Proceedings of CONF-SEML 2026 Symposium: Computational Analysis and Modeling in Complex Intelligent Systems
DOI: 10.54254/2755-2721/2026.GU32221

# *A Performance Comparison Study of Grid Index and Quadtree Index for Large-Scale Point Data Querying*

**Xingyu Yang**

*School of Earth Sciences and Spatial Information Engineering, Hunan University of Science and Technology, Xiangtan, China*
*rockfieldyang@gmail.com*

***Abstract.*** With the rapid development of the Internet of Things (IoT) and spatial information technology, efficiently gathering and applying large-scale point data has become a critical issue, posing key challenges in fields such as Geographic Information Systems (GIS) and spatial databases. In practical application scenarios, the choice of data indexing structures largely determines query performance, particularly influencing the efficiency of range queries and nearest neighbor queries. Traditionally, the grid index and quadtree index are two common spatial index methods. The grid index enables simple and efficient data mapping by dividing space into uniform grids, though it may lead to storage redundancy and degraded query efficiency when data is non-uniformly distributed. In contrast, the quadtree index better handles uneven data through adaptive spatial subdivision, but its dynamic tree structure introduces additional computational and storage costs. Current research has largely focused on optimizing individual index structures or evaluating their performance in specific scenarios; there remains a lack of comparison between grid index and quadtree index for large-scale point data querying, particularly under multi-dimensional scenarios. The primary contribution of this work is a data-driven performance analysis that provides guidelines for selecting between a gird index and a quadtree index based on specific application requirements.

***Keywords:*** Grid index structure, Quadtree index structure, Performance comparison, Large-scale point data, Spatial index structure

## 1. Introduction

As spatial datasets continue to grow in volume and complexity within GIS applications and spatial databases, the development of efficient spatial indexing techniques has become increasingly critical to meet the demanding requirements of spatial queries [1]. Managing and querying large-scale point datasets efficiently has become a key challenge, ranging from spatial databases to real-time analysis. To cope with this challenge, a series of specialized data structures, known as spatial indexes, have been proposed to efficiently organize and retrieve spatial datasets. These indexes are diverse in type, including structures such as kd-trees, octrees/quadtrees (also referred to as "Orth-trees"), R-trees, and bounding volume hierarchies (BVHs) [2]. The performance of core spatial operations, including

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.GU32221

range query and nearest neighbor query, is fundamentally based on the underlying spatial index structure.

Among the most popular spatial index structures are the grid index and the quadtree index [3]. The grid index is efficient and easy to realize through uniform spatial partitioning, which offers fast lookups and convenient maintenance, but is known to suffer from storage inefficiency and performance degradation under highly skewed data distributions. By contrast, a quadtree index adapts dynamically to data density, providing more stable performance for unevenly distributed data. As a cost, its spatial structural complexity and update overhead are higher than those of the grid index. To conduct a systematic and comprehensive performance comparison study between the grid index and the quadtree index in the context of large-scale point data querying, the focus is strictly placed on their operational efficiency under conditions that mirror real-world challenges.

This study is designed primarily to compare the query performance and spatial distribution characteristics of the grid index and the quadtree index when data volume increases and the dataset alters. To make the results as accurate as possible, the control variable method will be employed. The experiments will vary key parameters, including dataset size and spatial distribution patterns. The performance will be measured using standardized metrics such as query response time and index build time.

This study will establish a foundational framework for future comparative studies involving more advanced or complex spatial indexes by providing an empirical analysis of the performance behaviors of the grid index and quadtree index.

## 2. Index principles and related work

This section explains the basic principles of the grid index and the quadtree index involved in this study. It also provides a brief review of correlated work, placing the study in a broader research context.

