International Journal of Intelligent Computing and Information Science

# An Efficient Partitioning Technique in SpatialHadoop

Ahmed Elashry
Information System Department,
Faculty of Computers and Information,
Kafr El-Sheikh University, Egypt
Ahmed_Elashry@fci.kfs.edu.eg

Abdulaziz Shehab
Information Technology Department, Faculty of Computers and Information,
Mansoura University, Egypt
abdulaziz_shehab@mans.edu.eg

Alaa. M. Riad
Information System Department,
Faculty of Computers and Information,
Mansoura University, Egypt
amriad2000@yahoo.com

Ahmed Aboul-fotouh
Information Technology Department, Faculty of Computers and Information,
Mansoura University, Egypt
elfetouh@gmail.com

***Abstract:*** *SpatialHadoop is a Hadoop framework supporting spatial information handling in light of MapReduce programming worldview. A huge number of studies leads to that SpatialHadoop outperforms the traditional Hadoop in both overseeing and handling spatial data operations. Indexing at SpatialHadoop makes it better than Hadoop. However, the design of a proficient and powerful indexing technique is stay as a major challenge. This paper presents a novel partitioning technique in SpatialHadoop. It has a better performance compared to other partitioning techniques. The proposed technique performance has been studied in several cases utilizing a real datasets on a spatial range and k-Nearest-Neighbour (kNN) queries. The experimental results have demonstrated the efficiency of the proposed technique.*

***Keywords:*** *Spatial Data indexing; Spatial partitioning; Cloud computing; SpatialHadoop; PR-Tree.*

## 1. Introduction

Geospatial big data is comprised of both information and data that generated in so many ways. This can be done either passively or actively. Passively with little or no interaction such as utilizing different types of apps, websites, smartphones, satellites, in-situ sensor networks, sensing devices, etc. Actively with more interaction such as sharing GPS tracks, geo-locating social media posts, contributing to Volunteered GIS projects, etc. Extracting knowledge from such a big geospatial data has become an extensive challenge. The traditional geographical information systems (GISs) cannot deal with these aforementioned data [1, 2]. It lacks the adaptability of basic incorporated frameworks (e.g., local files frameworks and spatial database management systems (SDBMS)). Therefore, utilizing geographical information systems with cloud computing represents a new trend toward the progression of geospatial big data storing, processing, and its applications for GISs. Cloud computing represents the largest Information Technology (IT) transformation

and migration to the cloud become a mandatory demand. Doubtlessly, Cloud computing is expected to be the area of the most substantial growth and the most significant development.

Recently, Hadoop [3, 4] released in 2007 as an open source cloud-computing platform. Hadoop was written in Java and funded by Apache. It is implemented for Google MapReduce. Hadoop utilizes MapReduce [5-8] to build an effective large-scale data processing structure. MapReduce is a programming paradigm for distributed data processing [9]. It provides high scalability and fault tolerance mechanisms. Hadoop represents a solution for scalable big data processing in a variety of applications. Unlike traditional technologies which suffer low execution and the complexity experienced when processing and analysing big data, Hadoop can easily perform many operations in just a few seconds because it has a parallel clusters processing and a distributed file system. Hadoop is not a schema oriented so it can retain any kind of data, structured or not, from various sources. These heterogeneous data can be joined and processed in many arbitrary ways and enhancing further analysis [10].

In this paper, a novel partitioning technique in SpatialHadoop is presented. The proposed technique takes into consideration the desired number of partitions in partitioning different spatial datasets. Additionally, the spatial proximity of these spatial data is highly preserved. An expansive set of experiments on a different real datasets are executed to prove our contributions. The indexing time and query execution time is documented. The results show that the proposed technique overcomes all other techniques in terms of functionality and performance.

This paper is organized as follows: Section 2 presents related works on SpatialHadoop partitioning and indexing techniques. Section 3 presents the proposed technique. Section 4 illustrates the experimental configurations, the performance measures, and the results of the experimentations. Finally, Section 5 presents the conclusion and discusses future research directions.

