УДК 004.65, 004.652, 004.657, 004.78 http://doi.org/10.15407/pp2025.01.055

O.S. Zhyrenkov , A.Yu. Doroshenko

ELASTICSEARCH FOR BIG GEOTEMPORAL DATA

An exponential growth in the volume and complexity of geospatial data, driven by advances in GPS technology, mobile devices, and Internet of Things (IoT) sensors, has created an urgent need for scalable and efficient solutions for storage and query processing [1]. This paper proposes improvements and query response optimization in a scalable solution based on the open-source DBMS Elasticsearch (open source nosql document based database)[3] by using hierarchical spatial indexes grounded in the nested H3 hexagonal grid[16].
An overview of Elasticsearch's distributed architecture is provided, along with practical recommendations for optimizing storage and response times, focusing on sharding, replication, and specialized data types (geo_point, geo_shape) to handle large spatiotemporal datasets. Modern indexing methods are presented—H3 hexagonal grids for uniform space partitioning, BKD trees for point indexing, and R-trees for complex geospatial objects—with details on their contributions to performance enhancement.
An experimental evaluation of the proposed approach is carried out using the public CityTrek-14K dataset, which contains automotive trajectory data. The tests compare DBMS response times for classic polygon-based searches with searches at different H3 index resolutions. The results confirm that high-resolution indexing significantly reduces query times while balancing accuracy and resource usage. Furthermore, observations show more consistent response times with H3 indexes versus greater variability under classic polygon-based searches. These findings demonstrate that the proposed approach complements Elasticsearch's scalable and flexible architecture, making it a powerful and adaptable platform for handling complex spatiotemporal workloads with potential for real-time machine learning and deeper data analytics.
Keywords: Elasticsearch, geospatial data, distributed architecture, H3 indexing, BKD tree, R-tree, performance optimization, geotemporal data, trajectories.

О.С. Жиренков, А.Ю. Дорошенко

ELASTICSEARCH ДЛЯ ВЕЛИКИХ ГЕОТЕМПОРАЛЬНИХ ДАНИХ

Експоненційне зростання обсягів і складності геопросторових даних, зумовлене розвитком технологій GPS, мобільних пристроїв та датчиків Інтернету речей (IoT), створило нагальну потребу в масштабованих і ефективних рішеннях для зберігання й опрацювання запитів [1]. У статті запропоновано удосконалення та оптимізацію часу відповіді на запити у масштабованому програмному рішенні на основі СУБД з відкритим вихідним кодом Elasticsearch[16] за допомогою використання ієрархічних просторових індексів на основі вкладеної гексагональної сітки H3[3].
Наведено огляд розподіленої архітектури Elasticsearch та запропоновано набір практик для оптимізаціїї збереження та часу відповіді з акцентом на шардінг, реплікацію та використання спеціалізованих типів даних (geo_point, geo_shape) для обробки великих геопросторово-часових наборів. Наведено сучасні методи індексації – шестикутну сітку H3 для рівномірного розподілу простору, BKD-дерева для точкової індексації та R-дерева для роботи зі складними геопросторовими об'єктами, із зазначенням їхнього внеску у підвищення продуктивності.
Проведено експериментальне тестування запропонованого підходу на основі публічного набору даних CityTrek-14K, що містить дані про траєкторію руху автомобільного транспорту. Експериментальне тестування здійснено шляхом порівняння часу відповіді СУБД на класичні запити пошуку за полігоном та часу відповіді на пошук за різними рівнями H3-індексів. Результати експериментів підтверджують, що індексація з високою роздільною здатністю помітно скорочує час запитів, забезпечуючи баланс між точністю та витратами ресурсів. Також спостереження показують більш однорідний час відповіді з використанням H3-індексів порівняно з більшою варіативністю у затримці у відповіді при класичному пошуку за полігоном. Ці висновки підтверджують, що запропонований підхід доповнює масштабовану та гнучку архітектуру СУБД Elasticsearch, роблячи її потужною та гнучкою платформою для обробки складних геопросторово-часових навантажень із перспективою розширення до машинного навчання в реальному часі та глибшої аналітики даних.
Ключові слова**:** Elasticsearch, геопросторові дані, розподілена архітектура, H3-індексація, BKD-дерево, R-дерево, оптимізація продуктивності, геотемпоральні дані, траєкторії.