### 2.1. Principles of grid index

The grid index establishes a many-to-many mapping relationship between spatial objects and grid cells. The cardinal principle of it is to divide the study area into small grids using horizontal and vertical lines according to certain rules. This process divides the study area into discrete cells; each cell within the grid serves as a container that holds pointers to all data points whose coordinates fall within its boundary (Figure 1). A single object may span multiple cells if it crosses grid boundaries, while each cell typically contains multiple objects. This fundamental characteristic leads to inherent data redundancy and influences both storage design and query algorithms [4]. Nevertheless, the simplicity of mapping makes the index construction and point lookups of the grid index extremely fast; its time complexity is near O(1).

It is worth noting that the performance of the grid index depends on two critical factors to a large extent: The first one is cell size, which presents a fundamental trade-off. Finer grids provide higher query precision but increase storage redundancy and computational overhead. Conversely, coarser grids reduce storage costs at the expense of query accuracy. The optimal granularity depends on both the size distribution and the density of the spatial objects being indexed [4]. The second factor is spatial uniformity of data, the structure of the grid index is efficient for uniformly distributed data. Under a clustered distribution, cell overload and query degradation will be caused in dense regions, while a proliferation of empty cells results in significant memory waste, respectively.

DOI: 10.54254/2755-2721/2026.GU32221

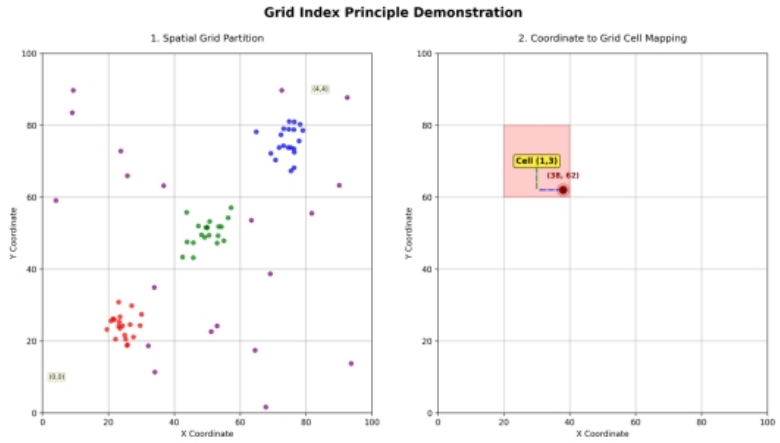


Figure 1. Grid index principle demonstration

## 2.2. Principles of quadtree index

The quadtree index, similar to the grid index, also partitions space into individual grid cells. However, instead of using a fixed resolution grid, it recursively subdivides the space into four quadrants to construct the quadtree until a predefined termination condition is met. For instance, when the number of primitives associated with each node does not exceed three. Nodes that exceed the threshold will be subdivided further, resulting in an adaptive and hierarchical tree structure (Figure 2). Each leaf node stores a list of identifiers for the primitives associated with its region, along with the region's spatial extent. In contrast, non-leaf nodes maintain only the spatial extent of their corresponding region [5].

There are several key factors that greatly determine the performance of the quadtree index: tree configuration parameters, such as maximum depth and splitting threshold, also known as maximum node capacity, which directly affect the granularity of partitions and query type, including point, range and nearest-neighbor queries, each imposing different traversal complexities.

The quadtree's adaptive subdivision mechanism automatically increases resolution in dense regions while avoiding unnecessary splits in sparse areas, which makes it particularly suitable for non-uniform spatial datasets.

