference Table [5]*

To sum up the key differences between the 2 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] |

*Table#1: Key differences between Yugabyte and Postgres with Citus*

### Containerized Deployment Using Docker

To ensure environmental consistency, isolation, and rapid provisioning in our comparative evaluation of PostgreSQL with Citus and YugabyteDB, we employed Docker, which is a lightweight containerization platform that bundles an application and its dependencies into immutable images [14]. Docker’s client-daemon-image container architecture consists of a Docker client that issues commands to a long-running daemon [14], immutable layered images that serve a build artifacts, and runtime containers that share the host kernel while maintaining isolated namespaces for processes, file systems, and networks [14].

Docker images use a layered filesystem and a copy-on-write semantics [14]. Each build instruction creates a new read-only layer, cached locally to accelerate subsequent builds, while container write operations occur on an overlay layer atop the base image, avoiding mutation of underlying layers [14]. Additionally, Docker integrates with Linux cgroups to enforce resource controls (CPU shares, memory limits, block I/O throttling, and network bandwidth constraints), thereby preventing any single database instance from monopolizing host resources during parallel benchmarks.[15][16]. For multi-node orchestration, we used Docker Compose, defining coordinator and worker services.

### Online Transaction Processing (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 [17].

# Research Questions & Hypotheses

This study is structured around four central research questions:

1. **RQ1:** What is the impact of increasing the number of nodes (cluster size) on throughput in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?
2. **RQ2:** What is the impact of increasing the number of nodes (cluster size) on latency in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?
3. **RQ3:** How does raising the number of concurrent client load (user requests) affect throughput in PostgreSQL with Citus and YugabyteDB?
4. **RQ4:** How does raising the number of concurrent client load (user requests) affect latency in PostgreSQL with Citus and YugabyteDB?

We test these questions via **statistical hypotheses**:

**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 and YugabyteDB as the number of concurrent user requests increases in OLTP scenarios.

# 

# Experiment Design & Execution

The primary objective of this experiment is to assess how two distributed SQL database systems, PostgreSQL enhanced with the Citus extension and YugabyteDB, perform under online transaction processing (OLTP) workloads. Specifically, the study aims to analyze how these systems respond to variations in horizontal scalability (number of worker nodes) and concurrency (number of simultaneous client requests), focusing on key performance metrics: **transaction throughput** and **transaction latency**.

## Selection of Database Systems

We selected **PostgreSQL with the Citus extension** and **YugabyteDB** as the two distributed database systems for this study. Citus transforms a single-node PostgreSQL instance into a distributed system by implementing sharding and parallel query execution across worker nodes, while maintaining full SQL compliance. YugabyteDB, on the other hand, is a fully distributed SQL database built on a PostgreSQL-compatible query layer, integrated with a distributed storage engine using Raft-based replication. The architectural contrast between these systems, Citus as an extension versus YugabyteDB as a native distributed system, makes them ideal candidates for evaluating the effects of horizontal scaling and concurrency on OLTP performance.

## Dataset and Schema

To simulate a realistic transactional workload, we implemented a retail-oriented OLTP schema consisting of three interrelated tables: customer, product, and order\_trans. The customer table stores customer profile information, while the product table contains item listings with prices and inventory levels. The order\_trans table records purchase transactions and references both the customer and product tables via foreign keys. The schema was designed to support both read-intensive (e.g., browsing products) and write-intensive (e.g., placing orders) operations.

## Data Distribution Strategy

In Citus, we used the CREATE DISTRIBUTED TABLE command to shard the order\_trans, customer tables by *customer\_id* across available worker nodes, with reference tables (product) replicated to each node to support joins. This ensures parallel execution of transactions while maintaining query consistency.

In YugabyteDB, tables were automatically sharded across tablet servers using primary keys.

## Experiment Variables

The experiment examined the effect of **three independent variables**:

1. **Database system**: PostgreSQL with Citus vs. YugabyteDB.
2. **Number of worker nodes (cluster size)**: 3, 5, and 7.
3. **Concurrent client load(client request)**: 20,000, 40,000,60,000, 80,000, and 100,000 simulated clients.

