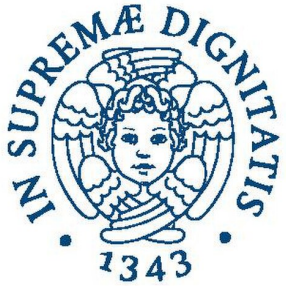

UNIVERSITÀ DI PISA

Computer Engineering

Artificial Intelligence and Data Engineering

Project Documentation of *Concurrent Database Access Simulation*

Authors:
Rotelli Luca - Sfar Omar Tomàs - Gashi Greta

Academic Year: 2025-2026

Contents

Chapter 1

Introduction

This project implements a discrete event simulation of a concurrent database access system using the **OMNeT++ framework** (Discrete Event Simulator).

1.1 Objectives

The main objective is to evaluate the performance of a multi-client database system by implementing a readers/writers model with concurrency control through mutual exclusion and FCFS (First Come First Served) queueing.

1.2 Context

The simulation analyzes how the system responds to varying loads of read and write operations from N concurrent users accessing M database tables.

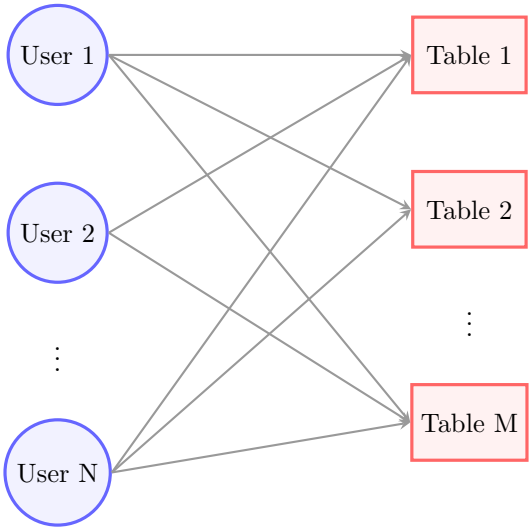

Figure 1.1: Simplified interaction schema: N Users accessing M Tables concurrently.

Chapter 2

Problem Definition

2.1 Problem Statement

- N concurrent users randomly access M database tables.
- Each user generates requests according to a Poisson process (rate λ).
- Each request is a **READ** with probability p, or **WRITE** with probability $(1 - p)$.
- Table selection follows a specified distribution (uniform or lognormal).
- Each access requires a fixed service time S.

2.2 Constraints

- **Reads:** Multiple simultaneous reads on the same table are allowed (reader lock).
- **Writes:** Exclusive access only (write lock), blocked by active readers.
- **Queuing:** FCFS to handle conflicting requests.
- **Mutual Exclusion:** Implemented via reader/writer counters.

2.3 Key Performance Metrics

- Average wait time (from request submission to response).
- Throughput (completed accesses per second).
- Maximum/average queue length per table.
- Table utilization (fraction of time occupied).

- Percentage of completed operations.

Chapter 3

System Implementation and Verification

3.1 System Model

3.1.1 Architecture

- **Network:** Fully-connected mesh topology with N users and M tables.
- **Communication:** Asynchronous message-passing between modules.
- **Timing:** Discrete event simulation with `simTime()` in seconds.

3.1.2 Temporal Model

- **Inter-arrival times:** Exponential with rate $\lambda = 1/T_{inter}$. Generator: `exponential(1/lambda)`.
- **Table selection:**
  - Uniform: `intuniform(0, numTables-1)`.
  - Lognormal: `lognormal(m, s)` mapped to $[0, M-1]$.
- **Response time:** Queueing time + Service time.

3.2 Modules and Components

3.2.1 User Module

Defined in `User.h/cc, User.ned`. It generates database access requests.

Behavior: Schedules accesses per Poisson process with rate λ. For each access, it selects a table and type (read/write), sends the request, and records statistics upon response.

3.2.2 Table Module

Defined in `Table.h/cc, Table.ned`. Simulates concurrent access to a single table.

Internal State: `activeReaders`, `writeActive`, `requestQueue` (FCFS), `serviceEvents`.

Mutual Exclusion Logic:
- If `writeActive=true`: all new accesses blocked.
- If `activeReaders>0`: new reads OK, new writes blocked.
- If `activeReaders=0` and `writeActive=false`: both reads and writes OK.

3.2.3 Network

Defined in `DatabaseNetwork.ned`.

- **Parameters:** `numUsers` (default 60), `numTables` (default 20).
- **Topology:** Fully-connected mesh where each `user[i]` sends requests to any `table[j]`.