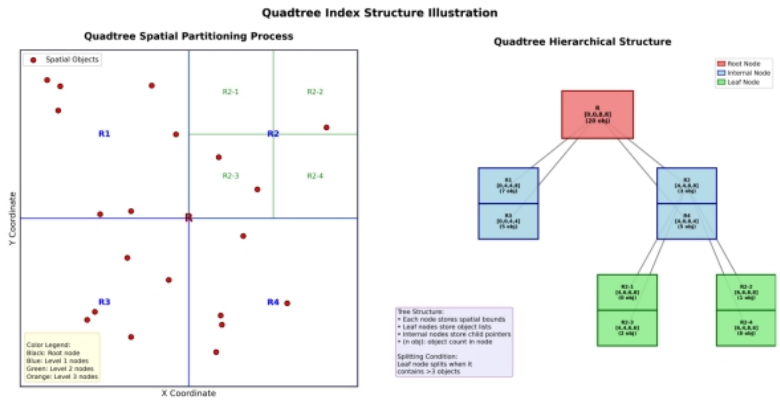


Figure 2. Quadtree index structure illustration

DOI: 10.54254/2755-2721/2026.GU32221

### 2.3. Brief review of related work

A large number of existing studies have been dedicated to the design, optimization and application of individual or hybrid indexing structures. For grid index, work has been focused on designing a query algorithm based on the grid index or its variants [6], improving the grid index generation algorithm [7] and implementing a wireless communication base station through a special grid index [8]. For quadtree index, relevant studies have explored an efficient algorithm for TIN construction using a quadtree index [9] and a quadtree-based fast clustering algorithm [10]. However, focused comparisons between the grid and quadtree structures are less common. This paper aims to fill this gap by providing a comprehensive performance comparison that varies data size, distribution pattern and query load.

## 3. Methodology

This section describes the experimental setup of this study, including datasets generation, experimental scenarios, parameter configuration and performance evaluation metrics.

### 3.1. Data and test scenarios

Multiple sets of synthetic point datasets with different characteristics were randomly generated and employed to systematically evaluate the performance of the two indexes used in this experiment. The results were averaged across multiple experiments to control variables, verify the robustness of the index structure, and cover typical scenarios.

The datasets are divided into two categories: uniform distribution and skewed distribution. The point counts are all 1M, 5M and 10M. The uniformly distributed points are generated randomly within the $[0, 10{,}000]^2$ unit space, aiming to compare the baseline performance of the indexes under ideal conditions. The skewed distribution places 80% of points in 20% of the space to test the robustness of the two indexes under highly non-uniform data. Data generation uses Python's NumPy library (with a fixed random seed set) to ensure the randomness of the results and the reproducibility of the experiments. The spatial load queries in the experiments are designed as range queries and K-nearest neighbor (KNN) queries. The range query involves a square window with variable side lengths, where the side length ratios are 0.1%, 1%, 5% and 10% of the data space. Each size of the region undergoes 1,000 random queries. For the KNN queries, the K values are set to 1, 10, 50 and 100, with 500 random query points for each K value. Since parameter optimization is not the main focus of this paper, the selection of these K values doesn't guarantee optimal query performance. However, both indexes use the same K values for testing to ensure fairness in comparison. Additionally, the number of queries for the two types differs, as range queries require more sample points to minimize errors caused by variations in query window size and local point density, while KNN query performance fluctuations are solely due to differences in point locations, requiring fewer queries to obtain reliable statistical estimates.

### 3.2. Implementation and parameters

The key parameters and memory management settings are based on commonly used empirical values. The grid index uses cell sizes of 50 units and 100 units, with all cells preassigned. The quadtree index has a node capacity of 50 points and a maximum depth limit of 15 levels. The memory pool block size for the quadtree index is set to 1024 nodes to reduce runtime overhead.

DOI: 10.54254/2755-2721/2026.GU32221

The performance comparison of the two indexes was conducted using C++. All experiments were executed in a unified environment: hardware consisted of an Intel Core i7-14650HX (8P+8E, 2.2-5.2GHz), 16GB DDR5 RAM @5600MHz and a 1TB SSD (NVMe). The operating system was Windows 11 24H2, and all C++ code was compiled using the Microsoft Visual C++ compiler (cl.exe) version 19.42.34435. To achieve a stable operating frequency in the experiment, the CPU had been configured with ThrottleStop 9.7, and the configured frequency remained stable around 2.2GHz. The interference of hardware dynamic frequency adjustment on index query time had been eliminated as much as possible, with a coefficient of variation of about 2.1% (verified by a C++ program).