**Dependent variables**

* **Transaction Throughput** (TPS): number of successful transactions per second
* **Transaction Latency** (ms): average response time per transaction

**Constant Variables**

To ensure experimental consistency and validity, several parameters were held constant across all test scenarios:

* **Hardware environment**: All tests were conducted on the same host machine.
* **Schema and data volume**: An Identical SQL schema and data size were used in all runs.
* **Transaction type and structure**: A fixed workload script was used for all tests, including SELECT and INSERT transactions.
* **Benchmarking tool and configuration**: The same version of pgbench with identical script logic was used throughout.
* **Container images and OS**: Standardized Docker images were used for both database systems to ensure uniformity.

## Environment Setup

The experiments were conducted on a single physical host equipped with a 12th Gen Intel Core i7-1255U CPU (12 cores), 16 GB of RAM, and a 64-bit operating system. All components of the experiment were deployed in isolated containers using Docker.

We created distributed clusters using Docker Compose to simulate 3, 5, and 7 worker nodes deployments. Each Citus cluster consisted of one coordinator node and multiple worker nodes, while each YugabyteDB cluster consisted of one master and multiple worker servers. Docker Compose files were configured to automatically launch and interconnect the containers, enabling repeatable and isolated test environments.

## Workload Simulation and Benchmarking

We used the pgbench tool to simulate OLTP transactions and measure system performance. A custom workload was created using SQL scripts that executed realistic operations, including browsing products and placing orders. An example of the workload is as follows:

-- Select products by price

SELECT \* FROM product ORDER BY price ASC LIMIT 10;

-- Insert new order

INSERT INTO order\_trans ()

SELECT nextval('order\_id\_seq'), :customer\_id, :product\_id, now(), ..

FROM product

WHERE product\_id = :product\_id;

Transactions were executed with prepared statements. We varied the number of concurrent client requests (from 20,000 to 100,000).

## Experiment Execution

To ensure that the observed differences in performance were due to the manipulated variable and not other confounding factors, we applied **controlled experimentation** by varying one independent variable at a time while holding the others constant. The following design strategy was applied:

* **To assess the effect of worker nodes**:  
   We fixed the number of client requests (e.g., at 20,000) and compared performance as the number of nodes increased from 3 to 5 to 7. This allowed us to isolate the impact of horizontal scaling on throughput and latency, without interference from changes in client load.
* **To assess the effect of the number of client requests**:  
   We fixed the cluster size (e.g., 3 nodes) and varied the number of concurrent client connections from 20,000 to 100,000. This configuration allowed us to study how well each system handled increased transaction pressure independently of cluster size.
* **To compare database systems**:  
   The same configurations were applied for both Citus and YugabyteDB, using identical data, workloads, and client loads. This direct pairing enabled meaningful performance comparisons between the two systems under the same experimental conditions.

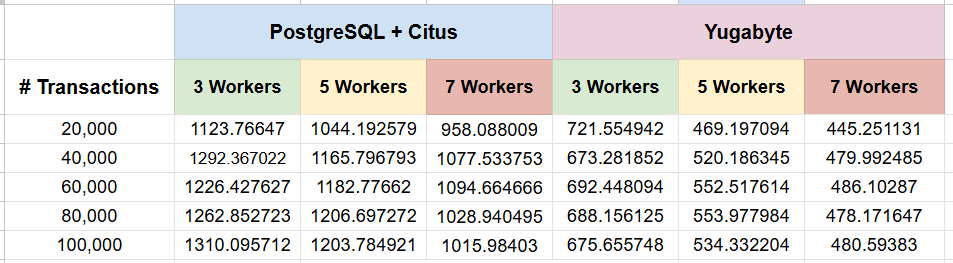
**Note:** All experiment steps were automated using Python and shell scripts that orchestrated cluster deployment, workload execution via pgbench, and result collection. This script ensured reproducibility, consistency, and reduced manual error across all test configurations. The full automation code is publicly available at <https://github.com/duhajarrar/distributed_database_citus_yugabyte> [13]. You can find all requested files:

1. Raw data in [Experiment Data](https://github.com/duhajarrar/distributed_database_citus_yugabyte/tree/master/Experiment%20Data) file.
2. Data analysis including graphs in the Data [Analysis](https://github.com/duhajarrar/distributed_database_citus_yugabyte/tree/master/Data%20Analysis) file.
3. [README.md](https://github.com/duhajarrar/distributed_database_citus_yugabyte/blob/master/README.md) file including some details how to set up the system to run our conducted experiment.

## Experiment Result

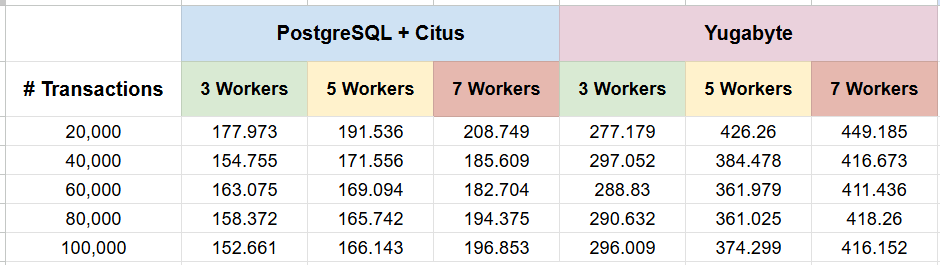
The findings in Table 1 is the Transaction Throughput and Table 2 is Transaction Latency were generated directly by executing our automated benchmarking tests. We used the pgbench utility to simulate OLTP workloads on PostgreSQL deployed with the Citus extension and YugabyteDB for various worker node 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.

### Throughput

****

*Table #2. Throughput (Transactions Per Second) result for Citus and YugabyteDB Under Varying Worker Counts and Transaction Volumes*

### Latency

****

*Table 3. Transaction Latency (ms) for Citus and YugabyteDB Under Varying Worker Counts and Transaction Volumes*

# 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 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.

## 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 addresses:

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

To investigate this, the formulated Hypotheses :

* **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.

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.

**At 3- worker nodes:**

| *Table #4: Throughput (tps) for Citus and YugabyteDB with 3 Worker Nodes Across Increasing Transaction Volumes* | *Figure #3: Throughput (tps) for Citus and YugabyteDB with 3 Worker Nodes Across Increasing Transaction Volumes* | *Figure #4: t-Test Results Comparing Throughput Between Citus and YugabyteDB with 3 Worker Nodes Across Increasing Transaction Volumes* |
| --- | --- | --- |

Our 3- worker nodes benchmarks reveal a clear performance advantage for PostgreSQL+Citus over YugabyteDB across all transaction volumes as in table 4 and Chart 3. At 60,000 transactions, Citus delivered 1226.43 TPS, which is about 77% higher than YugabyteDB’s of 692.45 TPS, while maintaining stable scalability as load increased. 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.

**At 5- worker nodes:**

| *Table #5: Throughput (tps) for Citus and YugabyteDB with 5 Worker Nodes Across Increasing Transaction Volumes* | *Figure #5: Throughput (tps) for Citus and YugabyteDB with 5 Worker Nodes Across Increasing Transaction Volumes* | *Figure #6: t-Test Results Comparing Mean Throughput Between Citus and YugabyteDB with 5 Worker Nodes* |
| --- | --- | --- |

Our 5-worker node benchmarks results confirm Citus's clear throughput advantage over YugabyteDB at every load. At 80,000 transactions, Citus hit 1206.70 TPS versus YugabyteDB 553.98 TPS, around 118% uplift. The t-test (p=1.49\*10 -6 ) strongly rejects the null hypothesis. Mean throughput was 1,160.70 TPS for Citus against 526.04 TPS for YugabyteDB ( 𝞓= 643.66 TPS), demonstrating that Citus scales OLTP workloads far more efficiently under identical horizontal scaling.

