Insertion speed of indexed spatial data: comparing MySQL, PostgreSQL and MongoDB

Christian Axell, Eirik Schøien, Ingvild Løver Thon, Lars-Olav Vågene, Lukas Tveiten
Department of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway
{cdaxell,eirischo,ingvlt,larsolov,lukasnt}@stud.ntnu.no

***Abstract*—**
As more data is generated in the world, the notion of insertion speed is increasingly more important. Especially in areas where sensors are used, such as IoT, spatial data is accumulated in vast amounts. This research aims to determine which one of the DBMSs MySQL, PostegreSQL and MongoDB has the best performance for insertion of spatial data using indexes. The hypothesis is that with indexed spatial data, PostgreSQL and MySQL will outperform MongoDB in regards to insertion speed. This paper uses the quantitative method with an experiment to compare the insertion speed of the three DBMSs. Our experiment showed that MongoDB had significantly higher throughput than both MySQL and PostgreSQL in this scenario, particularly for small transaction sizes. The addition of spatial indexing reduced write performance for all three systems, but MongoDB with indexes still outperformed MySQL and PostgreSQL without indexes. This could indicate that MongoDB is a better choice for applications which involve heavy writing of indexed spatial data, such as logging data from IoT devices. Our results clearly disprove our original hypothesis, suggesting that either the hypothesis was false or that there were errors in the experimental setup.

***Index Terms*—insertion speed, spatial indexing, DBMS, MySQL, PostgreSQL, MongoDB**

I. Introduction

Vast amounts of data is generated every day and performing useful analysis on this data is a resource-intensive task. Finding efficient ways to handle this is a significant research area within computer science, particularly in the sub-fields of computing and *database management systems (DBMS)*. We intend to investigate problems related to handling large volumes of *spatial data*, i.e. data which references a geographical location, such as GPS coordinates.

Insertion of big data is a major performance bottleneck in data intensive systems and the data analysis process [1]. The areas of environmental monitoring, Smart Cities, Internet of Things (IoT), and GPS satellites generate large volumes of spatial data [2]. Using indexes on top of spatial data can add a significant performance penalty for writes, but is essential for supporting fast reads. Good performance for both reads and writes is essential for allowing a DBMS to support a high volume of transactions. While the number of clients generating data can be scaled arbitrarily, scaling a DBMS to receive this data can add significant cost and complexity due to the hardware required and consistency issues in distributed systems [3]. We have therefore chosen this as our application area, looking specifically at the bottleneck related to high-throughput inserts of indexed spatial data.

We are interested in evaluating insertion performance of databases that can index this type of data. MongoDB, MySQL and PostgreSQL are DBMSs of particular interest for a comparison benchmark, because they are among the most popular DBMSs in general, and they represent both document- and relational database systems.

Prior research shows that MongoDB exceeds PostgreSQL and MySQL at insertion speed of spatial data [4]. However, we have not found any research which include the use of spatial indexes. Because indexing of spatial data requires specialized data structures and indexing generally slows down insertion speed, this is an interesting topic we wish to explore further.

Hypothesis: For insertion of indexed spatial data, PostgreSQL and MySQL will outperform MongoDB.

Our contributions will give further understanding in the discussed knowledge gap; how spatial indexing affects insertion speed.

II. Background

Previous studies have looked at performance of insertion of spatial data, however only comparing to a single reference solution [5]. Additionally, the articles [6] and [7] investigates the performance of insertion in general by using parallel computing techniques and multi-threading to achieve the best insertion speed possible. However, there is a lack of studies comparing insertion speed of indexed spatial data between different DBMSs, which we find to be a knowledge gap.

One study showed that MongoDB significantly outperforms PostgreSQL and MySQL in executing insert transactions with less than 30 insert queries [4]. Still, this evaluation did not look specifically at spatial data or include the use of indexing methods, which is likely to affect the results.