### 3.3. Evaluation metrics

The experiment employed multi-dimensional quantitative indicators for evaluation. For data quality, this study recorded the recall rates of KNN queries and the data integrity of all query results. In terms of index performance and operational overhead, metrics including average latency, median latency, P95 latency, throughput (recorded as Queries Per Second, QPS) and index construction time were documented. Additionally, the t-value at a significance level of 0.05 was recorded to compare the performance differences between the two indexing methods under identical scenarios.

## 4. Results and discussions

This section presents and analyzes the performance comparison results.

### 4.1. Performance under uniform data distribution

Table 1 summarizes the average latency, median latency and P95 latency of range queries for the grid index (with the grid size 50 and 100, respectively) and the quadtree index under uniform data distribution. The results indicate that the query latency of the quadtree is significantly lower than that of the grid index across all data scales. The grid with a size of 100 exhibits even higher average latency, suggesting that a grid size of 50 performs slightly better in this scenario. The grid index shows a large gap between median and P95 latency, implying noticeable fluctuations in query latency, which may be attributed to cell boundary effects.

The main reason for the inferior performance of the grid index compared to the quadtree index lies in its implementation mechanism: range queries need to traverse all cells intersecting the query window, and check each point in these cells one by one to determine whether it falls within the window. This process introduces additional overhead from traversal computation. In contrast, the quadtree index only accesses nodes intersecting the query window through recursive pruning, making it more efficient.

Furthermore, the choice of grid size significantly affects performance: with a size of 50, the number of cells is large (40,000), and a query accesses approximately 4 cells; with a size of 100, the number of cells decreases (10,000), but the number of points per cell increases, leading to higher linear scanning costs.

Neither size achieves an optimal trade-off, indicating that the performance of the grid index is highly sensitive to parameter settings.

DOI: 10.54254/2755-2721/2026.GU32221

Table 1. Comparison of range query latency under uniform distribution (ms)

| Index type | Grid size | Data scale | Average latency | Median latency | P95 latency | QPS |
|---|---|---|---|---|---|---|
| Grid | 50 | 1M | 0.056 | 0.052 | 0.060 | 17857 |
| Grid | 50 | 5M | 0.227 | 0.214 | 0.271 | 4405 |
| Grid | 50 | 10M | 0.478 | 0.458 | 0.569 | 2092 |
| Grid | 100 | 1M | 0.060 | 0.058 | 0.069 | 16667 |
| Grid | 100 | 5M | 0.275 | 0.259 | 0.367 | 3636 |
| Grid | 100 | 10M | 0.572 | 0.547 | 0.730 | 1748 |
| Quadtree | – | 1M | 0.017 | 0.017 | 0.021 | 58824 |
| Quadtree | – | 5M | 0.059 | 0.059 | 0.066 | 16949 |
| Quadtree | – | 10M | 0.105 | 0.104 | 0.115 | 9524 |

Table 2 presents the average latency, median latency and P95 latency of KNN queries with different K values under the uniform distribution of 1M points. The grid index achieves the lowest latency for small K. As K increases, the performance gap between the two gradually narrows. Paired t-tests indicate that the query differences between the grid index and the quadtree index are statistically significant except for a few individual cases. The advantage of the grid index in KNN queries stems from its cell-based priority queue search algorithm, which can quickly locate neighboring cells and perform pruning. The tree structure of the quadtree introduces indirect overhead, making it slightly slower.

Table 2. KNN query latency under uniform distribution (ms, 1M points)

| K value | Index type | Average latency | Median latency | P95 latency | Grid/ Quadtree (Average) |
|---|---|---|---|---|---|
| 1 | Grid (50) | 0.015 | 0.014 | 0.025 | 0.43 |
| | Quadtree | 0.035 | 0.034 | 0.044 | |
| 10 | Grid (50) | 0.023 | 0.022 | 0.030 | 0.57 |
| | Quadtree | 0.040 | 0.038 | 0.051 | |
| 50 | Grid (50) | 0.054 | 0.053 | 0.064 | 0.73 |
| | Quadtree | 0.074 | 0.073 | 0.090 | |
| 100 | Grid (50) | 0.088 | 0.086 | 0.095 | 0.80 |
| | Quadtree | 0.110 | 0.108 | 0.128 | |

## 4.2. Performance under non-uniform data distribution

Table 3 shows that the grid index exhibits a large gap between average and median latency under a skewed distribution. This corresponds to the "hot cell" problem: cells in dense regions contain thousands of points, and when a query window covers such cells, the overhead of linear scanning increases dramatically. Both the median and P95 latency of the quadtree remain low, demonstrating that it effectively controls tail latency.

