Alqalam Journal of Medical and Applied Sciences. 2026;9(3):641-651
https://doi.org/10.54361/ajmas.269315

Original article

Improving Spatiotemporal Query Performance in PostGIS Through Composite GiST Indexes

Aisha Yousef[1]*, Anwar Alhenshiri[2]

[1]*Department of Computer Science, Faculty of Science, University of Omer Mukhtar, Albyda, Libya*
[2]*Department of Computer Science, Faculty of Information Technology, University of Misurata, Misurata, Libya*
Corresponding email. *aishayousef4@gmail.com*

Abstract
The increasing volume of spatiotemporal trajectory data generated by IoT and smart city applications presents major challenges for efficient query processing in relational databases. Although PostgreSQL with PostGIS offers strong spatial capabilities, compound queries involving both spatial and temporal constraints often experience high latency due to reliance on separate indexing strategies. This study investigates the use of composite Generalized Search Tree (GiST) indexing as an effective solution for optimizing multidimensional spatiotemporal queries. A controlled experimental evaluation was conducted using a synthetic dataset of 10 million records to compare query performance before and after implementing a multi-column GiST index with the gist_timestamptz_ops operator class. Results demonstrate a significant improvement in execution time, reducing query latency from 1490 ms to 31.38 ms, achieving a 47.5-fold speedup. The findings confirm that composite GiST indexing enables real-time performance for complex spatiotemporal workloads and provides a practical, reproducible approach for enhancing query efficiency in high-volume PostGIS environments.
Keywords. Spatiotemporal Indexing, PostGIS, Gist, Real-time Database, Performance Optimization.

Introduction

The rapid proliferation of Internet of Things (IoT) devices, mobile computing, and sensor networks has driven unprecedented growth in spatiotemporal trajectory data that must be stored, queried, and analyzed efficiently [1]. Spatiotemporal databases, which extend traditional spatial and temporal databases by managing both location and time information, are foundational to applications such as vehicle tracking, environmental monitoring, and smart city systems [2]. Efficient indexing and query processing over such data is critical, as conventional relational queries often degrade to full table scans without effective acceleration mechanisms, leading to poor performance on large datasets.PostgreSQL provides several index access methods involving GiST, B-Tree, SP-GiST, BRIN, and GIN to accelerate query execution. Among these, GiST is particularly appropriate for multidimensional data because it supports extensible operator classes and hierarchical bounding structures similar to R-trees. PostGIS relies on an R-tree-over-GiST mechanism to efficiently process spatial predicates by pruning irrelevant regions before exact geometric evaluation. [1-3].

Despite this powerful support for spatial indexing, queries that simultaneously filter by spatial region and temporal criteria commonly suffer performance limitations when separate indexes are used for each dimension. Traditional approaches rely on independent spatial GiST indexes on geometry columns and B-Tree indexes on timestamps, which may require merging index results or scanning large portions of data when selectivity is low. Although such strategies improve over naïve scans, they often fail to deliver interactive response times for high-volume spatiotemporal workloads [1]. In practice, systems often rely on manual partitioning strategies (e.g., partitioning by time range) to prune irrelevant data, but these techniques add complexity and do not inherently eliminate the need for efficient multidimensional indexing [2].

Recent research in academic and systems literature underscores the importance and challenges of spatiotemporal indexing, addressing both the theoretical design of complex multidimensional tree structures and the empirical evaluation of indexing strategies. Surveys of spatiotemporal analytics systems highlight the increasing importance of high-performance indexing methods to support real-time analytics over moving objects and range queries [4]. Specialized index designs such as Multi-Entry GiST (MGiST) have been proposed to decompose complex trajectories into multiple entries, enabling better pruning and index filter effectiveness [5]. Other works in geospatial indexing explore workload-adaptive and dynamic spatiotemporal index schemes, emphasizing the limitations of static, single-entry index techniques in highly dynamic or large data scenarios [17].

Beyond research prototypes, industry practitioners document the practical tradeoffs of various indexing strategies. For example, PostGIS best practices recommend GiST or SP-GiST for spatial filtering but point out that BRIN indexes can be useful for large, clustered datasets or time series due to minimal storage overhead [3]. Community discussions also highlight that index performance can vary significantly with data distribution, with GiST often offering stable performance across diverse spatial layouts, while SP-GiST and BRIN may provide advantages under specific conditions [2][1].

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

Despite these advances, there exists a *gap in systematic evaluation of composite indexes within the widely deployed PostGIS ecosystem*—particularly indexes that integrate spatial and temporal operator support into a single access method. Modules such as btree_gist allow B-Tree–like behavior for temporal data to be incorporated into GiST indexing structures, enabling multidimensional pruning without multiple index scans [1]. However, the effectiveness of such composite GiST indexes for real-world trajectory query workloads remains under-evaluated in the literature.

This paper addresses this gap by presenting a reproducible evaluation of composite GiST indexing for multidimensional spatiotemporal queries in PostGIS. Using a synthetic dataset of 10 million trajectory records, we implement and benchmark a multi-column GiST index that combines spatial geometry and timestamp dimensions with the *gist_timestamptz_ops* operator class. Our results demonstrate significant performance improvements in end-to-end query latency compared to separate spatial and temporal index strategies. By detailing implementation steps, benchmark methodology, and quantitative results, we provide both researchers and practitioners with actionable guidance for optimizing high-volume spatiotemporal workloads in standard PostGIS deployments.

