In-memory Spatial-Aware Framework for Processing Proximity-Alike Queries in Big Spatial Data

Isam Mashhour Al Jawarneh, Paolo Bellavista, Antonio Corradi, Luca Foschini, Rebecca Montanari, Andrea Zanotti
DISI, University of Bologna, Bologna, Italy
isam.aljawarneh3@unibo.it, paolo.bellavista@unibo.it, antonio.corradi@unibo.it, luca.foschini@unibo.it, rebecca.montanari@unibo.it,
andrea.zanotti@studio.unibo.it

Abstract**—** The widespread adoption of sensor-enabled and mobile ubiquitous devices has caused an avalanche of big data that is mostly geospatially tagged. Most cloud-based big data processing systems are designed for general-purpose workloads, neglecting spatial-characteristics. However, interesting analytics often seek answers for proximity-alike queries. We fill this gap by providing custom geospatial service layer atop of Apache Spark. To be more specific, we leverage Spark to design a custom spatial-aware partitioning method to boost geospatial query performances. Our results show that our patches outperform state-of-the-art implementations by significant fractions.

Keywords—spatial data; smart city; boundary spatial objects; co-locality mining, density-based clustering.

I. Introduction

The last two decades or so have witnessed an accumulation of smart city and Industry 4.0 big data that are mostly geo-referenced. This mass of data is better known as *big spatial data* and is exploited for wide spectrum of applications, including participatory healthcare [1] and city planning [2]. Batch-processing systems are not designed to process this data avalanche. A point that has motivated the emergence of parallel computing ecosystems such as MapReduce [3] (and its variations) and MongoDB [4] for big data storage management (but including processing capabilities). Two entwined aspects that stand out in this context are data partitioning and query processing. Those are interlinked in such a way that query processing (as an access pattern) heavily depends on the way data is partitioned within the underlying parallel computing environment. Partitioning can be loosely defined as dispatching data loads to various cloud computing elements in a manner that respects some predefined constraints such as sending roughly equal data subsets to each processing element which is better known as load balancing in the relevant literature. While this works typically well for general-purpose non-geotagged workloads, it might perform worse for spatial workloads because spatial data loads can be highly skewed, where spatial objects tend to reside in densities in real geometries [5, 6]. One of the most common partitioning strategies for such workloads that can be traced in the literature is the Fixed Grid Partitioning, which (FGP) [7], which divides the search space into fixed-sized grids, thus sending data contained in each grid to a specific partition for parallel processing.

However, one of the intuitive challenges that this method induces when applied to spatial workloads is a phenomenon that we term as *boundary spatial objects (BSO)*, which are the objects that resides exactly on the borders between grid cells that is especially time-consuming for complex proximity-alike queries such as geodata clustering algorithms. Processing those stragglers impose a challenge as to which processing element those should be sent to. Some works in the literature mitigate this effect by an approach we refer to as replicate-and-postprocess, where BSOs are replicated between neighboring partitions, thereafter a postprocessing step is applied to eliminate replications [8] which significantly takes a huge toll on system performance, sometimes intuitively causing the system to come into a halt. Another important spatial characteristic is co-locality, where spatial objects normally share natural pairwise relationship in such way that geometrically co-located spatial objects share some other characteristics. For example, two friends gather in a bar for a coffee. As more and more smart city and Industry 4.0 applications are seeking answers for proximity and co-location alike queries, it seems interesting to preserve such spatial characteristic while disseminating spatial workloads to cloud computing elements, thus avoiding costly shuffling that may heavily tax the processing system and challenge its capacities.

However, current cloud-based ecosystems do not offer adequate solutions for those challenges because although those are domain-specific fixes and patches, they normally do not adequately trade-off partitioning challenges mentioned above. To overcome those limitations, our contribution is twofold. First, we fill this void by designing a spatial-characteristics-aware adaptive partitioning method that adequately trade-off BSOs, spatial co-locality and load balancing. Second, we adapt our partitioning method in a manner that better exploit its overarching traits for density-based clustering [9] with the ability to integrate spatial queries.

The rest of the paper is divided as follows. First, we provide a background covering spatial aware data partitioning challenges and most interesting spatial queries. We then introduce our framework describing the general architecture including patches we have provided. We then comprehensively explain the working mechanism of our spatial aware partitioning method. We then show results of our method and its application in a complex spatial analytics scenario, and then we conclude the paper providing some future research frontiers.