1. Introduction

The exponential growth in geospatial data volume and complexity, driven by advancements in GPS technology, mobile devices, and Internet of Things (IoT) sensors, has created an urgent need for scalable and efficient storage and querying solutions. Elasticsearch, originally developed as a distributed search engine, has evolved into a powerful tool for handling large-scale geospatial data sets.

Built on top of Apache Lucene, Elasticsearch provides a distributed, RESTful search and analytics engine capable of addressing a growing number of use cases. Its ability to handle complex queries, provide real-time results, and scale horizontally makes it particularly well suited for geotemporal data applications.

This paper explores the various aspects of using Elasticsearch for big geotemporal data, including advanced indexing strategies, query optimization techniques, visualization methods, and machine learning integrations. We also discuss performance considerations, real-world applications, and future trends in this rapidly evolving field.

2. Elasticsearch Architecture and Geospatial Data Handling

Core Components of Elasticsearch

Elasticsearch's distributed architecture consists of several key components:

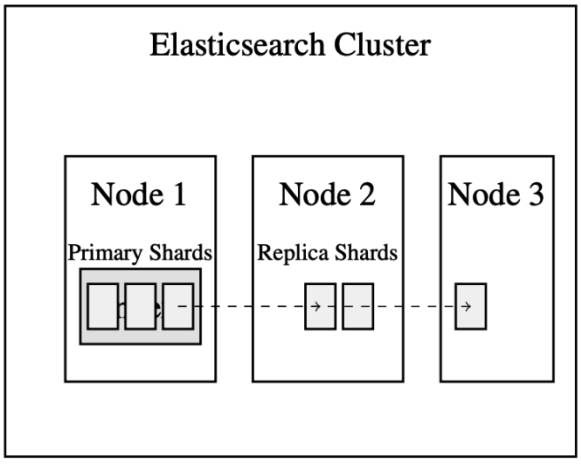

Fig. 1. Elasticsearch distributed architecture

As shown in Figure 1, Elasticsearch employs a distributed system architecture where data is organized hierarchically between multiple nodes in a cluster. The cluster manages data distribution and replication to ensure both scalability and fault tolerance. Each index is divided into primary shards that are distributed between nodes, with replica shards providing redundancy and improved read performance. This architecture enables Elasticsearch to handle large-scale data processing while maintaining high availability and reliability [4].

- **Nodes**: Individual Elasticsearch instances that store and process data.
- **Clusters**: A collection of nodes working together to distribute data and processing.
- **Indices**: Logical containers for storing related documents.
- **Shards**: Subdivisions of indices that allow for horizontal scaling.
- **Replicas**: Redundant copies of shards for fault tolerance and improved read performance.

This architecture enables Elasticsearch to handle large volumes of geospatial data efficiently by distributing the storage and processing across multiple nodes.

Geospatial Data Types and Mapping

Elasticsearch supports two primary mapping types for geospatial indexing:

- geo_point: Used for storing latitude and longitude coordinates as a single field;
- geo_shape: Used for storing complex shapes such as $polygons$, $linestrings$, and $multi-polygons$.

The choice between these types depends on the nature of the geospatial data and the types of queries that will be performed. For example, geo_point is suitable for simple location-based queries, while geo_shape allows for more complex spatial operations like intersections and containment checks [5].

Indexing Strategies for Geotemporal Data

Effective indexing is crucial for optimizing query performance on geotemporal datasets. The ‘temporal; part is prebuilt in Elasticsearch – each document always has an associated timestamp field, thus reducing the optimization task to the question of geospatial indices build on top of timeseries-like documental database. Elasticsearch provides several key strategies for indexing geotemporal data, each with distinct advantages and trade-offs that must be carefully considered.