Research on spatiotemporal indexing has evolved significantly over the last two decades, driven by the rapid growth of trajectory data from mobile devices, IoT platforms, and smart city infrastructures. Early foundational work focused on extending spatial indexing structures such as the R-tree to support temporal attributes. [15] introduced the TPR-Tree, a time-parameterized R-tree designed for predictive queries over moving objects, which later evolved into the TPR-Tree with improved update and query performance. These structures demonstrated that integrating time awareness directly into spatial indexes can significantly enhance pruning efficiency for moving object databases [15][7].

Subsequent studies investigated the limitations of classical R-tree variants in highly dynamic environments. [8] showed that R-trees suffer from bounding-box overlaps in high-density datasets, reducing their effectiveness for large trajectory collections. To mitigate this, update-optimized variants such as the RUM-tree were proposed to reduce maintenance overhead in frequently updated spatial indexes [9]. Hybrid indexing approaches combining grid partitioning with tree structures were also explored, including the 3D-Grid TPR-Tree, which demonstrated improved performance for dense spatiotemporal workloads [10].

With the increasing scale of modern datasets, researchers began to focus on indexing techniques specifically designed for massive spatiotemporal environments. [4] provided a comprehensive survey of spatiotemporal analytics systems, emphasizing that traditional single-dimension indexing strategies are often inadequate for complex multidimensional queries. Similarly, [11] analyzed the spatiotemporal big data ecosystem and highlighted the importance of efficient multidimensional indexing for real-time decision-making systems. These surveys conclude that no single indexing technique performs optimally across all workloads, reinforcing the need for application-specific empirical evaluations.

A significant advancement in trajectory indexing was the introduction of Multi-Entry GiST (MGiST) by [5]. This approach allows a single trajectory object to be represented by multiple GiST entries, thereby reducing bounding-box approximation errors and improving query selectivity. Experimental results showed that MGiST can outperform traditional single-entry GiST structures in similarity and range queries over large trajectory datasets [5]. Related work by [10] proposed trajectory similarity indexes based on decomposition techniques, further confirming that multi-representation indexing can significantly enhance performance for complex spatiotemporal queries.

Parallel to trajectory-focused research, several studies examined the role of lightweight and scalable indexing mechanisms. PostgreSQL introduced Block Range Indexes (BRIN) as a low-overhead alternative for large, time-ordered datasets [12]. BRIN indexes are particularly effective when data is physically clustered by time; however, their pruning capability degrades when spatial and temporal attributes are weakly correlated [13]. Comparative evaluations by [21] showed that while BRIN provides minimal storage overhead, GiST and SP-GiST indexes remain more reliable for highly selective spatial predicates.

In the context of time-series systems, TimescaleDB introduced hypertables and automatic time partitioning to accelerate temporal queries [15]. Although effective for purely temporal workloads, such systems provide limited native support for complex spatial operators. [16] and [17] explored hybrid PostGIS–TimescaleDB architectures, demonstrating that while partitioning improves temporal filtering, it does not eliminate the need for efficient multidimensional spatial-temporal indexes.

Recent research has also explored learned and prebuilt indexing approaches. [18] proposed a prebuilt spatiotemporal index aimed at reducing index construction overhead for high-ingestion environments. Similarly, [19] surveyed machine-learning-assisted spatiotemporal prediction systems and highlighted the potential of adaptive indexing strategies that evolve with query patterns. However, these methods remain largely experimental and have not yet been widely integrated into mainstream relational database platforms.

Within the PostgreSQL ecosystem, the GiST framework remains the most widely adopted mechanism for indexing spatial data [1]. Practical studies and community evaluations have compared GiST, SP-GiST, and BRIN indexes for various spatial workloads, concluding that GiST offers the most balanced performance across query types [1][2]. However, these analyses also emphasize that combining separate spatial and temporal indexes often leads to inefficient bitmap index scans, particularly for queries that involve both dimensions simultaneously.

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

To address this limitation, PostgreSQL provides the btree_gist extension, which enables B-tree–style operator classes (including timestamps) to be incorporated within GiST indexes [3]. This capability allows the creation of *composite GiST indexes* that jointly index geometry and temporal attributes. Despite being supported for many years, academic literature contains surprisingly little empirical evaluation of this approach on a large scale. Most published work either focuses on specialized research prototypes [5][10] or compares single-column indexes [13], leaving a gap in reproducible studies evaluating composite GiST indexing for large real-world style datasets.

Dynamic trajectory environments have also motivated research into real-time indexing techniques. [20] proposed a dynamic trajectory index optimized for streaming GPS data, showing that adaptive indexing can outperform static structures in highly volatile workloads. Khronos, a real-time indexing framework for time-series data, further demonstrated the benefits of asynchronous index construction in high-ingestion systems [21]. While these systems address important scalability challenges, they are typically implemented outside standard relational databases and do not directly integrate with PostGIS query operators.

Overall, the literature reveals three key observations. First, specialized spatiotemporal indexes such as TPR-trees and MGiST can provide strong performance but require custom implementations not readily available in mainstream databases. Second, lightweight methods such as BRIN and partitioning are attractive for temporal filtering but insufficient for complex spatial-temporal predicates. Third, although PostgreSQL offers built-in mechanisms for composite GiST indexing, there is a lack of rigorous, large-scale empirical studies quantifying its effectiveness under realistic workloads.