## 2. Related work

Last few years, many researchers are oriented towards the usage of both Hadoop and the MapReduce environment that works on big geospatial data. Their work can be classified into two main categories: (1) spatial-operation oriented and (2) Full-system oriented. For the first, it is focus on a specific spatial operation. In this category, the essence idea is to build the MapReduce functions for a specific spatial operation. It can be viewed as an upper layer that works upon traditional Hadoop. Instances of such researches incorporate: (1) Range query [11-13], the input dataset is investigated, and each object is analyzed with respect to the query range. (2) k Nearest Neighbour (kNN) query [12, 14, 15], locates the k closest objects, utilizing distance metrics, from a specific object. (3) All Nearest Neighbour (ANN) query [9, 16], having N objects and process to know which is the closest neighbor for each one of those N objects, objects are divided by their Z-values to find out the result. (4) Reverse Nearest Neighbour (RNN) query [14], a reverse nearest neighbor query is to scan for all objects which a specific location is their closest neighbor.

For the second, five systems were presented: (1) Parallel-Secondo [17] is a parallel and distributed spatial DBMS. it utilizations Hadoop to work as a distributed task scheduler, (2) MD-HBase [18] is a Hadoop non-

relational database that supports multidimensional indexes. It represents as an expansion of HBase [19], (3) Hadoop-GIS [20] is a Hadoop information distribution center foundation that utilizes a uniform grid index for different spatial operations. it represents an expansion of Hive [4], and (4) GeoSpark [21, 22] is an execution of a few spatial operations on the Apache Spark. It is in-memory huge information framework. It focuses on in-memory processing for better performance.

All aforementioned systems are built as an upper layer over Hadoop. They deal with Hadoop as a black box [23], and subsequently, they still have the limitations of the Hadoop system [24]. Hadoop does not support spatial data. Hadoop processes both non-spatial data and spatial data in a similar way. Hadoop has confinements and execution bottlenecks. Moreover, As Hadoop supports uniform grid index only, these systems are only suitable for uniform data distribution. These systems developed as a layer on top of Hadoop, so there is no way for the MapReduce programs to access any constructed spatial index. Thus, new spatial operations cannot be developed.

Unlike these systems, SpatialHadoop [25, 26] is developed to insert spatial data inside the essence of Hadoop. SpatialHadoop represents a Hadoop framework suited for spatial operations. By such a way, SpatialHadoop become more efficient to deal with spatial query processing. Moreover, SpatialHadoop introduces standard spatial indexes and MapReduce components that allow researchers and developers to implement new spatial operations efficiently in the framework [25, 27].

In SpatialHadoop, spatial data is splitted according to their spatial closeness into partitions. These partitions are disseminated to the cluster nodes where they are indexed later. SpatialHadoop has the ability to support a set of spatial index structures utilizing a set of partitioning techniques like grid [23], R-tree [28], R+-tree [29], Z-curve [30, 31], Hilbert curve [32], Quadtree [33], and KD-tree [34, 35]. Each one of these techniques were developed in Hadoop Distributed File System (HDFS). Thus allowing developing effective algorithms for query processing that search a fraction of the data and still provide the valid query result. Subsequently, this makes SpatialHadoop special unique regarding supporting data distribution in geospatial data [25].

## 3. The proposed technique

Figure 1 shows the four layers of SpatialHadoop: (1) The Language layer named Pigeon [36] which enables users to assign their spatial queries to the framework without worrying about the processing details. It adapted to the Open Geospatial Consortium (OGC) standard. By such a way, the Pigeon language could easily integrate with different systems through exporting/importing data in OGC standard formats. (2) The Operations / Query Processing layer includes the spatial operations upheld by SpatialHadoop. Three main SpatialHadoop's operations are range query, kNN, and spatial join. In addition, The SpatialHadoop query processing engine based on Hadoop MapReduce allows users to develop custom spatial operations that utilizes the constructed spatial indexes. (3) The MapReduce layer responsible for enabling the access to the spatial indexes. It contains two components, SpatialFileSplitter and SpatialRecordReader. The SpatialFileSplitter accesses the global index to return only file partitions that are related to the required query. On the other hand, the SpatialRecordReader works at the resultant partitions using the local index. (4) The Storage / Spatial Indexing layer has a two-layer index structure (one global index and many local

indexes). The global index splits data on the cluster nodes. Thereafter, using local index, each node indexes its partition separately. The separation of global and local indexes makes it easy to use MapReduce programming model.