**At 7- worker nodes:**

| *Table #6: Throughput (tps) for Citus and YugabyteDB with 7 Worker Nodes Across Increasing Transaction Volumes* | *Figure #7: Throughput (tps) for Citus and YugabyteDB with 7 Worker Nodes Across Increasing Transaction Volumes* | *Figure #8: t-Test Comparing Throughput of Citus and YugabyteDB with 7 Worker Nodes* |
| --- | --- | --- |

In the 7-worker node benchmarks Citus shows sustaining a clear throughput lead across all loads. At the 60,000 transactions, Citus delivered 1094.66 TPS versus YugabyteDB’s 486.10 TPS, approximately 125% uplift. The t-test (t=20.727, p= 7.38\* 10-6 ) strongly rejects the null hypothesis. Mean throughput was 1,015.99 TPS for Citus versus 472.02 TPS for YugabyteDB (𝞓= 543.97 TPS), confirming Citus’s superior scalability under heavy OLTP workloads.

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.

1. **Research Question 1 (RQ1): What is the impact of increasing the number of nodes (cluster size) on throughput in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?**

To investigate this, the formulated Hypotheses :

* **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.

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.

**At 20,000 transactions:**

| *Table #7: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 40,000 Transactions* | *Figure #9: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 20,000 Transactions* | *Figure #10. t-Test for Throughput Differences Between Citus and YugabyteDB at 20,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

Citus exceeded YugabteDB at every scale, delivering 1,123.77 TPS with 3 workers and easing to 958.09 TPS 7 workers, while YugabyteDB fell from 722 TPS down to 445.25 TPS, revealing Citus’s steadier horizontal scaling. 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.

**At 40,000 transactions:**

| *Table #8: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 40,000 Transactions* | *Figure #11: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 40,000 Transactions* | *Figure #12:t-Test for Throughput Differences Between Citus and YugabyteDB at 40,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

Citus peaked at 1292.37 TPS with 3 workers and sustained 1,077.53 TPS at 7 workers, while YugabyteDB slid from 673 TPS to 479.99 TPS, demonstrating Citus’s stronger scalability. 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.

**At 60,000 transactions:**

| *Table #9: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 60,000 Transactions* | *Figure #13: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 60,000 Transactions* | *Figure #14:t-Test for Throughput Differences Between Citus and YugabyteDB at 60,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

Citus held 1,226.43 TPS at 3 workers and only dipped to 1,094.66 TPS at 7 workers, while YugabyteDB fell from 692 TPS to 486.10 TPS, showing Citus’s resilience against coordination overhead. 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.

**At 80,000 transactions:**

| *Table #10:Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 80,000 Transactions* | *Figure #15: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 80,000 Transaction* | *Figure #16:t-Test for Throughput Differences Between Citus and YugabyteDB at 80,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

Citus exceeded YugabyteDB at every scale, rising from 1.262.85 TPS with 3 workers to 1,028.94 TPS at 7 workers, while YugabyteDB dropped from 688 TPS to 478,17 TPS, showing Citus’s steadier scalability. Testing the null hypothesis against alternative hypothesis with t- test ( t=6.3404, p=0.00375) leads to rejecting the null hypothesis and confirming approximately a 593 TPS mean advantage for Citus under heavy OLTP loads.

**At 100,000 transactions:**

| *Table #11: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 100,000 Transactions* | *Figure #17: Throughput (tps) for Citus and YugabyteDB Across Varying Worker Node Counts with 100,000 Transactions* | *Figure #18:t-Test for Throughput Differences Between Citus and YugabyteDB at 100,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

The results showed that Citus sustained 1,310.06 TPS at 3 workers down to 1,015.98 TPS at 7 workers, while YugabyteDB slid from 676 TPS to 480.59, highlighting Citus’s superior horizontal scalability. 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, and confirming a 747 TPS mean advantage for Citus.