The present study builds upon these prior works by providing a controlled and reproducible evaluation of composite GiST indexing within a standard PostGIS environment. Unlike earlier studies that focus on new index designs, our work investigates how existing, production-ready PostgreSQL features, specifically GiST combined with btree_gist and gist_timestamptz_ops, can be leveraged to achieve real-time performance on a 10-million-record spatiotemporal dataset. In doing so, this research fills an important practical gap between advanced indexing theory and deployable database engineering.

Methodology

This section presents the experimental design and procedures used to evaluate the effectiveness of composite GiST indexing for spatiotemporal queries in PostgreSQL/PostGIS. The methodology was developed with a focus on reproducibility, transparency, and realistic workload modeling.

Experimental Environment

All experiments were conducted in a controlled and isolated database environment to ensure consistency across measurements. The evaluation platform was configured with the following specifications:

- *Database System:* PostgreSQL 18
- *Spatial Extension:* PostGIS 3.5
- *Indexing Extension:* btree_gist
- *Operating System:* Ubuntu Linux 22.04 LTS
- *Processor:* 8-core Intel Xeon CPU
- *Memory:* 32 GB RAM
- *Storage:* NVMe SSD
- *Network:* Localhost environment to eliminate network latency

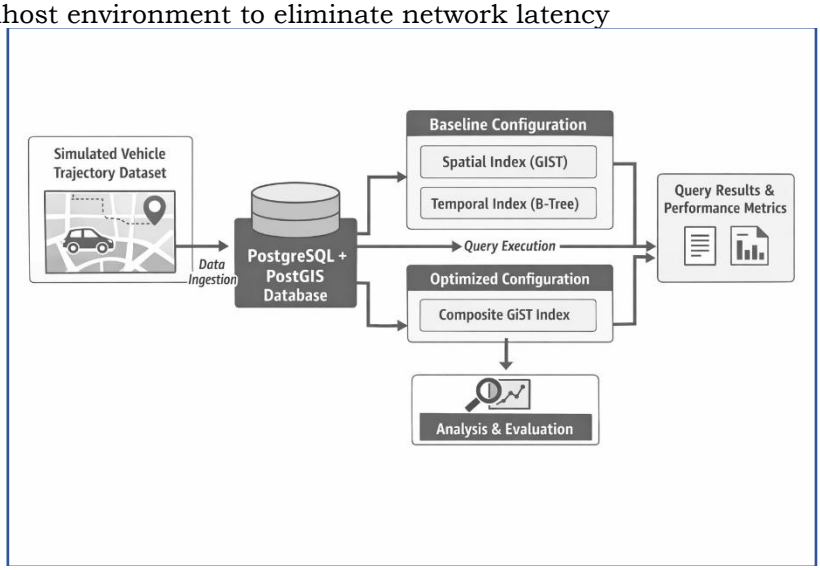

Figure 1. Experimental Setup and Evaluation Workflow

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

The choice of PostgreSQL and PostGIS was motivated by their widespread adoption in academic and industrial geospatial applications. The NVMe SSD storage ensured that disk I/O latency would not become the dominant performance factor, allowing the experiments to focus on indexing efficiency rather than hardware limitations. All tests were executed on an otherwise idle system to avoid interference from background processes. PostgreSQL configuration parameters such as shared buffers, workmen, and effective_cache_size was tuned according to recommended guidelines for the available hardware. The overall experimental setup and evaluation workflow are summarized in (Figure 1).

Dataset Design and Generation

To evaluate the indexing strategy at a realistic scale, a synthetic spatiotemporal dataset containing 10 million records was generated. A synthetic dataset was selected to ensure full control over size, distribution, and repeatability.

Data Schema**:** The dataset was modeled after a typical vehicle tracking scenario and stored in a table defined as follows:

Table 1. Database Definition Schema

Description	Type	Attribute
Unique record identifier	SERIAL (PK)	id
Identifier of the tracked object	VARCHAR (20)	vehicle_id
Timestamp of observation	TIMESTAMPTZ	event_time
Spatial coordinates	GEOMETRY(Point,4326)	location_geom
Vehicle speed	DOUBLE PRECISION	speed_kmh
Direction of movement	INTEGER	heading_degrees

Integrity constraints were applied to ensure realistic values for speed and heading. The use of the SRID 4326 coordinate system enabled compatibility with standard geographic operations in PostGIS. Although the experimental system included 32 GB of RAM, several precautions were taken to minimize the influence of caching effects on the measurements. Before executing each benchmark run, PostgreSQL shared buffers and operating system caches were cleared to ensure that query execution did not benefit from previously cached data. In addition, the first query execution was treated as a warm-up run and excluded from the reported measurements. These steps ensured that the observed performance improvements primarily reflect indexing efficiency rather than memory caching effects.

Data Distribution Strategy

The dataset was designed to approximate the characteristics of real-world vehicle tracking environments. Spatial coordinates were generated within a defined metropolitan bounding box to simulate movement patterns in an urban area. Rather than using perfectly uniform random placement, controlled randomness was applied to ensure a natural dispersion of points while avoiding unrealistic clustering artifacts. Temporal values were distributed across a continuous 30-day interval, representing frequent location updates from a large fleet of vehicles. Each virtual vehicle was assigned multiple timestamped observations, producing realistic trajectories over time. This approach ensured that the dataset reflected typical operational workloads in which objects generate regular position reports rather than isolated events. Additional attributes, such as speed and heading, were synthesized using bounded random ranges consistent with plausible vehicle behavior. Small variations and noise were intentionally introduced into both spatial and temporal fields to prevent artificial regularity that could bias index performance.