II. Background

Performance shortcomings of batch-processing systems has led to the birth of parallel-based systems, from which we select

Apache Spark [10] (thereafter, Spark for short) as a representative because it has established itself as a de facto standard for big data processing workloads. Spark works mainly by distributing a working set (known as RDD [11]) into the volatile memory of processing elements directly after the *Map* phase and just before the *Reduce* phase following the *MapReduce [3]* paradigm. In the following, we provide background about Spark and its current support of big spatial data partitioning and querying.

A. Big spatial data partitioning

Data partitioning can be loosely defined as dispatching big data loads to processing elements of a parallel computing network, aiming at distributing the workload thus improving the throughput [12]. Traditionally, strategies include range and hash partitioning that are designed for handling general-purpose workloads, which was beneficial before smart cities turn into a trend. In the last decade, initiatives such as smart cities and Industry 4.0 changes the way data is collected such that mostly all objects are tagged with location. The current shape of data makes it better termed as big spatio-temporal data, thus rendering the application of classical partitioning methods unsuitable. Perhaps most significantly is the fact that spatial analytics request answers that seek co-location or the so-called proximity-alike analysis. In a cloud computing context, this means that not respecting data co-locality while partitioning cause a costly shuffling between elements, thus heavily taxing the underlying ecosystem. This led to the emergence of spatial-aware partitioning methods, including quadtrees [13] and space-filling-curves (SFC) [14](examples include, Hilbert-curves [15] and Z-curves [16]), which aim at dimensionality reduction thus converting a planar multidimensional space into single dimension. Off-the-shelf, Spark does not support these spatial-aware methods, which motivated the birth of Spark-based frameworks that provide support for such methods. As a representative, GeoSpark [17] supports FGP and Hilbert-curves. Despite the fact those methods preserve co-locality to some good extent, they leave the cluster unbalanced, where some elements acquire more loads than others, thus turning them into stragglers that soon may become performance bottlenecks. In addition, BSOs effects are not considered. To fill those voids, we provide a custom method that is a hybrid of some of those from the relevant literature, aiming at a weighted balance between BSOs, spatial co-location and load balancing.

B. Big spatial data querying

Spatial data partitioning is a mean-to-an-end, where the goal is improving query performance. The main computing task of parallelly-executed queries is scanning appropriate elements for result sets. In a traditional parallel computing framework, this often means shuffling datasets among elements so as to offer subset copies locally for algorithms to achieve their designated tasks. This shuffling however is specifically challenging, especially if subsets to be shuffled constitute a huge fraction of the whole dataset. Query optimizers work by enhancing all factors that slow down query processing. In a distributed context, this involves many activities spanning from the baseline partitioning and including query profiling among many other solutions, thus collectively aiming at providing a better quality of service (QoS) of latency and throughput by pruning the search space.

Basic spatial queries include; i) range (containment) that returns objects surrounding a specific spatial object (radius or circumference measures, also known as Minimum Bounding Rectangle (MBR)). ii) *proximity-alike* (for example, *nearest neighbor (NN) query)* that finds objects geo- co-located with a specific object or set of objects (the case of *k*NN) such as finding nearest hospitals to an accident location. iii) *spatial join query*, joining two spatial subsets. In this paper, we concentrate on testing the performance of two spatial query types: range and join. Complex spatial queries are composable of some of those basics.

Moreover, we aim at providing optimizations for one of the most complex operations that spatial big datasets may encounter: density-based clustering. In simple terms, density-based clustering aims at grouping corelated objects into subgroups (known as clusters), where dense regions constitute clusters, whereas non-dense regions are boundaries. This kind of clustering cause a problem we termed as BSOs, which are those that reside exactly on-borders. A well-established algorithm in this family is DBSCAN that is widely used in the smart cities scenarios. However, the vanilla version is not designed to run in parallel computing environment. A point that has witnessed the birth of alternatives that adapt the classical version to work in clouds. From the constellation of works of the relevant literature, MapReduce-DBSCAN (MR-DBSCAN) [18] and DBSCAN-MR [9] stand out, aiming at optimizing throughput. DBSCAN requires two parameters; ε and MinPts. For any point p, ε is a measure to find the set of all points near p within maximum distance of ε (somewhat similar to kNN [19]).