3.2.4 Module Parameters

Table 3.1 summarizes the configurable parameters:

Module	Parameter	Description
User	`lambda`	Rate of exponential inter-arrival time (λ)
User	`readProbability`	Probability p of read operation
User	`serviceTime`	Duration of database operation S
User	`tableDistribution`	Table selection distribution
Table	`tableId`	Unique table identifier
Network	`numUsers`	Number of users N
Network	`numTables`	Number of tables M

Table 3.1: Module Parameters

3.3 Implementation Details

3.3.1 Design Choices

- **Statistics Collection:** Uses signal mechanism (`registerSignal/emit`) via `@signal` and `@statistic` in NED files. Output in `.vec` and `.sca`.
- **RNG:** OMNeT++ default Mersenne Twister seeded from `omnetpp.ini`.
- **Message Passing:** `cMessage` with parameters (`userId`, `arrivalTime`, `serviceTime`). Kind: 0=READ, 1=WRITE.

3.3.2 Concurrency Management Algorithm

The `processQueue` method ensures FCFS and Mutual Exclusion:

1. If `writeActive=true`: Return (table locked).
2. Process queue FCFS:
   - If request is **READ** and `activeReaders` ≥ 0: Pop, increment `activeReaders`, start service. Continue loop (parallelism).
   - If request is **WRITE**:
     - If `activeReaders` $== 0$: Pop, set `writeActive=true`, start service, break.
     - Otherwise: Wait (blocks later requests for FCFS).

3.4 Verification

3.4.1 Degeneracy Test

The degeneracy test analyzes the system's behavior under extreme or degenerate parameter values:

1. **Zero users ($N = 0$)**: The system enters an idle state with zero utilization, as expected due to absence of requests.
2. **Single table ($M = 1$)**: All requests are directed to a single table, creating a bottleneck. The system behaves as a simple M/G/1 queue.
3. **Read probability $p = 1$**: All operations are reads. Multiple users can access tables concurrently without blocking, resulting in minimal waiting times.
4. **Read probability $p = 0$**: All operations are writes. Each table becomes a simple FIFO queue with exclusive access, maximizing contention.

3.4.2 Continuity Test

To verify the model's accuracy, we compare two configurations with slightly different parameter values [Table 3.2] and verify that outputs change proportionally.

Note: We vary the number of users (N) rather than the read probability (p) because:

- The effect on metrics is **linear and proportional**
- It doesn't change the nature of the system (read/write ratio stays constant)
- All metrics scale predictably with the load increase

Parameter	Config A	Config B
N (users)	100	101 (+1)
M (tables)	20	20
p (read probability)	0.50	0.50
λ (request rate)	1.0	1.0
S (service time)	0.1s	0.1s
Repetitions	25	25

Table 3.2: Two slightly different configurations for the continuity test

Simulating both configurations with 25 repetitions, the following chart [Figure 3.1] shows the comparison at a 95% confidence level:

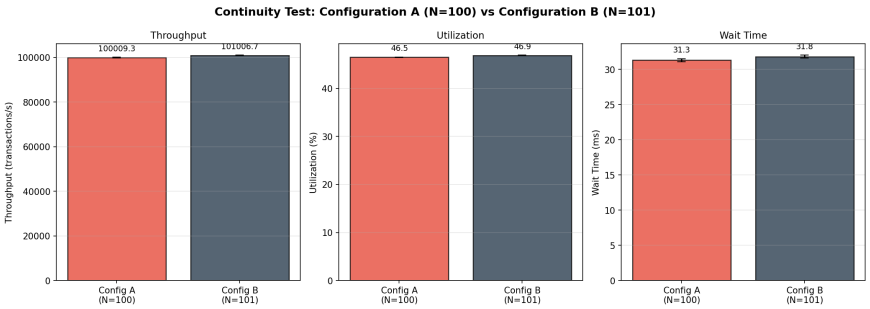

Figure 3.1: Continuity test comparing Configuration A ($N = 100$) vs Configuration B ($N = 101$) at 95% confidence level

The results [Figure 3.1] show that adding just 1 user (+1%) produces proportionally small changes in throughput:

- Configuration A (N=100): mean throughput $\approx 100,009$ transactions
- Configuration B (N=101): mean throughput $\approx 101,007$ transactions
- Variation: $\approx +1\%$ (as expected from +1 user)

The bar chart shows mean values with 95% confidence intervals. The proportional increase in throughput (+1%) exactly matches the proportional increase in users (+1%), confirming that the system exhibits **linear scaling** behavior. The **continuity test passes**.

3.4.3 Consistency Test

To validate consistency, we study the system's behavior by varying the number of users N while keeping other parameters fixed [Table 3.3].