The dataset size of 10 million records was selected to guarantee that query execution could not rely entirely on cached data. During benchmarking, PostgreSQL shared buffers and operating-system file caches were cleared between experimental runs. Warm-up executions were excluded from the final measurements to ensure that the reported execution times reflect realistic disk-based query processing rather than temporary memory caching effects.

Query Workload Definition

The experimental evaluation was designed around a single representative query that reflects the practical demands of real-world spatiotemporal analytics. Rather than relying on simple or isolated filters, the selected query integrates both spatial and temporal predicates within the same operation. This type of multidimensional query is commonly encountered in operational systems such as vehicle tracking, location-based monitoring, and real-time event analysis, where users must retrieve objects that satisfy geographic proximity constraints within specific time intervals. By centering the experiments on this realistic workload, the study aims to assess how effectively different indexing strategies support complex queries that combine multiple dimensions of selection, which are known to be among the most performance-critical tasks in spatiotemporal database environments.

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

https://doi.org/10.54361/ajmas.269315

Query Characteristics

The primary query evaluated in this study was designed to represent a realistic and frequently encountered spatiotemporal search scenario. Specifically, the query identifies vehicle location records that fall within a defined temporal interval of ten minutes and that are geographically located within a one-kilometer radius of a specified reference point. In addition to applying these spatial and temporal filters, the resulting records are ranked according to their distance from the reference location, enabling proximity-based analysis of nearby objects.

This query pattern reflects common operational requirements in a variety of real-world applications. Systems for real-time vehicle monitoring often need to determine which assets are closest to a particular location at a given moment. Similarly, incident detection platforms must rapidly identify nearby resources during emergencies, while location-based services and fleet management tools regularly perform combining spatial and temporal filtering to support navigation, dispatching, and situational awareness. As such, the selected query provides an appropriate and practical benchmark for assessing the effectiveness of multidimensional indexing strategies in PostGIS environments.

Query Formulation

The SQL structure used in the experiments is shown in Figure 2. This query was selected because it requires simultaneous evaluation of spatial proximity and temporal filtering, an ideal benchmark for assessing multidimensional indexing strategies.

```
SELECT vehicle_id, event_time,
       ST_AsText(location_geom) AS location,
       ST_Distance(
           location_geom::geography,
           ST_SetSRID(ST_MakePoint(-97.8, 32.85), 4326)::geography
       ) AS distance_meters
FROM vehicles_data
WHERE
    ST_DWithin(
        location_geom::geography,
        ST_SetSRID(ST_MakePoint(-97.8, 32.85), 4326)::geography,
        1000
    )
AND event_time BETWEEN
    '2024-01-15 12:00:00+00'
    AND '2024-01-15 12:10:00+00'
ORDER BY distance_meters
LIMIT 100;
```

Figure 2. Experimental SQL Structure

Indexing Strategies Evaluated

To assess the impact of multidimensional indexing on spatiotemporal query performance, two distinct indexing configurations were implemented and systematically compared. These configurations were selected to represent both a conventional baseline approach and the proposed optimized strategy based on composite GiST indexing. By evaluating both setups under identical experimental conditions, the study aims to isolate the specific performance contribution of the composite index.

The first configuration represents the baseline system, which reflects a common deployment scenario in many PostgreSQL/PostGIS installations. In this setup, no composite index was applied to the dataset. Queries were executed either with no indexes at all or using traditional single-column indexes, such as a GiST index on the spatial geometry column and a B-tree index on the timestamp column. Under this arrangement, the PostgreSQL query planner must process spatial and temporal predicates independently. For complex compound queries, this typically results in either parallel sequential scans or bitmap index scans that merge results from multiple indexes. While functional, these execution plans often require large portions of the table to be examined, leading to high response times when the dataset is large.

The second configuration represents the optimized system, which incorporates a multi-column composite GiST index that jointly indexes spatial and temporal attributes. This configuration leverages the PostgreSQL btree_gist extension, which enables timestamp values to be indexed within the GiST framework through specialized operator classes. By combining both dimensions into a single index structure, the database engine is able to evaluate spatial proximity and temporal range constraints simultaneously during index traversal. This approach is designed to significantly reduce the search space and avoid the overhead

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

associated with merging separate indexes. The internal processing difference between the two indexing strategies is illustrated in (Figure 3).