Table 3. Comparison of range query latency under skewed distribution (ms)

| Index type | Grid size | Data scale | Average latency | Median latency | P95 latency | QPS |
|---|---|---|---|---|---|---|
| Grid | 50 | 1M | 0.043 | 0.012 | 0.023 | 23256 |
| Grid | 50 | 5M | 0.229 | 0.054 | 0.064 | 4367 |
| Grid | 50 | 10M | 0.472 | 0.096 | 0.148 | 2119 |
| Grid | 100 | 1M | 0.058 | 0.013 | 0.068 | 17241 |
| Grid | 100 | 5M | 0.418 | 0.061 | 5.514 | 2392 |

DOI: 10.54254/2755-2721/2026.GU32221

Table 3. (continued)

| | | | | | | |
|---|---|---|---|---|---|---|
| Grid | 100 | 10M | 0.772 | 0.109 | 10.723 | 1295 |
| Quadtree | – | 1M | 0.015 | 0.008 | 0.015 | 66667 |
| Quadtree | – | 5M | 0.071 | 0.019 | 0.804 | 14085 |
| Quadtree | – | 10M | 0.121 | 0.030 | 1.599 | 8265 |

Table 4 presents the KNN query latency under a skewed distribution with 1M points across different K values. Notably, at K = 100, the latency of the grid index and the quadtree index is nearly identical, and the t-test shows no significant difference between them. This suggests that when data is highly clustered and a large number of neighbors are required, the search efficiency of the two indexing structures converges. The grid index remains significantly faster than the quadtree at other K values.

Table 4. KNN query latency under skewed distribution (ms, 1M points)

| K value | Index type | Average latency | Median latency | P95 latency | P value |
|---|---|---|---|---|---|
| 1 | Grid (50) | 0.017 | 0.016 | 0.030 | <0.001 |
| | Quadtree | 0.033 | 0.031 | 0.043 | |
| 10 | Grid (50) | 0.027 | 0.026 | 0.035 | <0.001 |
| | Quadtree | 0.038 | 0.035 | 0.049 | |
| 50 | Grid (50) | 0.065 | 0.062 | 0.083 | <0.001 |
| | Quadtree | 0.070 | 0.069 | 0.087 | |
| 100 | Grid (50) | 0.107 | 0.103 | 0.138 | 0.157 |
| | Quadtree | 0.106 | 0.104 | 0.124 | |

Grid index construction only requires a single traversal of all points to compute their cell indices, with a time complexity of O(n). Under uniform distribution, both grid sizes 50 and 100 are significantly faster than the quadtree. Quadtree construction involves recursive insertion and node splitting, which incurs higher overhead yet remains within an acceptable range (Table 5).

Table 5. Index build time under uniform distribution

| Grid size | Index type | Data scale | Build time (S) |
|---|---|---|---|
| 50 | Grid | 1M | 0.126 |
| 50 | Grid | 5M | 0.351 |
| 50 | Grid | 10M | 0.639 |
| - | Quadtree | 1M | 0.756 |
| - | Quadtree | 5M | 4.513 |
| - | Quadtree | 10M | 8.741 |
| 100 | Grid | 1M | 0.076 |
| 100 | Grid | 5M | 0.301 |
| 100 | Grid | 10M | 0.520 |