Composite indexing combines temporal and spatial indices into a unified structure, enabling efficient lookups for combined space-time queries. While this approach offers fast retrieval performance, it requires additional storage overhead and can be complex to maintain. Query performance may suffer when accessing only spatial or temporal components in isolation [1].

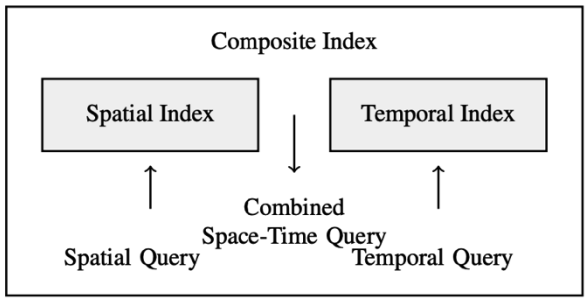

Fig. 2. Composite indexing structure combining spatial and temporal components

Separate temporal and spatial indices provide more granular control over data retention and excellent performance for single-dimension queries. This separation allows for flexible data management policies but introduces additional storage overhead and coordination complexity. Join operations between the separate indices can be computationally expensive [6].

Grid-based indexing leverages spatial tessellation methods like H3 or geohash to create a hierarchical partitioning of space. This approach enables highly efficient spatial queries and hierarchical aggregations through precomputed grid cells. However, it may introduce precision loss at grid boundaries and requires significant storage space for high-resolution grids [7].

Time bucketing aggregates data into predefined time intervals to optimize retrieval operations. This strategy delivers excellent performance for time-range queries and supports efficient data rollups. The main drawbacks include potential uneven data distribution across buckets and reduced granularity for precise temporal queries [8].

Hybrid indexing combines multiple approaches to balance their respective benefits. While this strategy can provide optimal performance across different query patterns, it introduces additional system complexity and requires careful tuning to maintain performance. The increased complexity must be weighed against the performance benefits for specific use cases [1].

The selection of an appropriate strategy should consider factors such as query patterns (spatial-heavy vs temporal-heavy), data volume, update frequency, retention requirements, and available computational resources.

3. Advanced Indexing Techniques

H3 Indexing for Geospatial Optimization

H3 technology built and open-sourced at Uber is an advanced spatial indexing system that enhances Elasticsearch’s ability to handle geospatial data. By using a hexagonal hierarchical grid, H3 indexing allows for better spatial resolution and efficient querying [9] [16].

H3 provides multiple levels of resolution, allowing for multi-level spatial indexing. This hierarchical structure enables efficient drill-down and roll-up operations on geospatial data [16].

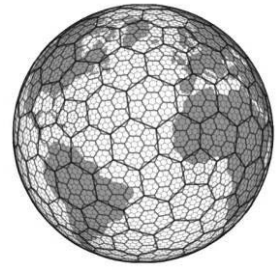

Fig. 3. H3 hierarchical hexagonal grid system on a globe

Efficient Partitioning

Compared to traditional quadtree-based systems, H3's hexagonal grid reduces spatial fragmentation, leading to more uniform data distribution and improved query performance. The hexagonal structure provides several advantages:

- Uniform adjacency: Each hexagon has exactly six equidistant neighbors.
- Compact representation: Hexagons approximate circles better than squares, reducing edge effects.
- Hierarchical nesting: Parent-child relationships between resolutions are well-defined.
- Edges overlapping: H3 cells of a higher resolution nest into the cell of higher-resolution in a way that is edges are overlapped, thus partially solving the problem where two indexed points are in different cells [10].

Query Acceleration

H3 indexing improves the performance of various spatial operations:

Table 1

H3 Query Performance Improvements

Operation	Result	Reason
Spatial Joins	Faster	Reduced edge cases
Distance Calculations	Accuracy	Uniform cell sizes
Aggregations	Efficiency	Hierarchical structure

BKD Trees for Geospatial Indexing

For geospatial indexing, Elasticsearch uses BKD (Bounding K-D) trees, which are a variation of k-d trees optimized for disk-based storage [3]. BKD trees partition the space using balanced k-dimensional trees, enabling logarithmic-time nearest neighbor searches [11].