MinPts is used to check the minimum number of neighboring points that can be collectively considered as a cluster. Despite some disparities, MR-DBSCAN proceeds as follows. First, it dispatches datasets to processing nodes, then perform a local clustering by applying the vanilla DBSCAN to each node independently. The last stage incorporates merging local results, aiming at constituting a global clustering result. On the vanilla DBSCAN, distance of all points from a focal point is calculated, which is not the case in MR-DBSCAN. To circumvent this, MR-DBSCAN first sorts the input data and then replicates the points located on the edges of adjacent partitions, so that they are considered in all relevant partitions, thereafter a postprocessing step is required to eliminate duplication, which is a resource-intensive process. One inherent problem with this design is that it only respects spatial co-locality while partitioning, thus neglecting the other two spatial characteristics (BSOs and load balancing). We have solved those in our adaptation for the MR-DBSCAN algorithm herein.

Out of the box, neither Spark nor GeoSpark supports density-based clustering algorithms. We aim to fill this void by providing optimization atop GeoSpark that transparently inject density-based clustering support. We have adapted MR-DBSCAN algorithm so that it works atop GeoSpark, which constitutes one of our core contributions.

Spatial-aware optimizations Partitioning and advanced query optimizers
Geopark (including spatial representational support)
Spark Core

Fig 1. Layered spatial-aware in-memory processing optimization architecture.

III. Big Spatial-Characteristic Preserving Data Processing Optimizers

Fig 1. shows a high-level architecture of our spatial-aware optimizations. Our patches reside atop GeoSpark (which itself sits at the core of Spark) constituting a transparent layer that hides implementation details from application layer. Our patches include spatial-aware partitioning strategies, basically accounting for a better tradeoff of three challenges (load balancing, BSOs and spatial co-locality) and adapt DBSCAN-MR algorithm [20] (which belongs to the co-location data mining family[21]) to work atop GeoSpark.

A. Self-Adaptable Spatial-Aware Partitioner (SASAP)

This section presents a custom spatial-aware partitioning strategy for Spark aimed at improving QoS in parallel processing ecosystems.

Several works from the relevant literature have employed Fixed Grid Partitioning (FGP) [7] for partitioning spatial data loads in cloud-computing environments. However, the naïve implementation of this method creates objects that resides on borders between grid cells, which sometimes contributes a huge factor of total data percentage, thus imposing a challenge that heavily taxes resource capacity. To alleviate this burden, we have designed a custom adaptive spatial-aware partitioning method that accounts for those BSOs in addition to co-location and load balancing. We term our method as self-adaptable spatial-aware partitioning method (*SASAP* for short). Our method is adaptable in the sense that for every application session, it self-tunes division factors for the benefit of subsequent sessions, aiming at balancing loads among participating elements while resolving BSOs and co-locality to some good degree. In particular, our method works by calculating automatically new cutting configurations (analogous to vertical partitioning lines in planar geometry**)** at each session (see Fig. 2). It takes running time and cutting values (the most recent that each partition takes) as an input, performs its computations (mathematical calculations that aim basically at minimizing BSOs) and returns new optimized cutting configurations. The partitioning factors are values that allow to model the partition sizes based on the number of points. In fact, our calculation method is simply a sliding mechanism, that moves the cut towards either east or west, north or south based on a previous knowledge of processing workloads and processing times of all processing elements, resulting in approximately similar timing in each of them. Running times are collected using one Spark Accumulator. This method acts basically as a BOS minimizer, at the same time, preserving spatial locality and balancing loads (to some good extent).

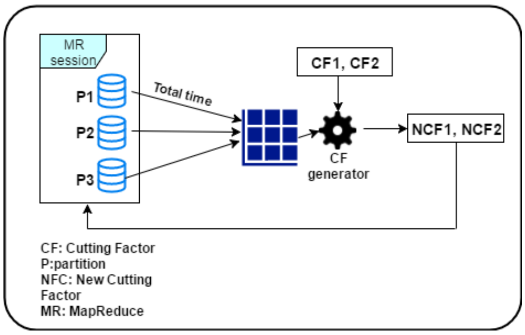

Fig 2. Self-Adaptable Spatial-Aware Partitioner (SASAP).