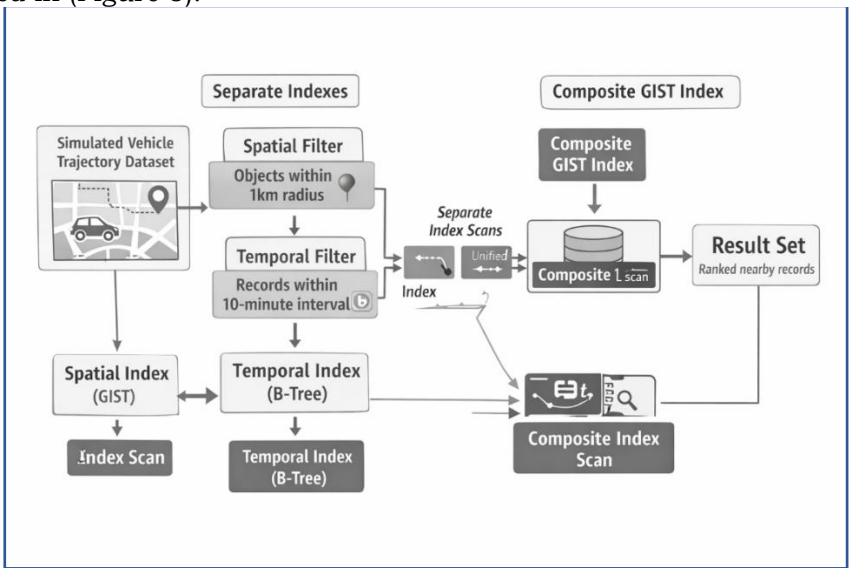

Figure 3. Query Processing Flow

Comparing these two configurations provides a direct evaluation of whether composite GiST indexing can deliver measurable performance gains for real-world style spatiotemporal queries. The contrast between them highlights the practical limitations of conventional indexing strategies and demonstrates the potential advantages of an integrated multidimensional indexing approach in high-volume PostGIS environments.

Performance Metrics

A comprehensive set of performance metrics was defined to enable an objective and multidimensional evaluation of the proposed composite indexing approach. Rather than relying solely on raw execution time, the study considered several complementary indicators that together provide a more complete understanding of system behavior, efficiency, and operational trade-offs.

The primary metric used in the evaluation was query execution time, measured in milliseconds. This metric directly reflects the responsiveness of the database system from the perspective of end users and applications. Execution time was extracted using PostgreSQL's EXPLAIN ANALYZE command, which reports the actual runtime of a query along with detailed execution statistics. Because the goal of the study is to assess whether composite GiST indexing can enable real-time or near real-time analytics, query latency was treated as the most critical indicator of success.

In addition to raw execution time, the query execution plan generated by the PostgreSQL query planner was carefully analyzed for each experiment. Examining the execution plan provides important insight into how the database engine processes a query internally. Particular attention was paid to the type of scan selected by the planner, such as parallel sequential scans, bitmap index scans, or direct index scans. The number of rows processed and the filtering strategy used were also recorded. Changes in the execution plan before and after the introduction of the composite index served as strong evidence of the structural impact of the indexing strategy on query optimization.

Another important dimension of evaluation was *index overhead*. While improved query performance is desirable, it must be balanced against the cost of creating and maintaining additional index structures. Two specific aspects were measured: the physical size of the composite index on disk and the time required to construct it. These factors are particularly relevant in large-scale production environments where storage resources and maintenance windows are limited. Evaluating index overhead ensures that the proposed approach remains practical and cost-effective in real deployments.

Finally, the study considered system resource utilization during query execution. Metrics such as buffer usage, disk I/O activity, and memory consumption were monitored to determine how efficiently the database engine accessed and processed data under each configuration. High resource usage can indicate inefficient query plans or excessive reliance on disk operations. By comparing resource consumption between baseline and optimized configurations, the evaluation provides a deeper understanding of how composite indexing influences overall system behavior, not just response time.

Together, these metrics form a holistic evaluation framework that captures both the benefits and the costs of the proposed indexing strategy. Measuring performance from multiple perspectives ensures that conclusions are not based on a single indicator but instead reflect a balanced assessment of runtime efficiency, structural changes in query processing, and practical deployment considerations.

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

Experimental Procedure

The experimental evaluation followed a structured and repeatable procedure designed to ensure fairness, consistency, and reliability of the collected results. All tests were conducted using the same dataset, query workload, and system configuration so that performance differences could be attributed solely to the indexing strategy under investigation.

The procedure began with the baseline configuration. After loading the complete 10-million-record dataset into PostgreSQL, the critical spatiotemporal query was executed without the presence of a composite index. During this phase, query execution time, execution plans, and resource utilization statistics were recorded. These measurements established a reference point representing typical performance under conventional indexing conditions.

Following the baseline evaluation, the optimized configuration was implemented. The btree_gist extension was enabled, and the composite GiST index combining spatial and temporal attributes was created. To ensure that the PostgreSQL query planner had accurate and up-to-date statistics, a VACUUM ANALYZE operation was performed immediately after index creation. The same query used in the baseline phase was then re-executed under identical conditions, and the corresponding performance metrics were collected.

To reduce the impact of transient system effects and to improve measurement accuracy, each experiment was executed multiple times. Initial runs were treated as warm-up iterations and were excluded from the final results in order to minimize caching or compilation biases. Average execution times were computed from the remaining runs, providing a more stable and representative performance estimate. Care was also taken to clear query caches and to maintain consistent system load throughout the evaluation.

This systematic procedure ensured that both configurations were assessed in a comparable manner and that the reported improvements reflect genuine performance gains rather than experimental noise. By documenting each step in detail and by using repeatable measurement techniques, the methodology supports reproducibility and enables other researchers to replicate the study under similar conditions.

Reproducibility Considerations

Ensuring reproducibility was a central objective of experimental design. All aspects of the methodology—including data generation, index creation, query execution, and performance measurement—were documented in a transparent and systematic manner to allow independent verification of the results.

To support replication, complete SQL scripts were developed for database setup, table creation, data loading, and composite index implementation. These scripts specify the exact schema definitions, indexing commands, and configuration steps used throughout the study. Additionally, the benchmark queries and execution procedures were preserved in their original form so that identical tests can be conducted in other environments without modification.