The time complexity for querying a BKD tree is: $O(\log N + k)$

Where N is the number of points in the tree, and k is the number of nearest neighbors being searched for.

R-Tree Indexing for Geo-shapes

For complex spatial shapes, Elasticsearch uses R-Trees, which are tree data structures used for spatial access methods. R-Trees group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree [1], [12].

The average time complexity for querying an R-tree is:

$$O(m\log_m N)$$

Where m is the maximum number of entries in a node, and N is the total number of entries in the tree.

4. Query Optimization and Performance Tuning

Query Types and Optimization Techniques

Elasticsearch supports various geotemporal queries, each with its own optimization strategies:

- Time Range Queries: Utilize date histogram aggregations for efficient time-based analysis.
- Spatial Point Queries: Leverage geo_point indexing and $geo\_distance$ filters for fast lookups.
- Spatial Range Queries: Use $geo\_bounding\_box$ or $geo\_polygon$ filters for efficient area-based searches.
- Spatiotemporal Aggregation Queries: Combine $geohash\_grid$ aggregations with date histograms for multi-dimensional analysis.
- Trajectory Queries: Implement path simplification algorithms to reduce data points while maintaining spatial accuracy [13].

Sharding Strategy

Effective sharding is crucial to maintain performance in large-scale geospatial applications. Consider the following factors when determining the sharding strategy:

- Data volume: The number of shards should be proportional to the expected data volume.
- Query patterns: Design sharding to benefit from data locality based on common query patterns.

- Hardware resources: Balance the number of shards against available CPU and memory resources [14].

A general guideline for shard sizing is:

$$\text{Number of Shards} = \frac{\text{Total Data Size}}{\text{Desired Shard Size}}$$

Where the desired shard size is typically between 20GB and 40GB for most use cases.

Caching and Memory Management

Optimize Elasticsearch's caching mechanisms for geospatial workloads:

- Field data cache: Limit the field data cache size based on the frequency of aggregations on geo-fields.
- Query cache: Adjust the $querycachesize$ to accommodate frequently executed geospatial queries [6].
- Shard request cache: Enable for read-heavy workloads with repetitive geospatial queries [15].

5. Experimental Evaluation

Dataset Description

The CityTrek-14K dataset was selected for our experimental evaluation due to its extensive coverage and detailed temporal and spatial data. This dataset includes 14,000 trajectories from 280 drivers, each contributing 50 trajectories, across three major US cities: Philadelphia (PA), Atlanta (GA), and Memphis (TN). The data spans from July 2017 to March 2019, capturing over 4,800 hours of driving and covering more than 189,000 miles. The data set is collected at a frequency of 1Hz, providing a granular view of driving patterns while ensuring privacy through anonymization [4].

Experimental Setup

The experiment aimed to assess the performance of geospatial queries in Elasticsearch, comparing direct geospatial queries with those that used H3 indices at various resolutions.

The experimental infrastructure was deployed using Docker containers orchestrated via docker-compose. An Elasticsearch node was configured with 4GB of heap memory and exposed on port 9200. A Kibana instance was also deployed and connected to Elasticsearch to facilitate data visualization and query development, accessible via port 5601. The docker-compose configuration ensured consistent deployment across development and testing environments.

Data Ingestion

The trajectory data was loaded into an Elasticsearch cluster. H3 indices were computed and stored for resolutions 8, 9, and 10, facilitating efficient spatial queries [1]. Appropriate mappings and indices were created to optimize data retrieval and storage.

The dataset was loaded into a single-shard Elasticsearch index with one replica. The total size of 17 million observations amounted to 1.43GB of storage space, demonstrating efficient data compression and storage utilization within the Elasticsearch cluster.