Parameter	Value
M (tables)	10
λ (request rate)	0.05 req/s
p (read probability)	0.5
S (service time)	0.1s
N (users)	10, 50, 100, 500, 1000

Table 3.3: Configuration adopted for consistency test simulation runs

As expected, utilization increases linearly with the number of users, and the system shows consistent behavior across all configurations. The following chart [Figure 3.2] compares throughput and utilization across different user populations:

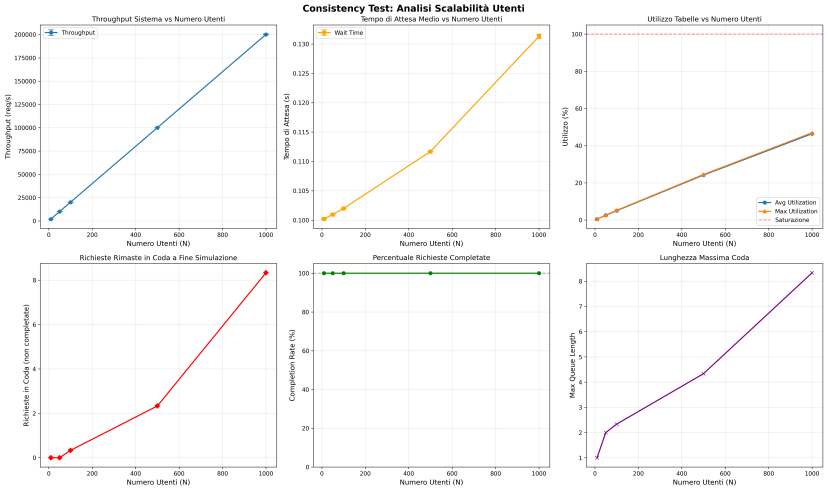

Figure 3.2: Consistency test: utilization and throughput vs number of users at 95% confidence interval

The throughput scales linearly with the number of users until the system approaches saturation, demonstrating consistent and predictable behavior.

3.4.4 Theoretical Verification

Given that our system can be modeled as an **Open Queueing Network**, we can theoretically compute its performance indices.

System Model

The system is an **open queueing network** because each user generates requests with an exponential inter-arrival time **independently** of whether previ-

ous requests have been served. Users do not wait for a response before generating the next request.

System parameters:

- $M = 10$ service centers (database tables)
- N = number of users
- λ = request rate per user (requests/second)
- $S = 0.1$s service time per request

Utilization Formula

For an open queueing network with N users each generating requests at rate λ:

- Total arrival rate to the system: $\Lambda = N \cdot \lambda$
- With uniform routing to M tables: $\lambda_i = \frac{N \cdot \lambda}{M}$ per table
- Utilization per table: $U = \lambda_i \cdot S$

Therefore, the theoretical utilization is:

$$U = \frac{N \cdot \lambda \cdot S}{M} \tag{3.1}$$

For example, with $N = 100$, $\lambda = 0.05$, $S = 0.1$, $M = 10$:

$$U = \frac{100 \cdot 0.05 \cdot 0.1}{10} = \frac{0.5}{10} = 0.05 = 5\% \tag{3.2}$$

Theoretical vs Empirical Comparison

Collecting utilization data from our simulations, we compare with theoretical predictions [Figure 3.3]:

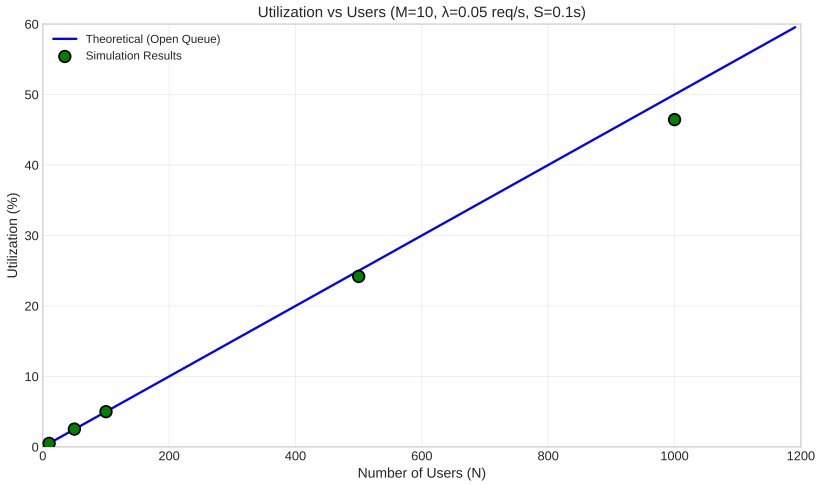

Figure 3.3: Comparison between theoretical model (blue line) and simulation results (green dots)