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.

## 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 addresses:

1. **RQ4:** How does raising the number of concurrent client load (user requests) affect latency in PostgreSQL with Citus and YugabyteDB?

To investigate this, the formulated Hypotheses :

* **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 and YugabyteDB as the number of concurrent user requests increases in OLTP scenarios.

**At 3 workers:**

| *Table #12: latency (ms) for Citus and YugabyteDB with 3 Worker Nodes Across Increasing Transaction Volumes* | *Figure #19: latency (ms) for Citus and YugabyteDB with 3 Worker Nodes Across Increasing Transaction Volumes* | *Figure #20: t-Test Comparing Latency of Citus and YugabyteDB with 3 Worker Nodes* |
| --- | --- | --- |

Table 12 and the barchart figure 19 shows that Citus delivers consistently lower latencies than YugabyteDB at every load level. For example, at 60,000 transactions Citus averaged 163.08 ms versus YugabyteDB’s 288.83 ms, nearly cutting response time in half. Across the full range of tested loads, Citus not only sustains lower absolute latencies but also exhibits tighter variance underscoring its more predictable OLTP performance.

We confirmed this gap 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. The group means 161.37 ms for Citus’s clear advantage in response time with 3 worker nodes.

**At 5 workers:**

| *Table #13: latency (ms) for Citus and YugabyteDB with 5 Worker Nodes Across Increasing Transaction Volumes* | *Figure #21: latency (ms) for Citus and YugabyteDB with 5 Worker Nodes Across Increasing Transaction Volumes* | *Figure #22: t-Test Comparing Latency of Citus and YugabyteDB with 5 Worker Nodes* |
| --- | --- | --- |

Results showed that Citus consistently delivers lower latency than YugabyteDB at every load level. At 40,000 transactions Citus averaged 171.56 ms versus YugabyteDB’s 384.48 ms, a gap of over 200ms highlighting Citus’s faster and more predictable OLTP performance.

We tested:

* Null hypothesis latency difference between Citus and YugabyteDB
* Alternative hypothesis: a significant difference exists.

A t-test (t= -16.194, df= 5.254, p = 1.009\* 10-5)rejects the null hypothesis. Mean latencies were 1272.82 ms for Citus and 381.61 ms for YugabyteDB ( 𝞓= 208.80 ms) confirming statistically significant advantage for Citus.

**At 7 workers:**

| *Table #14: latency (ms) for Citus and YugabyteDB with 7 Worker Nodes Across Increasing Transaction Volumes* | *Figure #23: latency (ms) for Citus and YugabyteDB with 7 Worker Nodes Across Increasing Transaction Volumes* | *Figure #24: t-Test Comparing Latency of Citus and YugabyteDB with 7 Worker Nodes* |
| --- | --- | --- |

As illustrated in results above, Citus delivers significantly lower latency than YugabyteDB at every load, averaging 194.38 ms versus 418.27 ms at 80,000 transactions, a 228.68 ms improvement.

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.

1. **RQ2:** What is the impact of increasing the number of nodes (cluster size) on latency in PostgreSQL with Citus versus YugabyteDB under OLTP workloads?

To investigate this, the formulated Hypotheses:

**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.

**At 20,000 transactions :**

| *Table #15: Average Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 20,000 Transactions* | *Figure #25: Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 20,000 Transactions* | *Figure #26: t-Test for Latency Differences Between Citus and YugabyteDB at 20,000 Transactions Across Varying Worker Counts* |
| --- | --- | --- |

Results illustrate that Citus’s latency rises modestly from 177.97 ms on 3- worker nodes to 208.75 ms on 7- worker nodes, while YugabyteDB’s jumps sharply from 277.18 ms to 449.19 ms, this underscoring Citu’s smoother scalability under horizontal expansion.

A 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 at 95% confidence, acknowledging that while the numerical and visual trends favor Citus’s latency efficiency, they fall short of formal significance under these test conditions.