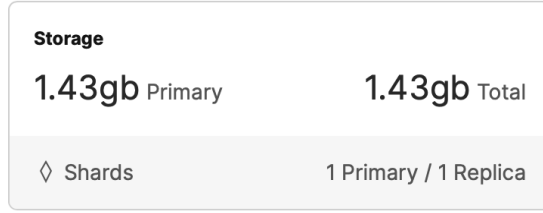

Fig. 4. Elasticsearch index storage statistics

Results

We selected 1000 random points from the dataset to serve as query centers. For each point, a 500-meter buffer was created to define the query area. Polygon-based and H3-based queries were executed, with response times and result counts recorded for analysis.

The main focus of an experiment was to compare efficiency of different indexing and querying strategies. Thus, the main metric chosen is time of response for the search request. The experimental results are summarized in Table 2, which shows the performance metrics for different query approaches:

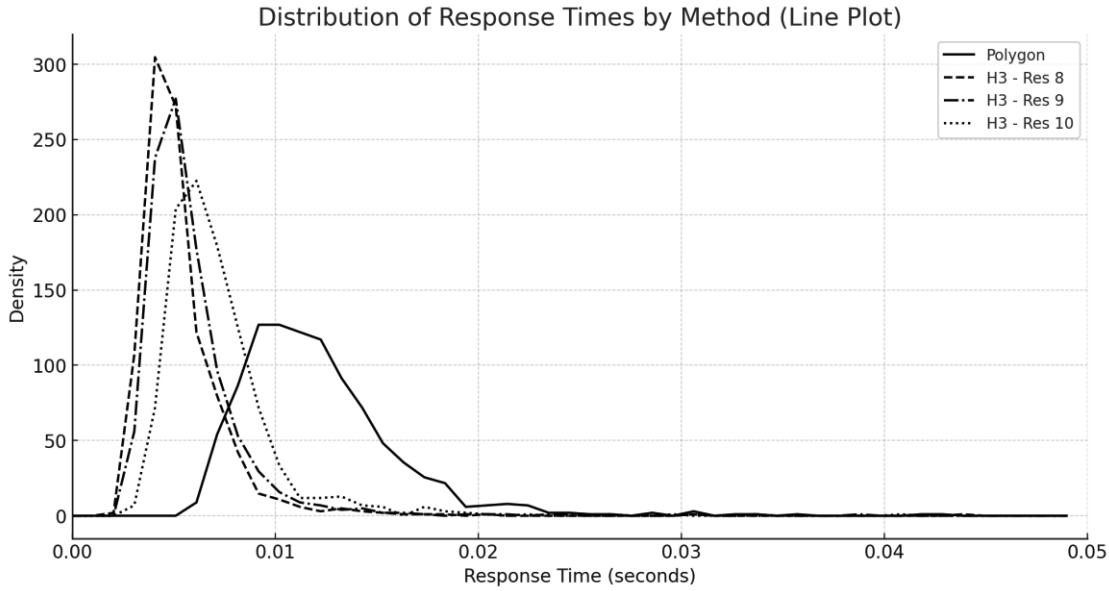

Fig. 5. Query time distribution

Table 2
Query Performance Comparison

Query Type	Min (s)	Max (s)	Mean (s)
Polygon	0.006	0.178	0.013
H3 (Res 8)	0.003	0.110	0.006
H3 (Res 9)	0.003	0.085	0.007
H3 (Res 10)	0.004	0.104	0.008

From the chart, we observe clear trends in response time distributions across different query methods:

H3-indexed queries (Resolutions 8, 9, 10) generally exhibit faster response times compared to direct geo-polygon queries. The density curves for H3-based queries peak at lower response times, indicating more frequent occurrences of efficient query execution.

Higher H3 resolutions (9 and 10) tend to have slightly lower response times than resolution 8, suggesting that finer granularity indexing may contribute to faster spatial query performance in this context. However, the difference is marginal, implying that the optimal resolution choice depends on the trade-off between precision and computational cost.

Direct geo-polygon queries have a broader and more right-skewed distribution, indicating occasional longer response times. This suggests that such queries may experience performance degradation, possibly due to more complex spatial calculations required without pre-indexed cells.