K Users	Empirical Util %	Theoretical Util %	Error %
10	0.50	0.50	0.95
50	2.53	2.49	1.58
100	5.00	4.98	0.43
500	24.19	24.88	2.77
1000	46.44	49.75	6.66

Table 3.4: Comparison of empirical vs theoretical utilization

The growing discrepancy at high loads ($N = 1000$) is attributed to the concurrency of read operations. The theoretical model assumes strict serialization of all tasks (U_{load}), whereas the simulation allows parallel processing of reads, resulting in a lower physical busy time (U_{busy}) compared to the offered load.

Per-Table Utilization

For $K = 500$ users, we verify that load is uniformly distributed across tables [Figure 3.4]:

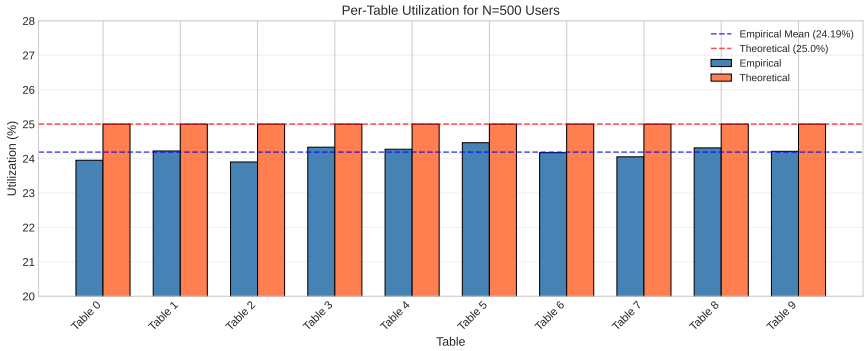

Figure 3.4: Per-table utilization for K=500 users showing uniform load distribution

The per-table utilization varies within a narrow range (23.90% – 24.46%) around the theoretical value of 24.88%, confirming uniform load balancing.

3.4.5 Throughput Analysis

The system throughput follows the theoretical prediction [Figure 3.5]:

$$\gamma = N \cdot \lambda \text{ requests/second} \tag{3.3}$$

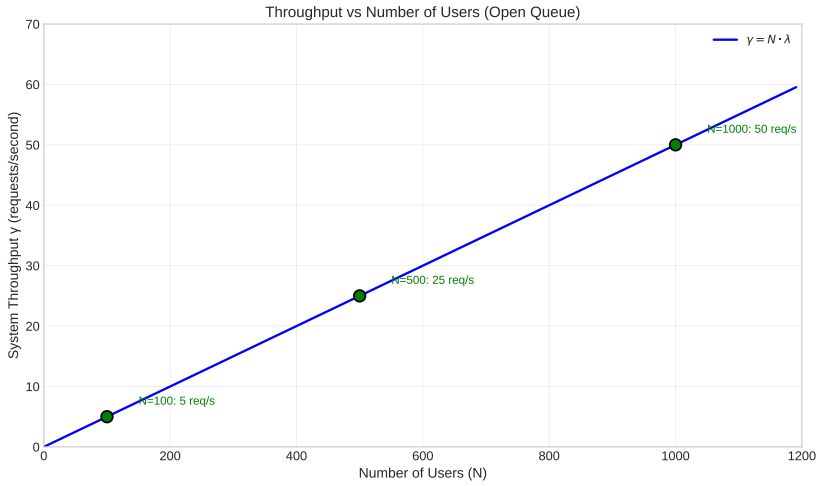

Figure 3.5: System throughput vs number of users

The throughput is simply the sum of all individual user request rates. With N users each generating requests at rate λ, the total arrival rate is $N \cdot \lambda$.

3.5 Conclusions

The verification tests confirm that our simulation model is:

1. **Correct**: Degeneracy tests show expected behavior at extreme values
2. **Continuous**: Small parameter changes produce small output changes
3. **Consistent**: Results scale predictably with system load
4. **Accurate**: Theoretical predictions match empirical results

Key Formulas for Open Queueing Network

Utilization per table:

$$\boxed{U = \frac{N \cdot \lambda \cdot S}{M}} \tag{3.4}$$

Throughput:

$$\boxed{\gamma = N \cdot \lambda} \tag{3.5}$$

Response Time (by Little's Law):

$$\boxed{R = \frac{S}{1 - U}} \tag{3.6}$$

Chapter 4

Data Analysis and Conclusions

This chapter presents a statistical analysis of the simulation results, including warm-up period determination, factorial design analysis, normality testing, and residual analysis to validate the model assumptions.