Building spatial index in SpatialHadoop goes into three stages: partitioning, local indexing, and global indexing [25]. For the first, the input data is divided to a number of partitions taking into consideration spatial proximity of different objects to be stored in the same partition. Each partition should be 64 MB to be stored in one HDFS block. Regardless of the spatial index type, the number of partitions are calculated based on the input files size, file block size, and the overhead of storing local indexes. Then, each partition is defined by a rectangle, differently according to the underlying index being constructed. Meanwhile, a MapReduce job initiated to physically splitting the input file. For the second, in local indexing stage, a local index structure is built upon the data contents of each partition. Here, records assigned to each partition are entered to a reduce function. The reduce function creates the local spatial index for that partition and then stores it in the local node index. For the last, all local indexes are aggregated into one file for all partitions. Then, the master node uses bulk loading and the rectangular boundaries of all file blocks as the index key to build an in-memory global index.

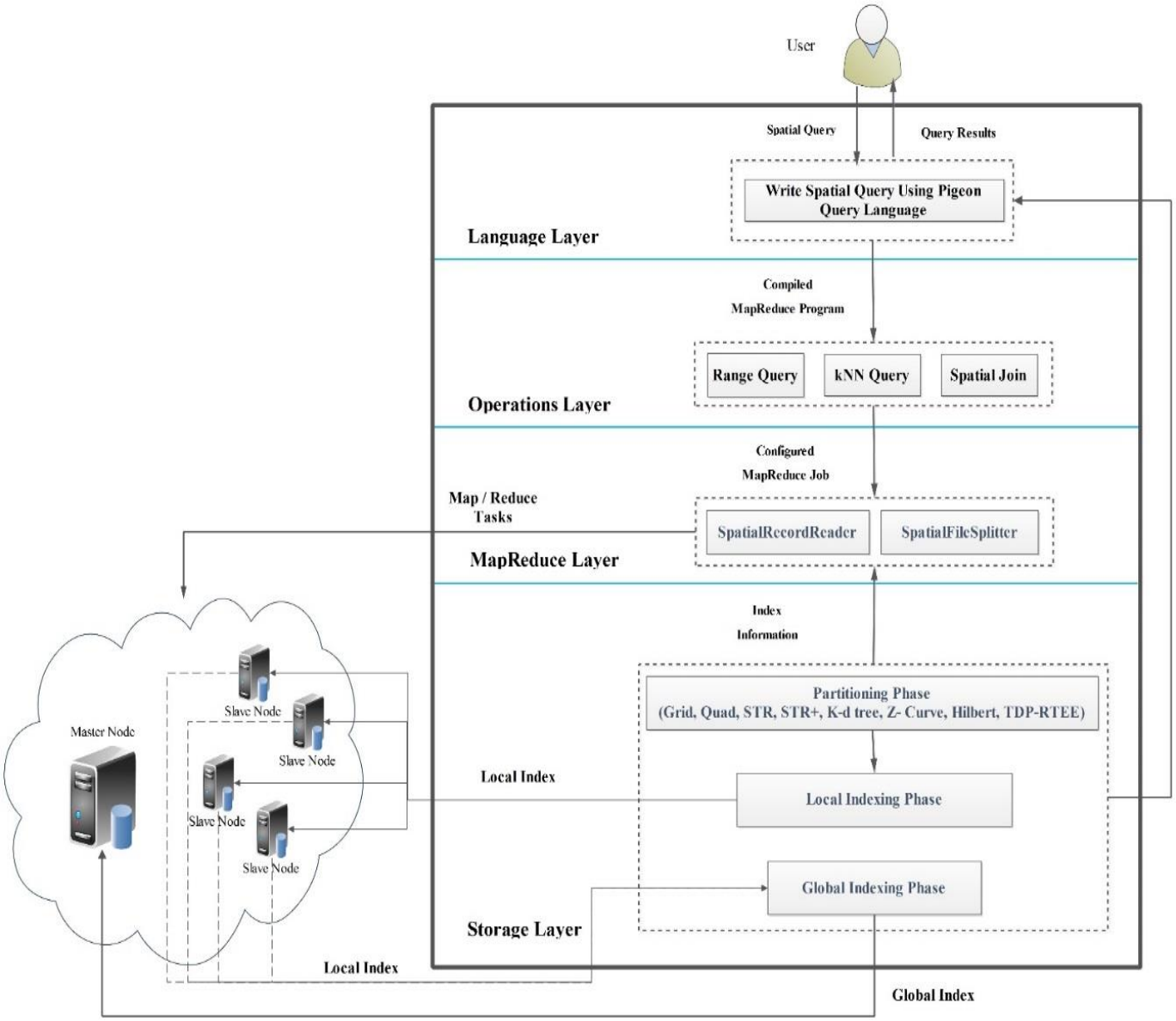



Figure 1. SpatialHadoop system architecture.

