The Eastasouth Journal of Information System and Computer Science
Vol. 2, No. 03, April 2025, pp. 183-186
ISSN: 3025-566X, DOI: 10.58812/esiscs.v2i03

Identifying and Managing Noisy Neighbors in Multi-Tenant PostgreSQL Deployments (On-Premise)

Murali Natti
Lead Database Engineer | DevOps Lead | Database Architect | Cloud Infrastructure Solutions Expert | DB Security Lead

Article Info

Article history:

Received Mar, 2025
Revised Apr, 2025
Accepted Apr, 2025

Keywords:

CPU Utilization;
Disk I/O;
Linux cgroups;
Multi-Tenancy;
Noisy Neighbors;
Performance Tuning;
PostgreSQL;
Query Optimization;
Resource Contention;
System Monitoring

ABSTRACT

In on-premise multi-tenant PostgreSQL deployments, multiple tenants share the same physical infrastructure to maximize resource utilization and reduce operational costs. However, this shared environment can give rise to significant resource contention, particularly when one tenant exhibits the "noisy neighbor" effect—where its workload consumes a disproportionate amount of CPU, memory, or disk I/O relative to others. This unbalanced resource consumption can lead to widespread performance degradation, manifesting as increased query latency, reduced throughput, and overall service instability. The present article investigates the challenges inherent in multi-tenant setups, focusing on the detection and management of noisy neighbors. It explores both native PostgreSQL monitoring techniques (such as system views and performance statistics) and external solutions including Linux control groups (cgroups) for isolating and limiting resource usage. Additionally, the article outlines best practices for proactive monitoring, query optimization, and resource allocation to ensure a balanced and efficient multi-tenant environment. By providing a comprehensive framework for understanding and mitigating the noisy neighbor phenomenon, this work aims to equip database administrators and system architects with effective strategies for maintaining robust performance, even under heavy and unevenly distributed workloads.

Corresponding Author:

Name: Murali Natti
Institution: Lead Database Engineer | DevOps Lead | Database Architect | Cloud Infrastructure Solutions Expert | DB Security Lead
Email: murali.natti@gmail.com

1. INTRODUCTION

Multi-tenancy in PostgreSQL has emerged as a popular architecture for hosting multiple independent clients or applications within a single instance, primarily due to its ability to reduce operational overhead and lower costs [1]. In this model, tenants can be isolated in various ways: one common approach involves placing each tenant in its own schema within a single database, while another strategy assigns each tenant a separate database, yet all share the same physical server infrastructure [2]. This flexibility allows organizations to maximize resource utilization and streamline administrative processes, as they do not need to maintain entirely separate hardware environments for each client. However, while multi-tenancy [3] offers significant economic and operational advantages, it also inherently introduces the risk of resource contention. Because all tenants draw from the same pool of resources—such as CPU, memory, and disk I/O—any imbalance in workload distribution can lead to performance degradation. When one tenant, due to inefficient query execution, misconfigured workloads, or unexpected usage spikes, consumes a disproportionate share of the available resources, it can adversely affect the performance of other tenants sharing the same infrastructure. This phenomenon is commonly known as the "noisy neighbor" problem. The noisy neighbor not only slows down the processing of queries but can also lead to increased latency and reduced throughput for the entire system. This article delves into the various challenges that arise from resource contention in multi-tenant PostgreSQL deployments. It discusses the underlying causes of the noisy neighbor effect and provides comprehensive guidance on how to identify such issues through effective monitoring techniques, including PostgreSQL's built-in system views and external resource management tools. Furthermore, it outlines practical strategies for managing and mitigating the impact of noisy neighbors. By applying these approaches, database administrators and system architects can better isolate resource-intensive workloads and maintain optimal performance across all tenants, ensuring a balanced and efficient multi-tenant environment.

2. IDENTIFYING NOISY NEIGHBORS

A noisy neighbor in a multi-tenant environment is defined as a tenant whose resource consumption disproportionately exceeds that of others, negatively impacting shared resources. This excessive usage can be attributed to various factors, including inefficient query execution, suboptimal database configurations, or sporadic workload surges. Since PostgreSQL does not natively isolate resource usage by tenant, administrators must rely on a combination of PostgreSQL's built-in views and external system monitoring tools. For instance, the system views such as pg_stat_activity and pg_stat_statements provide real-time data on active sessions and query performance metrics. By monitoring these views, administrators can pinpoint long-running or resource-intensive queries that may signal a noisy neighbor. Additionally, using Linux tools like top, htop, or ps allows for correlating PostgreSQL process IDs with CPU and memory usage. Disk I/O monitoring with tools like iostat or iotop, alongside insights from PostgreSQL metrics such as shared buffer activity and WAL generation, further aids in detecting tenants that may be overloading the system.