B. Co-location query optimization

One of the most interesting co-location data mining algorithms is a class of algorithms better known as density-based algorithms (DBSCAN) [20] (with variants such as MapReduce-DBSCAN (MR-DBSCAN) [18] and DBSCAN-MR [9]. This algorithm intrinsically encapsulates join operations, thus we selected it as a complex querying request for testing our SASAP partitioning method. We have added a patch that programmatically partition spatial data sets using SASAP injected within DBSCAN-MR.

In the partitioning phase, imagining the earth flattened out and working recursively, vertical strips are constructed in such a way that boundaries are calculated using medians. For example, to vertically partition the original dataset, we calculate the median of the points within the input RDD. Considering epsilon parameter as an input to this method, we expand each boundary to equal exactly double epsilon (imagine sliding the boundary epsilon to the west and the same to the east, where we apply the Haversine formula for calculating distance), we term this 2eps. RDD elements falling within 2eps are replicated to adjacent partitions. At this stage, each identified partition is assigned an identifier, and points are mapped to key/pair format injecting partition identifiers as keys. Classical DBSCAN is thereafter applied locally to each partition independently. This method guarantees the correctness of the local results by the fact that BSOs are replicated to neighboring partitions, offering a local copy of all spatial objects that are at distance epsilon from the focal point. The algorithm now proceeds normally as DBSCAN-MR however, this takes a huge toll on system resources. We have circumvented this by providing an adaptive step that gradually improve cutting configurations to minimize BSOs. The point where we collect running time of each partition for the current session and feed it back to SASAP for calculating new partitioning configurations for the next session.

IV. Experimental Results

This section reports a wide selection of tests aimed that assess the performance improvement enabled by SASAP.

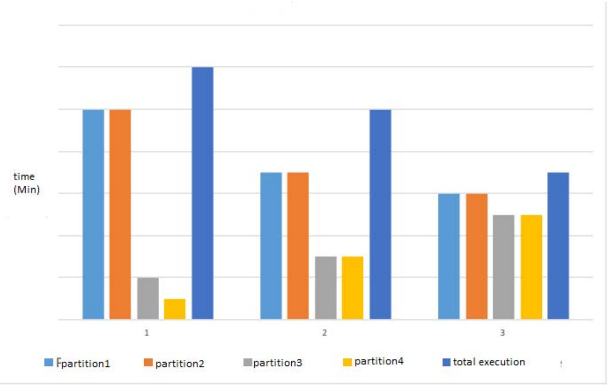

Fig 3. performance improvement within three consecutive iterations.

A. Experimental Setup

Our experimental setup here has utilized Amazon AWS cloud's computing services, specifically Amazon's EC2 service, where five nodes have been used for deployment (one master and four slaves). On each node, Spark 1.6.2 was installed. Our input database consisted of 250,000 (a fraction from ParticipAct total dataset extracted) spatial objects collected through the ParticipAct project. The reason for selecting this amount of data is that they are used for clustering algorithms, which are costly, hence exceeding this number of records means an exponential execution time.

ParticipAct is a project of the University of Bologna, which aims to study the potential, cooperation between citizens, leveraging smartphones as a tool for interaction and interconnection [22]. The project had achieved a large-scale spatial data collection using smartphone's sensors. a dataset extracted from the ParticipAct database containing nearly fourteen million records. The database maintains a user's location data, detected by smartphones, containing sampling timestamps, longitude and latitude, among others.

B. Results and Discussion

We first show the gain of SASAP in terms of running time at each partition of the cloud. Fig 3 depicts the change in the execution times as the cut configurations vary on consecutive iterations. We focused on the execution of the costliest clustering algorithm (i.e. DBSCAN-MR), which encloses both proximity and join queries. And compared the performance (applying our GeoSpark support) with MongoDB classical strategies to the same dataset. Fig 4 depicts the gain elucidating how spark-based support favorably outperforms the storage-oriented system MongoDB. However, it worth noticing that both implementations follow a similar trend where the speedup gain degrades as we increase the data size. This is interestingly healthy as in a density-based clustering, increasing the data size means more clusters, compute-intensive operations and inter-node shuffling. Also, it is expected that spark-based implementation outperforms that of MongoDB counterpart. It is a matter of fact that MongoDB is a storage-oriented ecosystem that is not intrinsically designed for processing big data workloads. It is instead more into big data storage management. Also, the fact that it is storage-oriented makes it bounded by the limitations of IO overheads during the intercourse of a density-based clustering algorithm.