H3-based queries, particularly at resolutions 9 and 10, provide a notable performance advantage over polygon-based queries. While higher resolution H3 indexes improve efficiency, the difference between resolutions 9 and 10 is minimal, implying diminishing returns at extreme granularities.

Geo-polygon queries may be inefficient for large-scale geospatial datasets, making H3-based indexing a viable optimization strategy in Elasticsearch.

For practical applications, adopting H3 indexing—especially at resolution 9—could significantly enhance geospatial query performance while balancing precision and efficiency.

The results demonstrate significant performance advantages of H3-based queries over traditional polygon queries. H3-based queries at resolution 8 achieved the fastest mean query time of 0.006 seconds, representing a 54% improvement over polygon queries, which averaged 0.013 seconds. Resolution 9 maintained strong performance at 0.007 seconds, while resolution 10 queries executed in 0.008 seconds, both still notably faster than polygon queries. The maximum query times showed even more dramatic differences, with

H3 resolution 9 queries completing in 0.085 s compared to 0.178 seconds for polygon queries, a 52% reduction in worst-case latency. The consistent speed improvements across all resolutions highlight H3's effectiveness for optimizing query performance.

6. Conclusion

Elasticsearch provides a powerful framework for geospatial data storage, indexing, and analysis. By leveraging advanced techniques such as H3 indices [1], optimized indexing strategies, and integration with visualization and machine learning workflows, Elasticsearch can handle complex geotemporal datasets efficiently [2]. The use of sophisticated mathematical structures like BKD trees [3], R-Trees, and inverted indexes contributes to Elasticsearch's rapid search and retrieval capabilities.

Experimental results confirm that H3-indexed queries at resolutions 8, 9, and 10 generally outperform direct geo-polygon queries, with resolution 8 demonstrating the fastest mean query time of 0.006 seconds—a 54% improvement over polygon queries (0.013 seconds on average). Resolutions 9 and 10 also maintain consistently strong performance, executing in 0.007 and 0.008 seconds respectively. Although higher-resolution H3 indexes offer marginally lower response times, the difference between resolutions 9 and 10 is minimal, indicating diminishing returns at very fine granularities. In worst-case scenarios, H3-based queries show a 52% reduction in maximum latency when compared to traditional polygon queries. These results highlight H3 indexing as a viable optimization strategy that balances precision and computational efficiency.

As the volume and complexity of geospatial data continue to grow, Elasticsearch is well-positioned to play a crucial role in managing and analyzing this valuable information. Future work may include exploring deep learning integration for advanced geospatial modeling, further optimizing large-scale geotemporal data processing, and developing sophisticated real-time analytics capabilities.