**At 40,000 transactions :**

| *Table #16: Average Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 40,000 Transactions* | *Figure #27: Average Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 40,000 Transactions* | *Figure #28: t-Test for Latency Differences Between Citus and YugabyteDB at 40,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

The appeared results in above showed that Citus maintains markedly lower latency than YugabyteDB as worker count rises, from 154.76ms at 3 workers to 185.61 ms at 7 workers, while YugabyteDb climbs steeply from 297.00ms to 416.67 ms, reflecting less efficient horizontal scaling. 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 transactions. These findings proved that Citus delivers faster, more scalable response times than YugabyteDB.

**At 60,000 transactions :**

| *Table #17: Average Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 60,000 Transactions* | *Figure #29: Average Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 60,000 Transactions* | *Figure #30: t-Test for Latency Differences* |
| --- | --- | --- |

Again the results illustrate that at 60,000 transactions Citus’s latency climbs modestly from 154.76 ms with 3- workers to 185.61 ms with 7 workers, while YugabyteDB latency jumps steeply from 297.00 ms to 41.6.67 ms, underscoring Citus smoother horizontal scaling. 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.

**At 80,000 transactions :**

| *Table #18: Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 80,000 Transactions* | *Figure #31: Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 80,000 Transactions* | *Figure #32: t-Test for Latency Differences Between Citus and YugabyteDB at 80,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

Results at 80,000 transactions across 3-7 workers we found that Citus’s latency climbed modestly from 158.37 ms to 194.38 ms, whereas YugabyteDB’s latency surged escalate from 291.00 ms to 418.27 ms, highlighting Citus’s more efficient scaling. 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.

**At 100,000 transactions :**

| *Table #19: Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 100,000 Transactions* | *Figure #33: Latency (ms) for Citus and YugabyteDB Across Varying Worker Node Counts with 100,000 Transactions* | *Figure #34: t-Test for Latency Differences Between Citus and YugabyteDB at 100,000 Transactions Across Varying Worker Count* |
| --- | --- | --- |

As shown in results above, Citus maintains lower latency as worker count increase, rising moderately from 152.66 ms with 3 workers to 196.85 ms with 7 workers, whereas YugabyteDB’s latency Jumps sharply from 296.00 ms to 416.15 ms over the same range, underscoring its less efficient horizontal scaling under high load.

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, across our five workload levels,the **only case where we could reject the null hypothesis was at 20,000 transactions (p>0.05)**. For 40,000 (p=0.02609), 60,000 (p=0.2971), 80,000 (p=0.00375), and 100,000 (p=0.02192), each t-test produced p<0.05, so we consistently reject the null hypothesis and accept the Alternative, confirming that Citus’s latency advantage over YougabyteDB is statistically significant at higher OLTP loads.

# 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.

## Study Limitations

Limitations of our study which are subject to several constraints:

* 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
* 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.
* Statistical breadth: our t-test demonstrates significance over multiple runs, but a larger sample size and non-parametric tests could strengthen the robustness of the inferences.

## 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.

# Conclusion

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.

# 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] Jarrar, D. (2025). *Distributed Database Citus vs Yugabyte Automation Benchmarking Scripts* [Source code]. GitHub.<https://github.com/duhajarrar/distributed_database_citus_yugabyte>

[14] Kithulwatta, W. M. C. J. T., Wickramaarachchi, W. U., Jayasena, K. P. N., Kumara, B. T. G. S., & Rathnayaka, R. M. K. T. (2021). Adoption of docker containers as an infrastructure for deploying software applications: A review. Advances on Smart and Soft Computing: Proceedings of ICACIn 2021, 247-259.

[15] Muzumdar, P., Bhosale, A., Basyal, G. P., & Kurian, G. (2024). Navigating the Docker ecosystem: A comprehensive taxonomy and survey. arXiv preprint arXiv:2403.17940.

[16] Thokala, V. S. (2021). Utilizing Docker Containers for Reproducible Builds and Scalable Web Application Deployments. Int. J. Curr. Eng. Technol, 11(6), 661-668.

[17] 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>