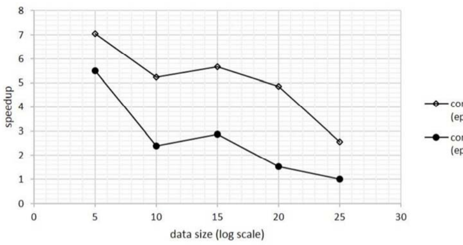

Fig 4. DBSCAN-MR optimization gain.

We have calculated the speedup for each session using a simple formula, adopted from Amdahl's Law [23]; $T_{mongoDB}$ $/T_{Spark}$; where $T_{mongoDB}$ is total running time using MongoDB, whereas T_{Spark} is the total running time using our partitioning and querying strategies based on GeoSpark, which in its turn is based on Spark.

We tested SASAP using two configurations for five sessions each, the first uses *eps* value that is equal to 0.2, *MinPt* value that is equal to 200 as elucidated in Fig. 5, whereas the second configuration uses eps value equals to 0.5, *MinPt* value equals 300 as depicted in Fig. 6.

As noted from the figures, we have obtained a speedup (by orders of magnitude) for all sessions with both configurations, comparing to MongoDB's implementation. However, it worth noticing that the performance gain tends to incline as we increase data size. This is a normal behavior due to the trade-offs between load balancing, SDL and BSOs, as improving one factor may slightly negatively impact others, causing a drop in the overall performance gain. Also, as we increase the number of input spatial objects, this means more results need to be passed to the reducers and it could mean more shuffling for performing join operations in the DBSCAN-MR algorithm. The sweet spot occurs at the middle, with data value approximately equals to 150000, where a balance among trade-offs occurs.

V. Related works

Several works in the relevant literature have provided various optimizations for spatial-aware in-memory big data processing. [24] design a custom density- and spatial-aware big data partitioning approach for digital pathology imaging atop Hadoop. They mitigate BSOs by neglecting them as statistical methods applied in this domain are not affected by BSOs that normally constitute a tiny fraction. Their model also balances loads by employing a mathematical cost model that calculates the data load of each partition so that to balance execution times. [25] breaks into the consortium by introducing a Hadoop-based model that incorporates a custom spatial-characteristics-aware partitioning method.

[26] employs PMR quadtrees and pre-sampling methods for load balancing. They further circumvented BSOs by tagging

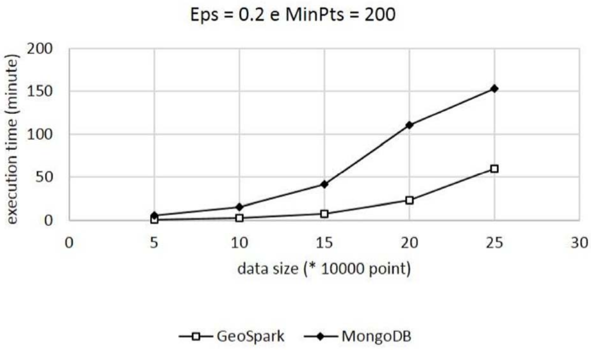

Fig. 5. DBSCAN-MR processing gain with config. Set1.

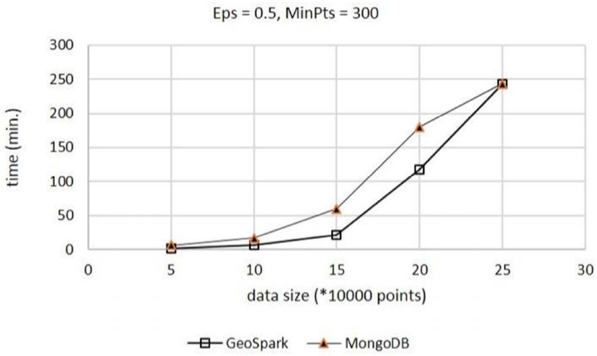

Fig 6. DBSCAN-MR processing optimization gain with config. Set2.

them with multiple indexes, and thereafter applying a postprocessing phase to eliminate duplications.