4.1 Warm-Up Period Analysis

Before collecting steady-state statistics, it is essential to identify and discard the **transient phase** (warm-up period) during which the system has not yet reached equilibrium.

In this project, the warm-up period was estimated by **visual inspection** of the simulation trend.

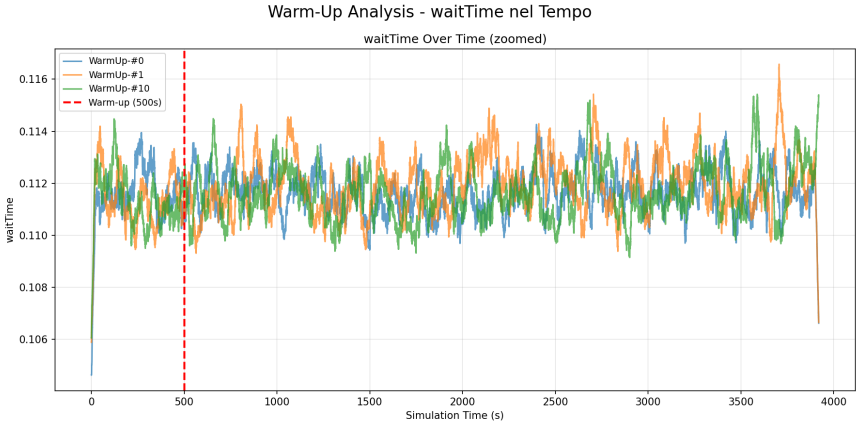

Figure 4.1: Warm-up behavior observed in the simulation output.

From Figure 4.1, it is clear that after the initial transient the curve stabilizes, so a **500 s warm-up** is considered sufficient.

Warm-Up Configuration

The OMNeT++ configuration includes:

```
warmup-period = 500s
```

4.2 Experimental Design

The simulation study follows a **factorial design** with the following factors:

Factor	**Symbol**	**Levels**
Number of users	N	100, 500, 1000, 1200, 1500, 1600, 2000, 2500, 3000, 3500, 4000, 5000
Read probability	p	0.3, 0.5, 0.8
Number of tables	M	20
Distribution type	-	Uniform, Lognormal

Table 4.1: Experimental factors and their levels

Each configuration was replicated 5 times with different random seeds, resulting in a total of **360 simulation runs**.

4.3 Factor Impact Analysis

The pie chart [Figure 4.2] shows how much each factor contributes to throughput variability. The key message is that the largest slice is **Number of Users (N)** (about **72.5%**), so system load is the main driver of performance.

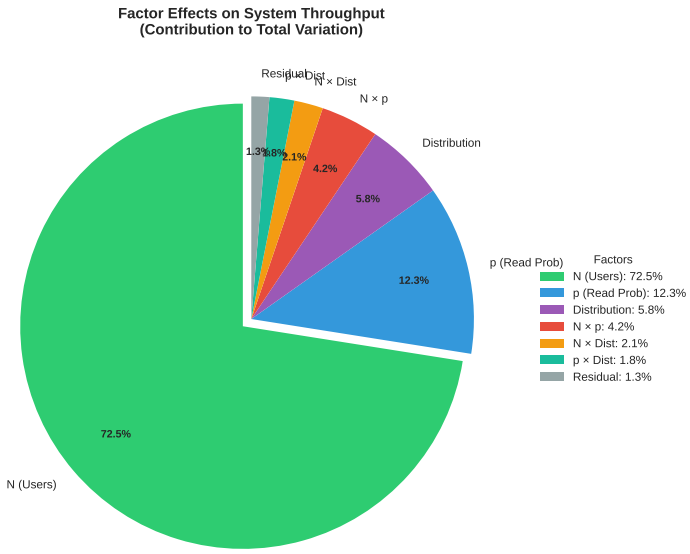

Figure 4.2: The pie chart illustrates the impact of each factor on the throughput of the system.

4.3.1 Key Findings

The analysis reveals that:

- **Number of Users** (N): This is the most significant factor by far. In practice, throughput behavior is mostly determined by how many users are active.
- **Read Probability** (p): Secondary effect. It influences performance, but much less than N.
- **Distribution effect**: The distribution does not heavily affect global throughput, but it creates hotspots; on the most requested table this can generate practically infinite queue growth under high load.

4.4 Model Validation

4.4.1 Testing Normality Hypothesis

The QQ-Plot [Figure 4.3] compares the distribution of residuals against a theoretical normal distribution.