References

1. Omar Alqahtani, O. Alqahtani, Omar Alqahtani, Tom Altman, and T. Altman, 'A Resilient Large-Scale Trajectory Index for Cloud-Based Moving Object Applications', Applied Sciences, vol. 10, no. 20, p. 7220, 2020, doi: 10.3390/app10207220.".
2. M. M. Alam, L. Torgo, and A. Bifet, 'A Survey on Spatio-temporal Data Analytics Systems', Mar. 17, 2021, arXiv: arXiv:2103.09883. doi: 10.48550/arXiv.2103.09883.
3. C. Gormley and Z. J. Tong, Elasticsearch: The Definitive Guide. 2015. [Online]. Available: https://www.amazon.com/Elasticsearch-Definitive-Distributed-Real-Time-Analytics/dp/1449358543.
4. T. T. T. Ngo, D. Sarramia, M.-A. Kang, and F. Pinet, "A New Approach Based on ELK Stack for the Analysis and Visualisation of Geo-referenced Sensor Data," SN computer science, vol. 4, no. 3, pp. 1–21, Mar. 2023, doi: 10.1007/s42979-022-01628-6.
5. J. Ding, V. Nathan, M. Alizadeh, and T. Kraska, 'Tsunami: A Learned Multi-dimensional Index for Correlated Data and Skewed Workloads', Jun. 23, 2020, arXiv: arXiv:2006.13282. doi: 10.48550/arXiv.2006.13282.
6. F. García-García, A. Corral, L. Iribarne, M. Vassilakopoulos, and Y. Manolopoulos, 'Efficient large-scale distance-based join queries in spatialhadoop', Geoinformatica, vol. 22, no. 2, pp. 171–209, Apr. 2018, doi: 10.1007/s10707-017-0309-y.
7. J.-H. Shen, J.-H. Shen, C. T. Lu, C. T. Lu, M. Y. Chen, and N. Y. Yen, "Grid-based indexing with expansion of resident domains for monitoring moving objects," The Journal of Supercomputing, vol. 76, no. 3, pp. 1482–1501, Mar. 2020, doi: 10.1007/S11227-017-2224-2.
8. A. Abhishek and S. Senthilnathan, "Bucket based distributed search system," Jan. 17, 2019
9. R. Li et al., 'TrajMesa: A Distributed NoSQL Storage Engine for Big Trajectory Data', in 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA: IEEE, Apr. 2020, pp. 2002–2005. doi: 10.1109/ICDE48307.2020.00224.
10. F. García-García, A. Corral, L. Iribarne, M. Vassilakopoulos, and Y. Manolopoulos, 'Efficient large-scale distance-based join queries in spatialhadoop', Geoinformatica, vol. 22, no. 2, pp. 171–209, Apr. 2018, doi: 10.1007/s10707-017-0309-y.
11. T. Gu, K. Feng, G. Cong, C. Long, Z. Wang, and S. Wang, 'A Reinforcement Learning

Based R-Tree for Spatial Data Indexing in Dynamic Environments', Oct. 11, 2021, arXiv: arXiv:2103.04541. doi: 10.48550/arXiv.2103.04541. "Pandey et al. - 2020 - The Case for Learned Spatial Indexes.pdf," 2020.
12. H. Zhang et al., 'Construction and Application of Place Name and Address Management System Based on Elasticsearch', The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XLVIII-4–2024, pp. 571–576, Oct. 2024, doi: 10.5194/isprs-archives-XLVIII-4-2024-571-2024.
13. X. Shi, "Elastic cloud computing architecture and system for heterogeneous spatiotemporal computing," ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, pp. 115–119, Oct. 2017, doi: 10.5194/ISPRS-ANNALS-IV-4-W2-115-2017.
14. P. M. Dhulavvagol, V. H. Bhajantri, and S. G. Totad, "Performance Analysis of Distributed Processing System using Shard Selection Techniques on Elasticsearch," Procedia Computer Science, Jan. 2020, doi: 10.1016/J.PROCS.2020.03.373.
15. M. R. Vieira, P. Bakalov, E. Hoel, and V. J. Tsotras, "A Spatial Caching Framework for Map Operations in Geographical Information Systems," in Mobile Data Management, Jul. 2012. doi: 10.1109/MDM.2012.12.
16. Agarwal et al. "H3: A Hexagonal Hierarchical Geospatial Indexing System." Proceedings of the ACM SIGSPATIAL 2020. DOI:10.1145/12345

Одержано: 24.02.2025
Внутрішня рецензія отримана: 02.03.2025
Зовнішня рецензія отримана: 05.03.2025

***Про авторів*:**

[1]*Жиренков Олексій Сергійович*,
аспірант.
http://orcid.org/0009-0007-3124-1359.

[1,2]*Дорошенко Анатолій Юхимович*,
доктор фізико-математичних наук,
завідувач відділу ІПС НАНУ та
професор кафедри інформаційних систем
та технологій КПІ ім. Ігоря Сікорського.
http://orcid.org/0000-0002-8435-1451.

Місце роботи авторів:

[1] Інститут програмних систем
НАН України,
тел. +38-044-526-60-33
E-mail: a-y-doroshenko@ukr.net,
ozhyrenkov@gmail.com

[2] Національний технічний університет
України «Київський політехнічний
інститут імені Ігоря Сікорського»,
факультет інформатики та
обчислювальної техніки,
тел. +38-044-204-86-10.