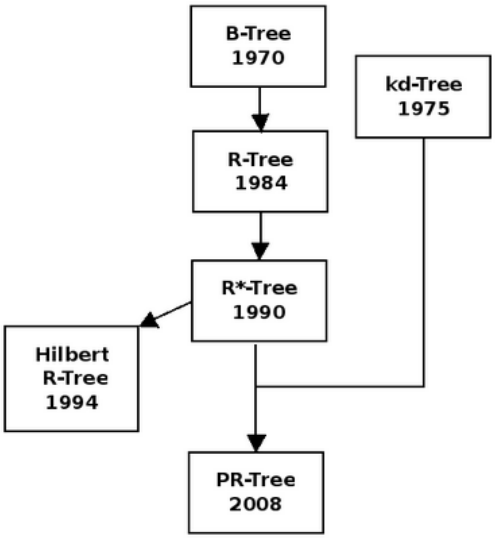



Figure 2. The history of PR-Tree [37].

In order to improve the performance of SpatialHadoop indexing and partitioning phase, a new indexing and partitioning technique is proposed. Our proposal is mainly based on the Priority R-tree (PR-Tree) [38]. Figure 2 summarizes the history of Priority R-tree stages. The proposed technique starts, as shown in Algorithm 1, by calculating the maximum number of shapes that can be fit in one partition (leaf). Then, each input shape is converted to a 4D point. After that, a root node is created and the total number of shapes is checked. If it is less than or equal to the maximum number of shapes that can be fit in one partition (leaf), if so then the algorithm generates a scalar priority leaf $\nu_\rho$. Unless, it generates a priority leaf $\mathcal{V}_p^{x_{min}}$ that stores a B shapes with minimal x-coordinates. then, the algorithm is process in the same way to produce the other three priority On the other hand, if the root node has a number of shapes higher than 4B, then a two sub PR-Trees and a four-priority leaves are produced. The algorithm recursively applies these calculations until no shapes remaining to be filled.

**Algorithm** 1: Building 4DPR-tree index

**Function 4DPRTree_Index_Building (** S, $P_{Num}$ **)**

**Input:** S *is a spatial data file that has a set spatial objects and* $P_{Num}$ *is the number of required partitions*

**Output:** a 4DPR-Tree as a stack

**Method:**

1. ***Calculate*** *B, which is the number of object, should be stored per each partition (leaf), by dividing the total number of shapes in the file by the desired number of partitions.*
2. ***Convert*** *each object in the input data file to a 4D point with* $(x_{min}, y_{min}, x_{max}, y_{max})$ *and store it in a new stack.*
3. ***Create*** *a root node that stores the minimum boundary rectangle of all input objects with depth equal to zero*

*and store that root node in a new stack, named tree_stack*

4. ***Foreach*** *node in the tree_stack* ***do*** *// starting from the root node*

*4.1.* ***if*** *the number of objects <= B* ***then***

*4.1.1.* ***Create*** *a single priority leaf that stores these objects*

*4.2.* ***Else If*** *the number of objects <= 4* B* ***then***

*4.2.1.* ***Create*** *the first priority leaf that stores a B objects with the lowest* $(x_{min})$ *values.*

*4.2.2.* ***If*** *the number of remaining objects <= B then*

***4.2.2.1. Create*** *a single priority leaf that stores the remaining objects.*

*4.2.3.* ***Else***

***4.2.3.1. Create*** *the second priority leaf that stores a B objects with the lowest* $(y_{min})$ *values.*

***4.2.3.2. If*** *the number of remaining objects <= B* ***then***

***4.2.3.2.1. Create*** *a single priority leaf that stores the remaining objects.*

***4.2.3.3. Else***

***4.2.3.3.1. Create*** *the third priority leaf that stores a B objects with the highest* $(x_{max})$ *values.*

***4.2.3.3.2. Create*** *the fourth priority leaf that stores the remaining objects.*

***4.3. Else***

*4.3.1.* ***Create*** *the first priority leaf that stores a B objects with the lowest* $(x_{min})$ *values.*

*4.3.2.* ***Create*** *the second priority leaf that stores a B objects with the lowest* $(y_{min})$ *values.*

*4.3.3.* ***Create*** *the third priority leaf that stores a B objects with the highest* $(x_{max})$ *values.*

*4.3.4.* ***Create*** *the fourth priority leaf that stores a B objects with the highest* $(y_{max})$ *values.*