Other data types are typically indexed by essentially storing a sorted order of the entries, which supports queries by key lookup and on key ranges. For queries on spatial data however, the index must also support point and window queries, i.e. finding what something is contained in or overlaps with, which requires more complex indexing methods [8]. It is also worth noting that while MongoDB uses a spatial index based on traditional B-trees [9], MySQL [10] and PostgreSQL [11] use R-trees. Conceptually B-trees and R-trees are organized in the same way, as a sorted height balanced n-ary tree where the data is stored in the leaf blocks [12]. R-trees is simply a generalization of B-tree for higher dimensional spatial data. As R-trees are a specialized data structure tailored for spatial data, it may outperform B-trees in this scenario.

III. Methods

Our hypothesis implies that there is a measurable difference in performance between MongoDB, MySQL and PostgreSQL when indexes are used. To test this we have therefore chosen to conduct an experiment where each DBMS is tested under controlled conditions and the main independent variable is whether the tested system uses a spatial index. Each system is therefore tested both without indexes, to provide a baseline for comparison, and with an index on the column with spatial data.

We could have investigated our hypothesis through other means, such as case studies or observations of real-world systems in use. However, it would be difficult to compare the different DBMSs with highly varying surrounding circumstances, which could affect the outcome. An experiment gives us more control over such variables, makes the research easier to reproduce, and takes less time to implement.

Experimental environment

All our experiments were carried out on a desktop computer, with a 12-core Ryzen 5900X CPU, 32 GB of 3600 MHz RAM and a 1 TB NVMe SSD (5000 MB/s write speed). All three DBMSs ran concurrently as Docker containers, but with limited system resources allocated to each (3 cores and 4096 MB RAM) to prevent potential resource conflicts. All three DBMSs maintain official Docker images, which makes it easier to run the experiments on a known and official configuration of the system. The experiments were then executed through a Python script, as documented in our Github repository [13].

The data set we chose for this experiment is based on real, anonymized user data [14], which consists of GPS coordinates as latitude and longtiude, as well as four other text/number columns (user ID, altitude, date and timestamp). The advantage of using a real data set over generating synthetic data is that real data is strongly correlated, meaning that sequential rows will have very similar values for time and location. This could potentially affect the performance of the indexes, in which case using correlated data is more applicable for our research as it mirrors real-world usage.

Experimental setup

We considered the following relevant parameters when designing the experimental setup:

- DBMSs to test
- Indexing methods
- Transaction size: number of records per transaction
- Experiment size: number of records per experiment
- Preload size: number of records inserted before the start of each experiment
- Connections: number of parallel connections to the database
- Runs: number of repetitions of each experiment
- Insertion method: whether to use batch inserts or one query per record inserted

Parameter	**Values**
DBMSs	MongoDB, MySQL, PostgreSQL
Indexing	No index, spatial index
Transaction sizes	[2, 2000]
Experiment size	100 000
Preload size	1 000 000
Connections	1
Runs	10
Insertion method	Batch insert

TABLE I
Experiment parameters

Given our hypothesis, we are testing three DBMSs both with and without a spatial index. Preliminary testing with various values for transaction, experiment and preload size indicated that the three systems differed most in the range of 2-2000 for transaction sizes, while increasing experiment and preload size had less effect. While all of listed parameters may affect the result and are worth considering, due to time constraints we chose to limit the number of variables and focus only on the effect of different transaction sizes. We therefore left the other parameters at fixed values and tested 12 different values for transactions size in the range 2-2000 records. This then gives us 72 permutations of the experiment, which were then repeated 10 times in a randomized order each time. This shuffling and repetition was done to limit the extent to which external factors and the order of operations could affect the final results.

Before each experiment the relevant table/collection was dropped and recreated, and 1M records inserted. Having some number of records already in the database is important for comparing indexed vs non-indexed configurations, because indexes slow down writing data due to potentially needing to reorganize the index after each transaction.

Data generation

While measuring performance is a complex topic and an argument could be made for other methods of comparison, we have chosen to measure performance by two relatively simple metrics:

1) How long does it take (in ms) to complete a transaction?
2) How many records (per second) can the system process?