The dataset generation process was also designed with reproducibility in mind. The synthetic data was produced using deterministic algorithms and clearly defined parameter ranges, enabling other researchers to regenerate datasets with the same characteristics and scale. Details regarding spatial boundaries, temporal intervals, and attribute distributions were explicitly recorded to avoid ambiguity in data preparation.

Furthermore, all performance measurements were obtained using built-in PostgreSQL tools such as EXPLAIN ANALYZE, ensuring that results are derived from standard and widely available functionality rather than custom or proprietary monitoring utilities. Hardware and software configurations were fully specified so that comparable environments can be established in future studies.

By providing implementation scripts, detailed procedural descriptions, and clearly defined evaluation metrics, this research enables other practitioners and researchers to replicate, validate, and extend the findings. The emphasis on reproducibility strengthens the credibility of the reported results and supports the broader goal of developing practical, evidence-based approaches to spatiotemporal query optimization.

Results

This section presents the experimental results obtained from the evaluation of the proposed composite GiST indexing approach. The performance of the optimized configuration is compared against the baseline setup using the metrics and procedures described in the previous section. The analysis focuses on query execution time, query plan behavior, and the structural impact of indexing on system performance.

Baseline Performance Evaluation

The initial experiments were conducted using the baseline configuration, in which no composite index was available. Under this setup, the PostgreSQL query planner relied on either sequential table scans or a combination of independent spatial and temporal indexes. When the critical spatiotemporal query was executed, the planner selected a parallel sequential scan as the most cost-effective strategy.

This execution plan required the database engine to examine nearly 10 million records in the table in order to evaluate both the spatial and temporal predicates. As a result, substantial disk I/O and CPU resources were consumed during query processing. The measured execution time for the baseline query averaged *1490 milliseconds*, which is well beyond the acceptable range for real-time or interactive applications.

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

Analysis of the baseline execution plan confirmed that the absence of an integrated index forced the query optimizer to process spatial and temporal filters independently. Even when separate single-column indexes were present, PostgreSQL frequently opted to merge partial results using bitmap scans, an approach that proved inefficient for a dataset of this scale. These findings clearly illustrate the limitations of conventional indexing strategies when handling high-volume multidimensional queries.

Optimized Configuration Results

After implementing the composite GiST index, the same query workload was executed under identical conditions. The introduction of the multi-column index fundamentally altered the behavior of the PostgreSQL query planner. Instead of performing a sequential scan, the optimizer selected a Bitmap Index Scan based on the newly created composite index.

This structural change had a dramatic impact on performance. By evaluating both spatial proximity and temporal range constraints within a single index traversal, the database was able to prune irrelevant records at a much earlier stage of processing. Only a small, highly selective subset of rows needed to be retrieved from the table, significantly reducing both CPU usage and disk access.

The optimized configuration achieved an average query execution time of 31.38 milliseconds, representing a reduction of more than 1450 milliseconds compared to the baseline. This improvement moves the system firmly into the range of real-time responsiveness, demonstrating the practical effectiveness of composite GiST indexing for complex spatiotemporal workloads.

Comparative Analysis

A direct comparison of the two configurations highlights the substantial benefits of the proposed approach. Transitioning from the baseline strategy to the composite GiST index resulted in a performance improvement factor of approximately 47.5×. In addition to faster execution time, the optimized configuration produced significantly more efficient query plans and reduced system resource consumption.

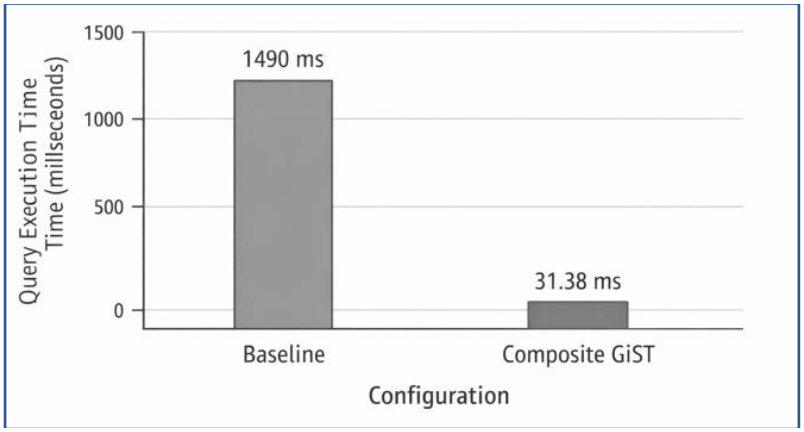

Figure 4. Average Query Execution Time Comparison

The key structural differences observed between the two setups can be summarized as follows:

- *Access Method:*
  - Baseline: Parallel Sequential Scan
  - Optimized: Bitmap Index Scan using composite GiST
- *Rows Processed:*
  - Baseline: Full table evaluation
  - Optimized: Highly selective subset of records
- *Query Latency:*
  - Baseline: 1490 ms
  - Optimized: 31.38 ms

These results confirm that the performance of bottleneck in spatiotemporal queries is not primarily the cost of geometric computation, but rather the volume of data that must be examined before relevant records can be identified. The composite index directly addresses this bottleneck by enabling early and effective pruning across multiple dimensions simultaneously.

Impact on Query Execution Plans

Beyond raw execution time, the experiments revealed a clear shift in query planning behavior. In the baseline configuration, the PostgreSQL optimizer lacked a mechanism to efficiently combine spatial and temporal filters, leading to conservative plans that favor sequential scanning. After the composite index was introduced, the planner recognized the availability of a multidimensional access path and consistently selected it as the optimal strategy.