*4.3.5.* ***Calculate*** *the μ value equal to* $(\lfloor(\lfloor(n - 4 * B)/(4 * B)\rfloor/2)\rfloor * 4 * B)$ *that is used to splitting the rest of the objects into two subsets.*

*4.3.6.* ***If*** *current node depth remainder by four equal to zero* ***then*** *split based on* $(x_{min})$ *values.*

*4.3.7.* ***Else If*** *current node depth remainder by four equal to one* ***then*** *split based on* $(y_{min})$ *values.*

*4.3.8.* ***Else If*** *current node depth remainder by four equal to two* ***then*** *split based on* $(x_{max})$ *values.*

*4.3.9.* ***Else,*** *split based on* $(y_{max})$ *values.*

*4.3.10.* ***Create*** *a new two sub tree nodes with depth greater than the current tree node depth by one.*

*4.3.11.* ***Add*** *these two sub trees nodes to the tree_stack.*

## 4. Experimentation

### *4.1 Experimental setup*

Amazon cluster consists of one master node and four slave nodes, all of type 'm3.xlarge', was used to perform all the experiments. Each 'm3.xlarge' node has 4vCPU Intel Xeon processors with a high frequency with 15 GB RAM and 2*40 GBSSD storage [39]. All cluster nodes have be configured to run Linux operating systems with Java 8. Hadoop2.7.2 and SpatialHadoop were installed and configured on all cluster nodes. A real datasets extracted from OpenStreetMap: Buildings (28.2 GB), Roads (25.9 GB), Lakes (9 GB), Cities (1.4 GB), and Sports (590 MB) [25] were used to test all partitioning and indexing techniques.

### *4.2 Experimental results*

Table 4 shows the time in seconds that is required by the gplot function which responsible for plotting the different real datasets files. It is noted that the proposed 4DPR-Tree has the shortest plotting time for different datasets. This because 4DPR-Tree consider the preserved spatial proximity of the spatial shapes.

Table 4. Plotting time of different real datasets

| | **Sports** | **Cities** | **Lakes** | **Roads** | **Buildings** |
|---|---|---|---|---|---|
| 4DPTree | 26.443 | 31.605 | 72.068 | 153.031 | 167.726 |
| Kd-tree | 26.453 | 36.710 | 77.395 | 157.994 | 178.338 |
| Quadtree | 41.505 | 47.256 | 122.864 | 254.828 | 319.729 |
| Z-curve | 26.463 | 36.809 | 72.339 | 158.086 | 187.859 |
| Hilbert | 31.643 | 33.678 | 72.377 | 148.016 | 168.645 |
| STR | 31.731 | 46.691 | 77.110 | 153.505 | 168.187 |
| STR+ | 26.561 | 46.563 | 77.390 | 148.431 | 168.212 |

Table 5 presents partitions number generated by different partitioning techniques for the different real datasets. All partitioning techniques except STR, STR+, and Quadtree create the required partitions.

Table 5. Partitions generated by different indexing techniques.

| **Datasets** | **Partitions Num.** | | | | |
|---|---|---|---|---|---|
| | **Sports** | **Cities** | **Lakes** | **Roads** | **Buildings** |
| 4DPR-Tree | 6 | 14 | 87 | 232 | 252 |
| KD-Tree | 6 | 14 | 87 | 232 | 252 |
| Quadtree | 25 | 34 | 246 | 593 | 705 |
| Z-curve | 6 | 14 | 87 | 232 | 252 |
| Hilbert | 6 | 14 | 87 | 232 | 252 |
| STR | 6 | 18 | 91 | 241 | 252 |