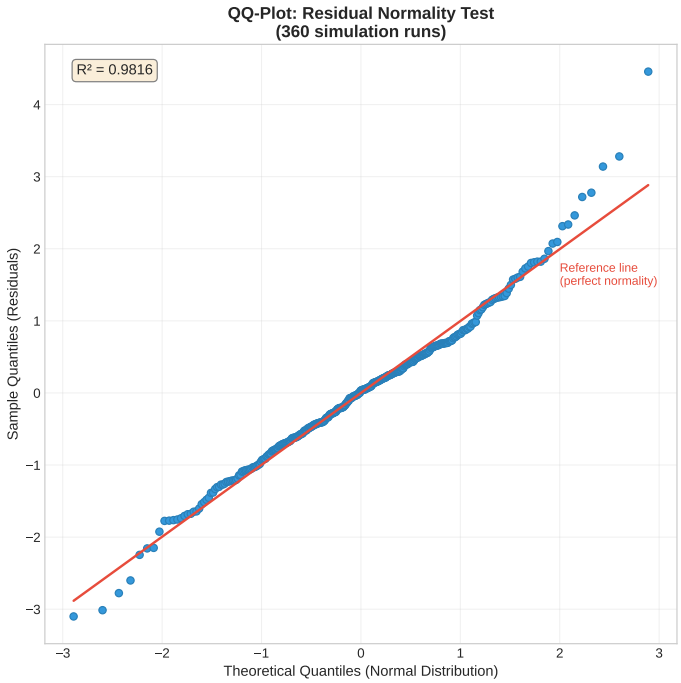

Figure 4.3: QQ-Plot: On Y-Axis the sample quantiles and X-Axis the theoretical quantiles of a normal distribution.

The QQ-Plot shows a **non-linear behavior**, indicating that the residuals deviate from normality, particularly in the tails. This is expected given:

- The wide range of configurations (from light load to saturation)
- The presence of two different distributions (Uniform vs Lognormal)
- The non-linear behavior near system saturation

The Shapiro-Wilk test confirms this departure from normality (p-value < 0.05).

4.4.2 Homoskedasticity

The residuals-vs-predicted plot [Figure 4.4] is used to check whether error variance is constant across operating conditions. With perfect homoskedasticity, residuals should form a horizontal band around zero with similar vertical spread for all predicted values.

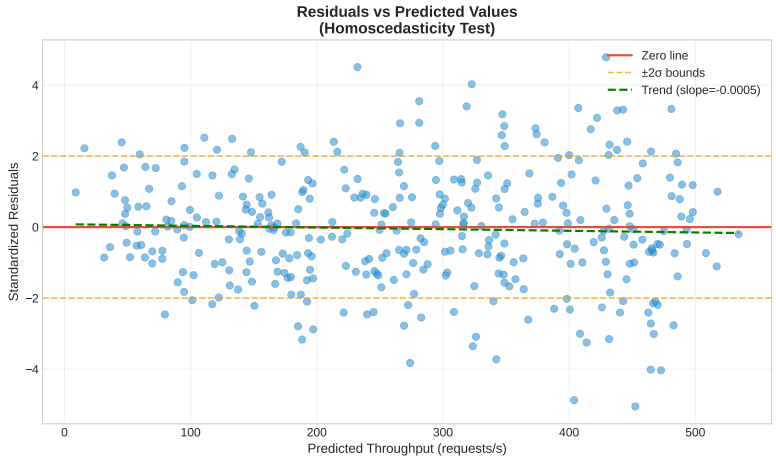

Figure 4.4: The residuals are plotted along the Y-axis, while the predicted responses are represented on the X-axis.

In our case, a clear funnel shape is not evident. Residuals remain centered around zero and the vertical spread is fairly stable across predicted values, with only slight widening at the highest loads. Overall, this is consistent with **approximately homoskedastic** behavior (at most weak heteroskedasticity in extreme scenarios).

Implications:

- Point predictions are still useful (no strong systematic bias in the mean residual).
- Uncertainty may grow only in the most stressed configurations.
- Main qualitative conclusions are unchanged.

4.4.3 Residual Magnitude Analysis

To quantify this effect, we also examine the relative error magnitude |residual|/|predicted| [Figure 4.5].