Each test therefore involved inserting 100k records into the database, in batches of between 2 and 2000 records. We used standard Python profiling utilties to measure the total execution time, logging this along with the relevant experimental parameters. From this we could derive average execution time per transaction in milliseconds and average number of rows inserted per second.

Data analysis

As we are basing our experiment on previous work [4], we will use similar data analysis methods in order to get comparable results. This will provide a basis for supporting or disproving our hypothesis.

We will compare the insertion execution time of the different DBMSs, in the range of different record sizes generated in our experiment. The results will be visualized in line graphs for comparison between the databases.

There are many factors which could affect the validity of our experiment. The choice of data set and insertion method, DBMS configuration, test hardware, and number of repetitions of the experiment are all significant factors. To combat this we will strive to make the experiment easily replicated with a standard configuration.

IV. Results

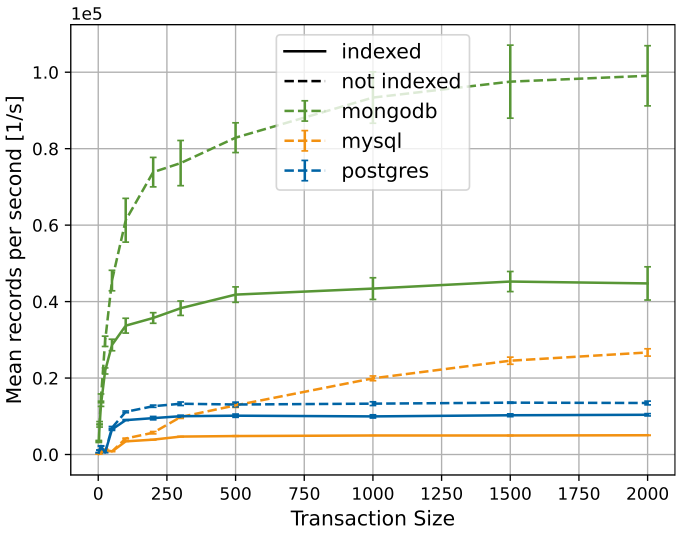

Fig. 1. Mean records per second for transaction sizes 2 to 2000

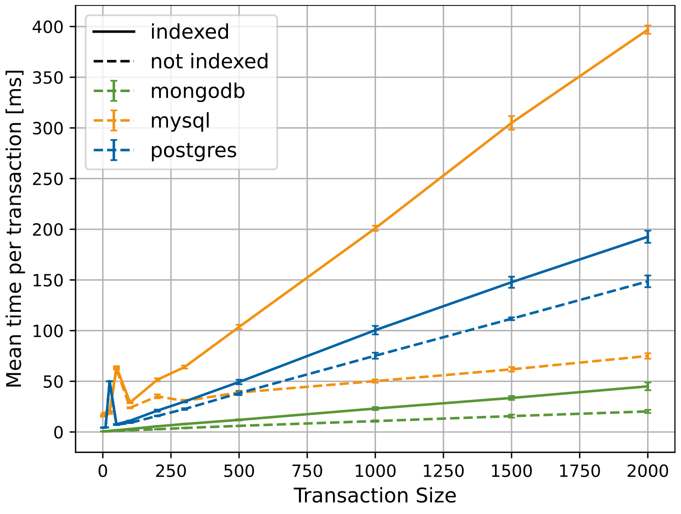

Fig. 2. Mean ms per transaction for transaction sizes 2 to 2000

Figure 1 shows the number of rows inserted per second for different transaction sizes, with an error bar showing standard deviation. The configuration with the best performance is the one with the highest value for a given transaction size. We see that MongoDB without spatial indexes has the highest insertion speed, followed by MongoDB with spatial indexes. Non-indexed insertion with PostgreSQL and MySQL each perform better than their counterpart where spatial indexes are used. More interestingly, MongoDB outperforms both PostgreSQL and MySQL for all transaction sizes. PostgreSQL has better performance than MySQL with transaction sizes under 500, while MySQL performs better than PostgreSQL with transaction sizes over 500.