| | | | | | |
|---|---|---|---|---|---|
| STR+ | 6 | 18 | 91 | 241 | 252 |

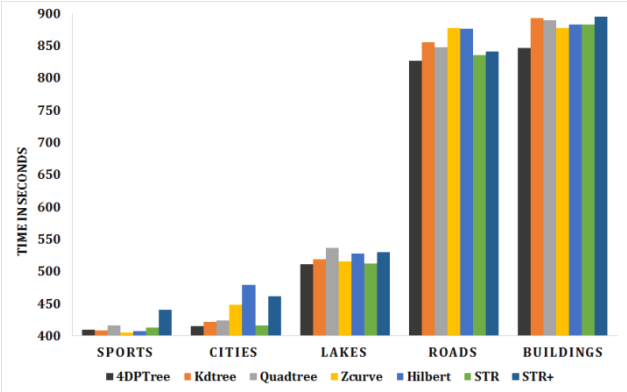
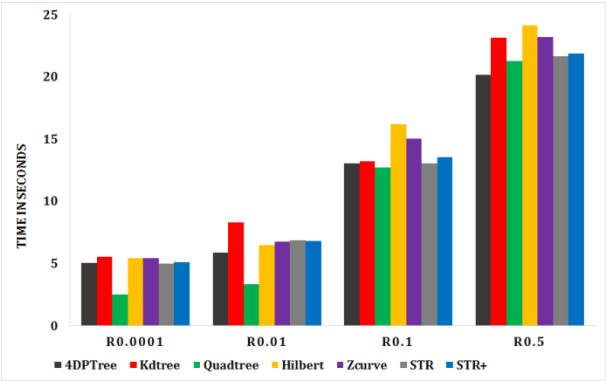
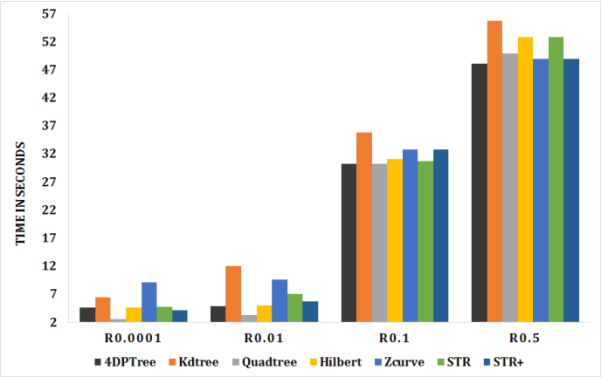



Figure 3. Sports, Cities, Lakes, Roads, and Buildings index building time.

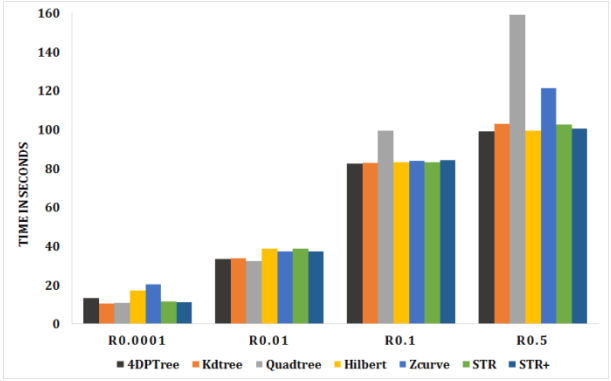
Figure 3 shows Sports, Cities, Lakes, Roads, and Buildings indexing time. 4DPR-Tree has the best index building time especially for Roads, and Buildings. For the range query, Figure 4 (a, b, and c) shows the performance of range query on the Sports, Cities and Roads datasets. As noticed, Quadtree performance rapidly decreased with the changing in query window areas to 10-50% of the input dataset area. The reason behind that is the partitions number required to be accessed to answer the query is increased as the query window area is increased. Inversely to Quadtree, the proposed 4DPR-Tree technique outperforms other methods for 10-50% for the queries with query window 10-50% of the input dataset area. As it generates the required number of partitions and simultaneously it preserve the spatial proximity of the input objects.




(a) Sports

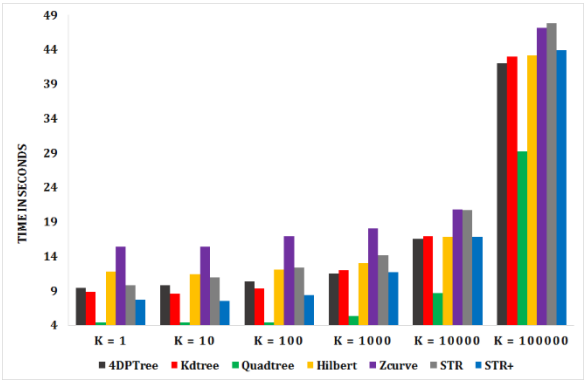



(b) Cities

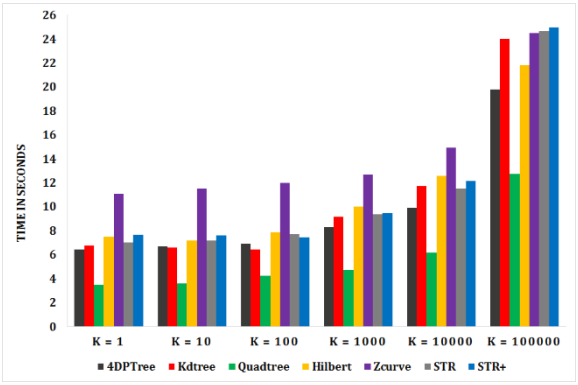



(c) Roads

Figure 4. Sports, Cities, and Roads execution time for Range query.