## 5. Conclusion

This paper compares the query performance of the grid index and quadtree index on large-scale point datasets (1M, 5M, 10M) under both uniform and skewed distributions, and the main findings are as follows: for range queries, the quadtree index significantly outperforms the grid index in all tested scenarios, with its average query latency being approximately 3-5 times lower than that of the grid index under uniform distribution and the advantage expanding to 5-8 times under skewed

DOI: 10.54254/2755-2721/2026.GU32221

distribution, as the recursive pruning mechanism of the quadtree endows it with strong robustness to data distribution while the grid index suffers from degraded tail latency caused by the "hot cell" problem in skewed distributions; for KNN queries, the grid index performs better at small K values with a latency about half that of the quadtree, and the performance gap between the two narrows as K increases.

Regarding grid size sensitivity, range query latency generally increases when the grid size increases from 50 to 100, which is more significant under skewed distribution, and although grid size 50 outperforms grid size 100 in most scenarios, neither grid size reaches the performance level of the quadtree, demonstrating that the performance of the grid index is highly dependent on parameter tuning. In terms of construction overhead, the grid index has a much shorter construction time than the quadtree, but its inferior query performance makes it more suitable for scenarios with few post-construction queries or low real-time requirements.

This study has several limitations: the data on memory usage could not be fully obtained due to recording issues, making it impossible to quantitatively analyze the storage efficiency of the indexes; the experiments were based on static datasets, failing to consider the impact of dynamic updates (insertions and deletions) on index performance; and only two grid sizes (50 and 100) were tested for the grid index, and a more refined parameter scan would help reveal the optimal value; In addition, the BIOS restrictions of the experimental platform made it impossible to fully lock the CPU multiplier and voltage, resulting in slight frequency fluctuations during the testing process. To mitigate this impact, methods such as thread affinity, high-performance power plans, and median statistics were adopted to ensure the reliability of relative performance comparisons, but the absolute performance values may deviate due to platform characteristics. Future work can be carried out in the following directions: introducing memory usage analysis to comprehensively evaluate the spatial efficiency of the indexes; testing the performance of the indexes in dynamic update scenarios to simulate incremental data in real applications; exploring hybrid index structures, combining the advantages of grid and quadtree indexes to design adaptive indexes.

## References

[1] Minjun Wu. (2006) Research on Method of Spatial Index for GIS. https: //www.cnki.net/kcms2/article
[2] Ziyang Men, Bo Huang, Yan Gu, Yihan Sun. (2026) Parallel Dynamic Spatial Indexes. https: //www.arxiv.org/pdf/2601.05347
[3] Zafaryab Rasool, Rui Zhou, Lu Chen, Chengfei Liu, Jiajie Xu. (2020) Index-Based Solutions for Efficient Density Peak Clustering. https: //www.arxiv.org/pdf/2002.03182
[4] Xiutao, C., Jianping, W. (2002) The Development of Storage Methods and Indexing Mechanisms for Spatial Vector Data. Remote Sensing Technology and Application, (04): 215-219.
[5] Zhuocheng, S. (2008) An Investigation of Several Spatial Indexing Methods in GIS. Science & Technology Association Forum, (02): 97-98.
[6] Dengyi, Z., Xiang L. (2017) A Density-Grid-Index-Based K-Nearest Neighboring Query Algorithm. Acta Electronica Sinica, 45(02): 376-383.
[7] Menglong, H., Peng, H. (2004) An improved method for grid index generation. Science for Surveying and Mapping, (02): 85-87.
[8] Aowei, Z., Huagen, C., Zhibo, W. (2020) Positioning and Implementation of Wireless Communication Base Stations Based on Road Network grid index. Geography and Geo-Information Science, 36(01): 82-86.
[9] Meixia, Z., Yanbing, W., Xiangxu, M. (2012) An Efficient Composite Algorithm for Constructing TIN based on quadtree index. Geography and Geo-Information Science, 28(02): 20-23.
[10] Lihua, L., Genlin, J. (2005) A Fast Clustering Algorithm Based on Quadtree. Journal of Computer Applications, (05): 1001-1003.