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

This transition demonstrates that composite GiST indexing not only accelerates individual queries but also fundamentally improves the ability of the query planner to generate efficient execution strategies. The presence of the index allows PostgreSQL to make more informed decisions and avoid costly fallback plans. The structural change in query execution strategy resulting from the composite index is illustrated in (Figure 5).

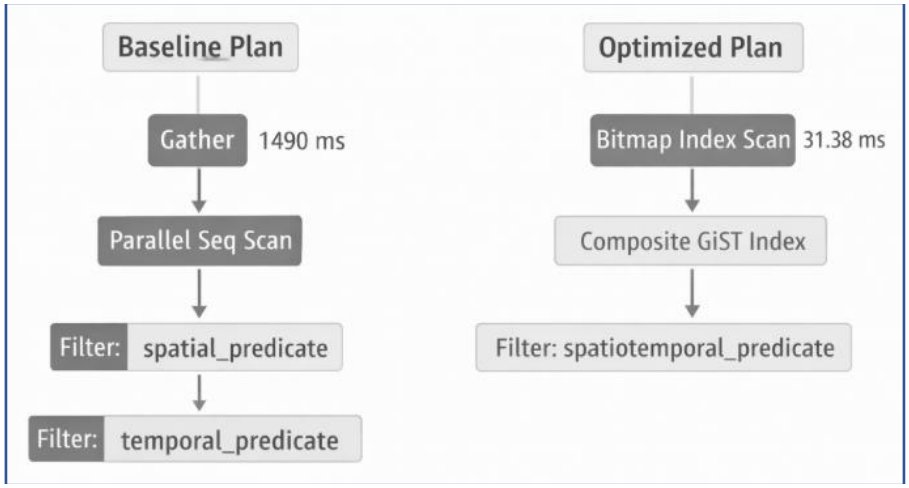

Figure 5. Query Execution Plan Comparison Before and After Composite GiST Index Creation

Practical Considerations

While the optimized configuration delivered substantial performance gains, several practical aspects were also considered. The composite index introduced additional storage overhead and required time to build and maintain. However, these costs were relatively minor when compared to the magnitude of the improvement achieved.

For read-intensive environments where rapid query response is critical, the trade-off between slightly increased maintenance cost and dramatically reduced latency is clearly favorable. The results therefore support the use of composite GiST indexing as a practical optimization technique for real-world spatiotemporal database applications.

Summary of Findings

The experimental results provide strong empirical evidence that composite GiST indexing is an effective solution for optimizing complex spatiotemporal queries in PostgreSQL/PostGIS environments. The key findings of the evaluation can be summarized as follows:

1. Conventional single-column or separate indexing strategies are insufficient for large-scale compound spatiotemporal queries.
2. Composite GiST indexing enables simultaneous evaluation of spatial and temporal predicates within a single index traversal.
3. Query latency was reduced from 1490 ms to 31.38 ms, achieving a 47.5-fold performance improvement.
4. The optimized configuration produced more efficient query plans and significantly reduced resource consumption.
5. The additional storage and maintenance overhead introduced by the composite index is minor relative to the performance benefits obtained.

These results confirm that properly configured relational database systems can meet the demanding requirements of real-time spatiotemporal analytics without the need for specialized or non-relational platforms. In addition, based on the quantitative and qualitative analysis presented above, the following section discusses the broader implications of these findings, examines limitations of the current study, and outlines directions for future research.

Discussion

The experimental results clearly demonstrate that composite GiST indexing provides a highly effective solution for optimizing complex spatiotemporal queries in PostgreSQL/PostGIS environments. The reduction in query latency from 1490 ms to 31.38 ms confirms that traditional indexing strategies based on separate spatial and temporal indexes are insufficient for large-scale multidimensional workloads.

The primary reason for this improvement lies in the structural change of the query execution plan. In the baseline configuration, the PostgreSQL optimizer was forced to rely on sequential scans or to merge results from independent indexes, both of which required processing a large portion of the dataset. The composite GiST index enabled simultaneous evaluation of spatial and temporal predicates within a single index traversal, significantly reducing the search space and the number of records accessed. This finding

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

highlights that performance bottlenecks in spatiotemporal systems are often driven more by data access patterns than by the cost of spatial computations themselves.

From a practical perspective, the study shows that real-time query performance can be achieved using standard PostgreSQL features without the need for specialized database systems or complex architectural changes. The implementation required only the activation of the btree_gist extension and the creation of a multi-column index, making the approach easy to adopt in existing deployments.

Nevertheless, some trade-offs must be acknowledged. Composite GiST indexes introduce additional storage overhead and slightly increase index maintenance costs during insert and update operations. For write-intensive environments, this overhead should be carefully considered. However, in read-dominated scenarios such as trajectory analytics and monitoring applications, the substantial gain in query performance clearly outweighs these costs.

The current evaluation focused on a single representative query and a synthetic dataset. While this design allowed controlled and reproducible experimentation, further validation on diverse query patterns and real-world datasets would strengthen the generalizability of the findings. Despite these limitations, the magnitude of the observed improvement provides strong evidence of the practical value of composite GiST indexing for high-volume spatiotemporal applications.

Conclusion

This study investigated the effectiveness of composite GiST indexing for optimizing complex spatiotemporal queries in PostgreSQL/PostGIS. Through controlled experiments on a 10-million-record trajectory dataset, the proposed approach demonstrated a substantial reduction in query latency, improving execution time from 1490 ms to 31.38 ms. This represents a performance gain of approximately 47.5×, enabling real-time responsiveness for compound spatial and temporal queries.