We also present the time per transaction for transaction sizes up to 2,000. as seen in figure 2. In this case, high insertion performance means short time per transaction. Again, MongoDB has the best performance, both with and without spatial indexing. It can also be seen that all DBMSs seem to have a linear relation between the transaction size and the time per transaction. The exceptions to this is the performance of PostgreSQL at a transaction size of 25, and MySQL at a transaction size of 50.

V. Discussion

Our hypothesis was that PostgreSQL and MySQL would outperform MongoDB for insertions of spatial data with spatial indexes. However, our results shows that MongoDB significantly outperforms PostgreSQL and MySQL, disproving our hypothesis. In this section we will discuss why making our initial hypothesis might have been made based on wrong assumptions.

Our hypothesis was based on the assumption that because R-trees are typically used for spatial indexing they would generally outperform B-trees. Further research showed us that this assumption was flawed, as indexes are primarily used to speed up reads and our experiment exclusively considered writes. While it is true that R-trees are faster than B-trees for reading spatial data [12], they would have no advantage over B-trees in our experiment.

When inserting or updating data the index also needs to be updated, which is a potentially costly operation. For both B-trees and R-trees when a block gets full after multiple insertions, a new block will be created and the existing full block will be split in two where the one half goes to the new block. For B-trees this split is just based on the sorted order that is already in place. For R-trees it is based on bounding boxes that are calculated based on points that have close distance to each other. This means that a new bounding box have to be calculated based on the new points and the points in the full block. Conceptually this leads to significantly more calculation compared to a traditional B-tree, meaning the B-tree should have the advantage of the R-tree when exclusively looking at inserts [12]. This is the opposite of the assumption that we made as the basis for choosing our hypothesis, and we believe can in part explain why our results were so surprising.

Our hypothesis was also partly based on the results by [4], and we therefore expected our non-indexed baseline results to be similar. That this was not the case was surprising, but in retrospect we have found significant differences and possible weaknesses in their experimental setup compared to ours. Firstly, they did not use the same data set for each DBMS, which makes their direct comparison difficult. They also used a max transaction size of only 500 records, while our experiment shows clear differences in performance for larger transactions. Therefore we believe that basing our hypothesis on this paper was a mistake.

One major difference between MongoDB versus relational databases like PostgreSQL/MySQL is that MongoDB does not require the use of ACID-compliant transactions. Such transactions require atomicity and durability, which is implemented through write-ahead logging (WAL) in both PostgreSQL [15] and MySQL [16]. This essentially means that changes are appended to a log file on disk before the transaction is committed, which causes performance overhead. While MongoDB does not use ACID-compliant transactions by default, it still uses a form of WAL called journaling [17]. The journal is not immediately flushed to disk however, allowing for more efficient writes with MongoDB which we believe could explain the observed performance difference.

To summarize, we think our hypothesis was wrong because:

- In theory R-trees should have more performance penalty from inserts than B-trees, not other way around.
- The results in [4] that we were basing our hypothesis on, might have flaws in experimental setup.
- SQL databases comply with ACID properties, leading to performance overhead, especially through use of WAL.

VI. Conclusion

Our experiment showed that MongoDB had significantly higher throughput than both MySQL and PostgreSQL in this scenario, particularly for small transaction sizes. The addition of spatial indexing reduced write performance for all three systems, but MongoDB with indexes still outperformed MySQL and PostgreSQL without indexes. This could indicate that MongoDB is a better choice for applications which involve heavy writing of indexed spatial data, such as logging data from IoT devices. Our results clearly disprove our original hypothesis, suggesting that either the hypothesis was false or that there were errors in the experimental setup.