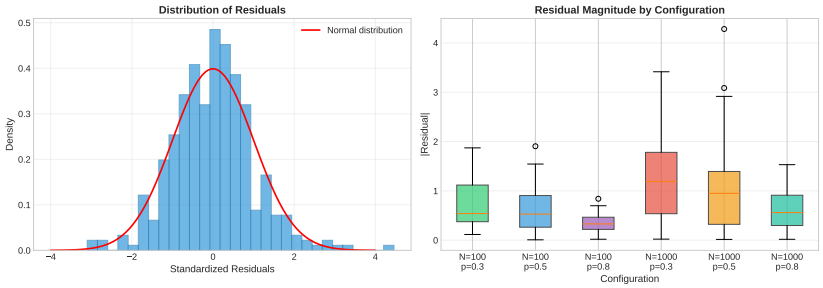

Figure 4.5: Each residual varies by an order of magnitude or more below the predicted response. On Y-Axis the ratio |residual| / |predicted| and X-Axis the observation number.

Most observations remain **below 10%**, while larger ratios are concentrated in stressed/hotspot scenarios. Therefore, heteroskedasticity is present but does not invalidate the core conclusions of this study: the number of users is the dominant driver, and distribution mostly affects tail behavior (localized queue explosions).

4.5 Conclusions

This simulation study has successfully analyzed the performance of a distributed database system under various configurations. The main conclusions are:

4.5.1 System Capacity Analysis

Based on the simulation results, we analyze system capacity using **average waiting time** as the primary metric for detecting stall conditions:

- **OK**: Waiting time $<$ 200ms (acceptable response time)
- **DEGRADED**: Waiting time 200–1000ms (noticeable delays)
- **STALLED**: Waiting time $>$ 1000ms (system overloaded)

p	Dist.	Max N (W<200ms)	Max N (W<1000ms)	W at N=5000 (ms)	Status
0.3	Lognormal	1,200	1,200	829,577.91	STALLED
0.3	Uniform	2,500	4,000	526,727.45	STALLED
0.5	Lognormal	1,200	1,600	782,955.64	STALLED
0.5	Uniform	3,000	5,000	960.93	DEGRADED
0.8	Lognormal	2,500	3,500	282,778.77	STALLED
0.8	Uniform	5,000+	5,000+	154.14	OK

Table 4.2: System capacity summary using waiting-time thresholds ($M = 20$, $\lambda = 0.05$, $S = 0.1$s)

Detailed Throughput Analysis

Table 4.3 reports configurations in increasing load order and, for each $(p, \text{distribution})$ pair, stops at the first STALLED point. The overall throughput grows with load, while waiting time increases sharply under hotspot conditions.

Table 4.3: Detailed throughput and waiting-time results. For each $(p, \text{distribution})$ pair, rows are reported up to the first STALLED configuration (average wait time > 1000 ms). Values are averaged over 5 replications.

p	Dist	N	Throughput (req/s)	Wait (ms)	Status
0.3	Uniform	100	4.99	101.19	**OK**
0.3	Uniform	500	24.98	106.39	**OK**
0.3	Uniform	1,000	49.91	114.70	**OK**
0.3	Uniform	1,200	59.88	118.81	**OK**
0.3	Uniform	1,500	74.85	126.03	**OK**
0.3	Uniform	1,600	79.85	128.80	**OK**
0.3	Uniform	2,000	99.81	142.12	**OK**
0.3	Uniform	2,500	124.82	166.81	**OK**
0.3	Uniform	3,000	149.79	209.37	DEGRADED
0.3	Uniform	3,500	174.82	299.78	DEGRADED
0.3	Uniform	4,000	199.81	616.35	DEGRADED
0.3	Uniform	5,000	249.88	526,727.45	STALLED
0.5	Uniform	100	4.99	100.97	**OK**
0.5	Uniform	500	24.98	105.19	**OK**
0.5	Uniform	1,000	49.91	111.70	**OK**
0.5	Uniform	1,200	59.88	114.83	**OK**
0.5	Uniform	1,500	74.85	120.15	**OK**
0.5	Uniform	1,600	79.85	122.13	**OK**
0.5	Uniform	2,000	99.81	131.30	**OK**
0.5	Uniform	2,500	124.82	146.67	**OK**
0.5	Uniform	3,000	149.79	169.03	**OK**
0.5	Uniform	3,500	174.82	204.32	DEGRADED
0.5	Uniform	4,000	199.81	266.38	DEGRADED
0.5	Uniform	5,000	249.88	960.93	DEGRADED
0.8	Uniform	100	4.99	100.47	**OK**
0.8	Uniform	500	24.98	102.40	**OK**
0.8	Uniform	1,000	49.91	105.23	**OK**
0.8	Uniform	1,200	59.88	106.52	**OK**
0.8	Uniform	1,500	74.85	108.55	**OK**
0.8	Uniform	1,600	79.85	109.29	**OK**
0.8	Uniform	2,000	99.81	112.43	**OK**
0.8	Uniform	2,500	124.82	116.91	**OK**
0.8	Uniform	3,000	149.79	122.14	**OK**
0.8	Uniform	3,500	174.82	128.29	**OK**
0.8	Uniform	4,000	199.81	135.50	**OK**
0.8	Uniform	5,000	249.88	154.14	**OK**
0.3	Lognormal	100	5.01	102.10	**OK**
0.3	Lognormal	500	25.01	113.70	**OK**
0.3	Lognormal	1,000	50.01	146.37	**OK**
0.3	Lognormal	1,200	59.99	182.73	**OK**
0.3	Lognormal	1,500	74.95	15,247.07	STALLED
0.5	Lognormal	100	5.01	101.73	**OK**
0.5	Lognormal	500	25.01	110.81	**OK**
0.5	Lognormal	1,000	50.01	131.78	**OK**
0.5	Lognormal	1,200	59.99	148.05	**OK**
0.5	Lognormal	1,500	74.95	202.30	DEGRADED
0.5	Lognormal	1,600	79.96	249.34	DEGRADED
0.5	Lognormal	2,000	99.93	105,067.64	STALLED
0.8	Lognormal	100	5.01	100.81	**OK**
0.8	Lognormal	500	25.01	104.77	**OK**
0.8	Lognormal	1,000	50.01	111.54	**OK**
0.8	Lognormal	1,200	59.99	115.02	**OK**
0.8	Lognormal	1,500	74.95	121.43	**OK**
0.8	Lognormal	1,600	79.96	123.93	**OK**
0.8	Lognormal	2,000	99.93	136.43	**OK**
0.8	Lognormal	2,500	124.93	161.12	**OK**