(a) Lakes




(b) Buildings

Figure 5. Lakes and Buildings execution time for kNN query.

Figure 5 (a and b) presents the performance of kNN for the Lakes and Buildings with different k values that has been changed from 1 to 10,000. It is obvious that Quadtree outperforms the other methods. However, Quadtree does not committed with the required partitions that should be generated. As shown in table 5, although the required partitions for Lakes and Buildings datasets are 87 and 252, Quadtree has partitioned the Lakes and Buildings datasets into 246 and 705 partition. Furthermore, the proposed technique outperforms all other techniques that obligated to the required partitions especially for high k values (1000, and 10000).

## 5. Conclusions and Future Work

In this paper, 4DPR-Tree is proposed as a novel partitioning technique in SpatialHadoop. Various SpatialHadoop partitioning techniques have been experimentally evaluated. The experiments show that spatial query processing is very reliant on the size and nature of the dataset. Indexing and partitioning techniques demonstrate diverging performance with the alternative datasets types. The experimental results show that Quadtree, STR, STR+ generate a number of partitions higher than the desired and take the maximum indexing time. In addition, for Range query, all other techniques performance is highly decreased as the input dataset size and the query window area become larger. On the other hand, 4DPR-Tree has a better indexing time and a better Range query execution time for all datasets sizes as it generates the desired number of partitions and highly preserves the spatial proximity of the input objects. Moreover, for kNN query, the performance of 4DPR-tree becomes better than all other techniques as the k values and the dataset size become higher. As part of our future work, we will develop new multi-dimensional spatial data types on SpatialHadoop and a new indexing technique for these data types will be developed with the goal of further enhancing query response time.

## References


1. Li, Z., et al., A spatiotemporal indexing approach for efficient processing of big array-based climate data with MapReduce. International Journal of Geographical Information Science, 2017. 31(1): p. 17-35.
2. Cary, A., et al. Leveraging Cloud Computing in Geodatabase Management. in 2010 IEEE International Conference on Granular Computing. 2010.
3. White, T., Hadoop: The Definitive Guide, O'reilly. 2012.
4. Thusoo, A., et al., Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow., 2009. 2(2): p. 1626-1629.
5. F.Li, et al., Distributed data management using MapReduce. ACM Comput, 2014(46(3)): p. 31:1-31:42.
6. Doulkeridis, C. and K. Nrvag, A survey of large-scale analytical query processing in MapReduce. VLDB J, 2014.
7. Eldawy, A., et al., CG_Hadoop: computational geometry in MapReduce, in Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2013, ACM: Orlando, Florida. p. 294-303.
8. Dean, J. and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters. Communications of ACM, 2008. 51.
9. Wang, K. and e. al, Accelerating Spatial Data Processing with MapReduce. ICPADS, 2010.
10. Oussous, A., et al., Big Data technologies: A survey. Journal of King Saud University - Computer and Information Sciences, 2017.
11. Aly, A.M., et al., Kangaroo: Workload-Aware Processing of Range Data and Range Queries in Hadoop, in Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. 2016, ACM: San Francisco, California, USA. p. 397-406.
12. Zhang, S., et al. Spatial Queries Evaluation with MapReduce. in 2009 Eighth International Conference on Grid and Cooperative Computing. 2009.