Limitations and future work

There are a number of limitations in our experimental setup which could have impacted our results. Notably, the entire experiment was run on a single computer which does not mirror real-world usage, which would involve multiple clients sending data to the DBMS. Using Docker and Python for the setup could also have incurred an unecessary performance penalty. It is also worth pointing out that MongoDB has support for ACID transactions, equivalent to MySQL and PostgreSQL, but that we ran it using default settings without this feature enabled. The experiment would also have benefited from more data, both in terms of records inserted per experiment and by testing a larger variety of transaction sizes.

With these limitations in mind, there are several other scenarios and configurations which warrant further investigation. Notably, by adding other types of queries (read, update and delete) to the experiment it could better mimic real-world usage. Setting up a distributed system with multiple client nodes communicating with a central database would also be of interest.

References

[1] A. Dziedzic et al. "DBMS Data Loading: An Analysis on Modern Hardware". In: *Data Management on New Hardware* 10195 (2017), pp. 95–117.

[2] M. Frediksen. "Database Management Systems in Smart Cities: Requirements for IoT and Time-Series Data". MA thesis. 2020.

[3] M. Kleppmann. *Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017. ISBN: 9781449373320.

[4] S. Kontogiannis Christodoulos Asiminidis George Kokkonis. "Database Systems Performance Evaluation for IoT Applications". In: *International Journal of Database Management Systems (IJDMS)* (2018).

[5] S. Park, D. Ko, and S. Song. "Parallel Insertion and Indexing Method for Large Amount of Spatiotemporal Data Using Dynamic Multilevel Grid Technique". In: *Applied Sciences* 4261.9 (2019).

[6] B. W. Low, Y. B. Ooi, and C. S. Wong. "Scalability of Database Bulk Insertion with Multi-threading". In: *Software Engineering and Computer Systems* 181 (2011), pp. 151–162.

[7] Antonio Barbuzzi et al. "Parallel bulk insertion for large-scale analytics applications". In: *LADIS '10: Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware* (2010), pp. 27–31.

[8] P. Rigaux et al. *Spatial Databases: With Application to GIS*. Series in Data Management Systems. Elsevier Science, 2002. ISBN: 9781558605886.

[9] MongoDB Inc. *New Geo Features in MongoDB 2.4*. 2021. URL: https://www.mongodb.com/blog/post/new-geo-features-in-mongodb-24 (visited on 02/25/2022).

[10] Oracle Corporation. *MySQL :: MySQL 8.0 Reference Manual :: 11.4.10 Creating Spatial Indexes*. 2022. URL: https://dev.mysql.com/doc/refman/8.0/en/creating-spatial-indexes.html (visited on 02/25/2022).

[11] PostgreSQL Global Development Group. *11.2 Index types*. 2022. URL: https://www.postgresql.org/docs/current/indexes-types.html (visited on 02/25/2022).

[12] D. Patel P.; Garg. "Comparison of Advance Tree Data Structures". In: *International Journal of Computer Applications* 41.2 (2012), pp. 11–21. URL: https://arxiv.org/ftp/arxiv/papers/1209/1209.6495.pdf.

[13] Group 2 (IT3010 spring 2022). *Insertion speed of indexed spatial data: comparing MySQL, PostgreSQL and MongoDB*. URL: https://github.com/LarsV123/it3010 (visited on 04/01/2022).

[14] Yu Zheng et al. *Geolife GPS trajectory dataset - User Guide*. Geolife GPS trajectories 1.1. Geolife GPS trajectories 1.1. July 2011. URL: https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/.

[15] PostgreSQL Global Development Group. *30.3. Write-Ahead Logging (WAL)*. 2022. URL: https://www.postgresql.org/docs/current/wal-intro.html (visited on 04/01/2022).

[16] Oracle Corporation. *14.6.5 Doublewrite Buffer*. 2022. URL: https://dev.mysql.com/doc/refman/5.7/en/innodb-doublewrite-buffer.html (visited on 04/01/2022).

[17] Inc. MongoDB. *Journaling*. 2022. URL: https://www.mongodb.com/docs/manual/core/journaling/ (visited on 04/01/2022).