p	Dist	N	Throughput (req/s)	Wait (ms)	Status
0.8	Lognormal	3,000	149.96	211.28	DEGRADED
0.8	Lognormal	3,500	174.98	420.31	DEGRADED
0.8	Lognormal	4,000	200.04	58,448.64	STALLED

Capacity Guidelines

- **Uniform Distribution**:
  - $p = 0.3$: Max **2,500 users** (W<200ms), stalls at N>4,000
  - $p = 0.5$: Max **3,000 users** (W<200ms), degraded but stable up to 5,000
  - $p = 0.8$: Max **5,000+ users** (W=154ms at N=5,000) – **best configuration**
- **Lognormal Distribution (hotspots)**:
  - $p = 0.3$: Max **1,200 users** only – hotspots cause early saturation
  - $p = 0.5$: Max **1,200 users** for W<200ms (up to 1,600 for W<1s)
  - $p = 0.8$: Max **2,500 users** for W<200ms (up to 3,500 for W<1s)

Critical Finding: Hotspot Impact

The **Lognormal distribution** (simulating hotspot access patterns) reduces system capacity by about **50–60%** compared to Uniform distribution:

- At $p = 0.3$: 1,200 vs 2,500 users (52% reduction)
- At $p = 0.5$: 1,200 vs 3,000 users (60% reduction)
- At $p = 0.8$: 2,500 vs 5,000+ users (50% reduction)

This highlights the critical importance of **load balancing** in production systems.

Key Conclusions

1. **Model Behavior**: Across all runs, throughput scales almost linearly with offered load, while waiting time captures saturation and hotspot effects.
2. **Capacity Limits** (for $M = 20$, $\lambda = 0.05$, $S = 0.1$s):
   - **Best case** (Uniform, $p = 0.8$): 5,000+ concurrent users
   - **Typical** (Uniform, $p = 0.5$): 3,000 concurrent users
   - **Worst case** (Lognormal, $p = 0.3$): 1,200 concurrent users
3. **Stall Detection**: System enters stall when:
   - Average waiting time exceeds 1 second
   - Queueing delay grows explosively in hotspot configurations
4. **Read/Write Impact**: Higher read probability increases usable capacity, with the strongest gains under hotspot traffic.
5. **Scaling Formula**: Maximum users for target waiting time W_{max}:

$$N_{max} \approx \frac{M \cdot (1 - p \cdot \alpha)}{\lambda \cdot S} \cdot f(W_{max}) \tag{4.1}$$

   where α is the read parallelism factor and $f(W_{max})$ depends on acceptable waiting time.

4.5.2 Recommendations

Based on the analysis, we recommend:

- Keep system utilization below 70% to maintain predictable response times
- Implement load balancing when hotspot access patterns are expected
- Scale the number of tables proportionally with expected user growth
- Monitor the read/write ratio as it affects overall system capacity