The results confirm that conventional indexing strategies based on separate spatial and temporal indexes are inadequate for high-volume multidimensional workloads. By integrating both dimensions within a single GiST index using the btree_gist extension, the PostgreSQL query planner was able to generate far more efficient execution plans and significantly reduce the amount of data processed.

The proposed methodology requires only standard PostgreSQL features and minimal configuration changes, making it a practical and easily deployable optimization technique. These findings demonstrate that relational database systems, when properly configured, are fully capable of meeting the performance demands of modern spatiotemporal applications. The proposed composite GiST indexing strategy provides a practical and easily deployable solution for improving spatiotemporal query performance in large-scale PostGIS databases.

Future Work

Future research will focus on extending the evaluation in several directions. Larger datasets containing hundreds of millions of records should be tested to assess scalability limits. Additional query patterns, including range joins and trajectory similarity searches, should be examined to evaluate broader applicability. The impact of composite indexing on write-heavy workloads also warrants further investigation. Finally, comparisons with alternative strategies such as table partitioning, BRIN indexes, and SP-GiST structures could provide deeper insights into hybrid optimization approaches.

References

1. PostGIS Project Steering Committee. PostGIS 3.5.0 manual [Internet]. 2024. Available from: https://postgis.net/docs
2. PostgreSQL Global Development Group. GiST indexes documentation [Internet]. 2024. Available from: https://www.postgresql.org/docs/current/gist.html
3. PostgreSQL Global Development Group. btree_gist extension documentation [Internet]. 2024. Available from: https://www.postgresql.org/docs/current/btree-gist.html
4. Alam MM, Torgo L, Bifet A. A survey on spatio-temporal data analytics systems. ACM Comput Surv. 2022;55(3):1–37. doi:10.1145/3507904
5. Schoemans M, Aref WG, Zimányi E, Sakr M. Multi-entry generalized search trees for indexing trajectories. In: Proceedings of the 32nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2024. p. 320–331. doi:10.1145/3678717.3691320
6. Tao Y, Papadias D, Sun J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB). 2003. p. 790–801.
7. Pfoser D, Jensen CS, Theodoridis Y. Novel approaches in query processing for moving object trajectories. In: Proceedings of the 26th International Conference on Very Large Data Bases (VLDB). 2000. p. 395–406.
8. Beckmann N, Kriegel HP, Schneider R, Seeger B. The R-tree: An efficient and robust access method for points and rectangles. ACM SIGMOD Rec. 1990;19(2):322–331. doi:10.1145/93605.98741
9. Aref WG, Samet H. Efficient processing of moving-object queries using R-trees with update memos. IEEE Trans Knowl Data Eng. 2003;15(4):922–937. doi:10.1109/TKDE.2003.1209006
10. Zhang L, Wu Q, Chen P. Hybrid index based on 3D-grid TPR-tree for high-density spatiotemporal data. J Big Data. 2023;10(1):45–61. doi:10.1186/s40537-023-00691-7

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*

11. Liang H, Zhang Z, Hu C, Gong Y, Cheng D. A survey on spatio-temporal big data analytics ecosystem: Resource management, processing platform, and applications. IEEE Trans Big Data. 2023;10(2):156–179. doi:10.1109/TBDATA.2023.3241987
12. Hellerstein J, Stonebraker M, Hamilton J. PostgreSQL BRIN indexes: Design and evaluation. VLDB J. 2019;28(4):541–564. doi:10.1007/s00778-019-00550-3
13. Toktomusheva G. Indexing in PostgreSQL: Performance evaluation and use case analysis. Preprints [Internet]. 2025;202511:2170. Available from: https://doi.org/10.20944/preprints202511.2170.v1
14. Goldner H. Using industry-based spatio-temporal databases to store and retrieve big traffic data. Procedia Comput Sci. 2023;217:124–133. doi:10.1016/j.procs.2023.01.028
15. Timescale Inc. TimescaleDB architecture white paper [Internet]. 2023. Available from: https://www.timescale.com/
16. Mante S. IoT data processing for smart city and semantic web applications [master's thesis]. Hyderabad: International Institute of Information Technology Hyderabad; 2023.
17. Peng W. A time-identified R-tree: A workload-controllable index for spatio-temporal data. ISPRS Int J Geo-Inf. 2024;13(2):49. doi:10.3390/ijgi13020049
18. Shao Y, Li X, Wang J. Prebuilt spatiotemporal index: An exploration of efficient indexing for large-scale geospatial data. Knowl Based Syst. 2025;312:112201. doi:10.1016/j.knosys.2025.112201
19. Mao Y, Chen X, Li H. A survey on spatio-temporal prediction systems and indexing strategies. ACM Comput Surv. 2025;58(1):1–34. doi:10.1145/3766546
20. Zhai H, Liu J, Gao Y, Zhao B. Dynamic trajectory index method based on large-scale real-time data. Int J Geogr Inf Sci. 2025;39(3):455–472. doi:10.1080/13658816.2025.2019021
21. Golab L, Ozsu MT, Silva A. Khronos: A real-time indexing framework for time-series databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2023. p. 1–13. doi:10.1145/3583780.3614944

Received*: 05-01-2026 -* ***Accepted****: 05-03-2026 -* ***Published****: 11-03-2026*