13. Ma, Q., et al., Query processing of massive trajectory data based on mapreduce, in Proceedings of the first international workshop on Cloud data management. 2009, ACM: Hong Kong, China. p. 9-16.
14. Akdogan, A., et al. Voronoi-Based Geospatial Query Processing with MapReduce. in 2010 IEEE Second International Conference on Cloud Computing Technology and Science. 2010.
15. Nodarakis, N., et al., (A)kNN Query Processing on the Cloud: A Survey, in Algorithmic Aspects of Cloud Computing: Second International Workshop, ALGOCLOUD 2016, Aarhus, Denmark, August 22, 2016, Revised Selected Papers, T. Sellis and K. Oikonomou, Editors. 2017, Springer International Publishing: Cham. p. 26-40.
16. Sankaranarayanan, J., H. Samet, and A. Varshney, A fast all nearest neighbor algorithm for applications involving large point-clouds. Comput. Graph., 2007. 31(2): p. 157-174.
17. Lu, J. and R.H. Guting. Parallel Secondo: Boosting database engines with Hadoop. in ICPADS 2012.
18. Nishimura, S., et al., MD-HBase: Design and Implementation of an Elastic Data Infrastructure for Cloud scale Location Services. DAPD, 2013. 31(2): p. 289–319.
19. HBase. Apache HBase. 2008 [cited 2017 10 JUN ]; Apache HBase™ is the Hadoop database. Use it when you need random, realtime read/write access to your Big Data. This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware.]. Available from: http://hbase.apache.org/.
20. Aji, A., et al., Hadoop GIS: a high performance spatial data warehousing system over mapreduce. Proc. VLDB Endow., 2013. 6(11): p. 1009-1020.
21. You, S., J. Zhang, and L. Gruenwald, Large-scale spatial join query processing in cloud. ICDE Workshops, 2015: p. 34-41.
22. Yu, J., J. Wu, and M. Sarwat, GeoSpark: a cluster computing framework for processing large-scale spatial data, in Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2015, ACM: Seattle, Washington. p. 1-4.
23. Eldawy, A., L. Alarabi, and M.F. Mokbel, Spatial Partitioning Techniques in SpatialHadoop, in International Conference on Very Large Databases. 2015: Kohala Coast, HI.
24. Siddiqa, A., A. Karim, and V. Chang, Modeling SmallClient indexing framework for big data analytics. The Journal of Supercomputing, 2017: p. 1-22.
25. Eldawy, A. and M.F. Mokbel, SpatialHadoop: A MapReduce framework for spatial data, in ICDE Conference. 2015. p. 1352-1363.
26. Maleki, E.F., M.N. Azadani, and N. Ghadiri. Performance evaluation of SpatialHadoop for big web mapping data. in 2016 Second International Conference on Web Research (ICWR). 2016.
27. Eldawy, A., SpatialHadoop: towards flexible and scalable spatial processing using mapreduce, in Proceedings of the 2014 SIGMOD PhD symposium. 2014, ACM: Snowbird, Utah, USA. p. 46-50.
28. Guttman, A., R-trees: a dynamic index structure for spatial searching. SIGMOD Rec., 1984. 14(2): p. 47-57.
29. Beckmann, N., et al., The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec., 1990. 19(2): p. 322-331.
30. Zhang, R. and C.-T. Zhang, A Brief Review: The Z-curve Theory and its Application in Genome Analysis. Current Genomics, 2014. 15(2): p. 78-94.
31. Zhang, R. and C.-T. Zhang, Z Curves, An Intutive Tool for Visualizing and Analyzing the DNA Sequences. Journal of Biomolecular Structure and Dynamics, 1994. 11(4): p. 767-782.

32. Meng, L., et al., An improved Hilbert curve for parallel spatial data partitioning. Geo-spatial Information Science, 2007. 10(4): p. 282-286.
33. Zhang, J. and S. You, High-performance quadtree constructions on large-scale geospatial rasters using GPGPU parallel primitives. International Journal of Geographical Information Science, 2013. 27(11): p. 2207-2226.
34. Wei, H., et al., A k-d tree-based algorithm to parallelize Kriging interpolation of big spatial data. GIScience & Remote Sensing, 2015. 52(1): p. 40-57.
35. Nandy, S.K., et al., K-d Tree based Gridless Maze Routing on Message Passing Multiprocessor Systems. IETE Journal of Research, 1990. 36(3-4): p. 287-293.
36. Eldawy, A. and M. F. Mokbel, Pigeon: A spatial MapReduce language. 2014. 1242-1245.
37. Davies, J. Implementing the Pseudo Priority R-Tree (PR-Tree), A Toy Implementation for Calculating Nearest Neighbour on Points in the X-Y Plane. 2011 April 18th, 2011 [cited 2017 18 Aug]; Available from: http://juliusdavies.ca/uvic/report.html.
38. Arge, L., et al., The priority r-tree: A practically efficient and worst-case optimal r-tree. ACM Transactions on Algorithms, 2008. 4(1).
39. Amazon. Amazon EC2. 2017 [cited 2017 10 JUN ]; Available from: http://aws.amazon.com/ec2/.