3. TECHNIQUES FOR MANAGING NOISY NEIGHBORS

Once identified, managing noisy neighbors requires a multi-faceted approach that combines resource tracking with proactive tuning [4]. One effective strategy involves enforcing resource limits at the system level. Although PostgreSQL itself does not support tenant-specific resource isolation [5], Linux control groups (cgroups) can be used to limit CPU and memory usage for PostgreSQL processes. By assigning processes to specific cgroups, administrators can restrict a tenant's resource consumption and mitigate the adverse effects of a noisy neighbor. Another crucial aspect of management is query optimization [6]. Often, noisy behavior is driven by inefficient queries. Using commands such as EXPLAIN ANALYZE helps in identifying bottlenecks and optimization [7] opportunities. Regular maintenance—such as vacuuming to remove dead tuples and proper indexing—also plays a vital role in ensuring that each tenant's workload runs efficiently without monopolizing resources. In addition to system-level controls and query tuning [8], third-party monitoring tools offer detailed insights into tenant-specific activities. Tools like pgBadger, Prometheus (with PostgreSQL exporters), or New Relic provide granular visibility into resource usage, enabling administrators to set up automated alerts when resource consumption exceeds predefined thresholds. Such proactive monitoring ensures that any resource imbalance is addressed promptly.

4. BEST PRACTICES FOR PREVENTION

Preventing the emergence of noisy neighbor issues before they take root is a proactive strategy that can save significant effort and maintain overall system stability. One effective method is to allocate dedicated tablespaces for individual tenants. By doing so, disk I/O operations become isolated; each tenant's data is managed separately, which reduces the chance that high I/O demands from one tenant will impact the performance of others. In addition to this isolation at the storage level, using connection pooling solutions such as PgBouncer is highly beneficial. These tools efficiently manage and distribute database connections, ensuring that no single tenant can overwhelm the server with an excessive number of simultaneous connections. Regular workload profiling is another critical practice. By routinely analyzing and reviewing tenant-specific query patterns and schema designs, administrators can identify potential performance bottlenecks early. This ongoing evaluation helps ensure that every tenant operates within predefined performance parameters and that the system maintains a balanced resource allocation across all workloads. Furthermore, implementing query timeouts is essential; this measure prevents long-running and resource-intensive queries from monopolizing system resources indefinitely, which can lead to cascading performance issues. Overall, these best practices form a comprehensive approach to preemptively mitigating noisy neighbor problems by ensuring that resource consumption is balanced and that each tenant's workload is effectively contained.

5. CONCLUSION

Effectively managing noisy neighbors in multi-tenant [9] PostgreSQL deployments is not merely a reactive task—it is a proactive strategy that underpins system performance, fairness, and overall user satisfaction. When a single tenant begins to consume disproportionate resources, it can trigger a cascade of performance issues that ripple through the shared environment, affecting query latency, throughput, and even system stability. To counteract these challenges, a multi-pronged approach is essential. By leveraging PostgreSQL's native monitoring views such as pg_stat_activity and pg_stat_statements, administrators can gain real-time insights into query behavior and resource usage. These views, when used alongside external system tools like Linux control groups (cgroups), enable a dynamic method for isolating and containing resource-heavy workloads [10]. Furthermore, the implementation of proactive measures such as dedicated tablespaces for individual tenants helps isolate disk I/O operations, ensuring that a single tenant's high demand does not interfere with the performance of others. Connection pooling solutions like PgBouncer play a vital role as well, as they help to efficiently manage and distribute database connections, thereby preventing any one tenant from overwhelming the server with excessive simultaneous requests. Rigorous query optimization practices and the enforcement of query timeouts ensure that long-running, resource-intensive queries do not hog critical system resources indefinitely. By continuously profiling workloads [10] and adjusting configurations in real time, organizations can enforce strict resource limits that maintain an equilibrium across all tenants. This proactive stance not only mitigates the immediate impacts of noisy neighbors but also lays the foundation for a scalable, high-availability system that adapts to varying load conditions. In essence, a comprehensive strategy—encompassing native monitoring, external resource management, and rigorous optimization—ensures that multi-tenant [3] PostgreSQL environments can meet the diverse performance demands of modern applications while maintaining a stable and efficient operational state for all users.

REFERENCES

[1] B. Momjian, *PostgreSQL: Introduction and Concepts*. Addison-Wesley., 2020.
[2] M. Stonebraker, *Scaling Databases for the Cloud Era*. Morgan Kaufmann., 2019.
[3] W. Shim and J. Kim, "Database Multi-Tenancy: Challenges and Approaches," *IEEE Trans. Knowl. Data Eng.*, 2018.
[4] M. Finkel, *Mastering PostgreSQL: Advanced Performance Tuning*. Packt Publishing., 2022.
[5] S. Roy and A. Gupta, *Effective Resource Isolation in Shared Database Systems*. Springer., 2021.
[6] D. Ferguson, *PostgreSQL High-Performance Optimization*. O'Reilly Media., 2021.
[7] PostgreSQL Global Development Group, "PostgreSQL Documentation: Performance Optimization," 2023. https://www.postgresql.org/docs/
[8] T. Nguyen, *Practical Techniques for Database Performance Tuning in Multi-Tenant Environments*. Elsevier., 2022.
[9] J. Peterson, *Managing Resource Contention in Multi-Tenant Systems*. ACM Digital Library., 2020.
[10] SQL Performance Blog, "Optimizing PostgreSQL Performance for High-Transaction Workloads," 2021. https://www.sqlperformance.com