They further achieved co-locality by spatially ordering the input datasets. Along the same lines, LocationSpark [6] introduces and transparently injected within Spark layers a novel framework that incorporates an adaptable partitioning method. Their method acts by simply gathering statistical information from each partition including running times of previous sessions, thereafter building a cost model and repartition accordingly to diminish stragglers. Co-locality is preserved by sampling input data, thus building a knowledge regarding the real geometrical distribution of objects, thus distributing to co-locate geometrically-close objects. It resolves BSOs by replicating them to adjacent partitions and filtering results thereafter.

Two limitations found in the literature is the fact that most solutions are based on Hadoop, which is much slower than Spark. Further, Spark-based solutions do not adapt their custom partitioning methods for density-based clustering and proximity-alike spatial queries, which render them inconvenient for several interesting smart cities and Industry 4.0 analytics.

VI. CONCLUSIONS AND ONGOING WORKS

Smart city and Industry 4.0 initiatives have new requirements for complex analytics. An avalanche of data is gathered every day, specifically spatially and temporally tagged, thus paving the way for more interesting co-location data mining and proximity analytics. Today, after nearly a decade of optimizations and tuning, big data in-memory and batch-processing systems lack the ability to consider spatial traits that normally appear as pairwise relationships among spatial objects. For example, two persons coexist in city center for a coffee and chatting. Not respecting those characteristics while distributing data loads in cloud computing domains is detrimental to query performance health therein. Thus, optimizations are necessary that consider three conflicting requirements regarding spatial attributes. Those are the classical load balancing in addition to BSOs and co-locality preservation. Those are contradicting in a manner that exaggeratively optimizing one of them may negatively counteracts the benefits of others.

Depending on the case at hand, the problem may render NP-hard and intractable in some contexts, thus we recommend seeking a weighted balance that better trade-off those depending on the ad-hoc requirements.

We have designed a custom partitioning method that outperforms state-of-the-art methods by orders of magnitude when applied in density-based clustering.

One of our works that is underway aims at designing spatial-aware custom partitioning methods that works on fast arriving data streams and adaptable for burst workloads. This depends mainly on a cost model we are developing that can predict the arrival of burst and react conveniently to accommodate the change in a tweakable manner.

ACKNOWLEDGMENTS

This research was supported by the SACHER (Smart Architecture for Cultural Heritage in Emilia Romagna) project funded by the POR-FESR 2014-20 (no. J32I16000120009) through CIRI.

REFERENCES

[1] C. R. D. Rolt, R. Montanari, M. L. Brocardo, L. Foschini, and J. d. S. Dias, "COLLEGA middleware for the management of participatory Mobile Health Communities," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, 2016, pp. 999-1005.

[2] E. Gomes, M. A. R. Dantas, D. D. J. d. Macedo, C. D. Rolt, M. L. Brocardo, and L. Foschini, "Towards an Infrastructure to Support Big Data for a Smart City Project," in *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2016, pp. 107-112.

[3] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM,* vol. 51, pp. 107-113, 2008.

[4] K. Banker, *MongoDB in Action*: Manning, 2012.

[5] F. Wang, A. Aji, and H. Vo, "High performance spatial queries for spatial big data: from medical imaging to GIS," *SIGSPATIAL Special,* vol. 6, pp. 11-18, 2015.

[6] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref, "LocationSpark: a distributed in-memory data management system for big spatial data," *Proc. VLDB Endow.,* vol. 9, pp. 1565-1568, 2016.

[7] J. M. Patel and D. J. DeWitt, "Partition based spatial-merge join," *SIGMOD Rec.,* vol. 25, pp. 259-270, 1996.

[8] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, *et al.*, "Hadoop GIS: a high performance spatial data warehousing system over mapreduce," *Proc. VLDB Endow.,* vol. 6, pp. 1009-1020, 2013.

[9] B. R. Dai and I. C. Lin, "Efficient Map/Reduce-Based DBSCAN Algorithm with Optimized Data Partition," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 59-66.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," presented at the Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, Boston, MA, 2010.

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley*, et al.*, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," presented at the Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, San Jose, CA, 2012.

[12] M. T. Özsu and P. Valduriez, "Distributed Database Design," in *Principles of Distributed Database Systems, Third Edition*, ed New York, NY: Springer New York, 2011, pp. 71-129.

[13] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Inf.,* vol. 4, pp. 1-9, 1974.

[14] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer, "Space-filling curves and their use in the design of geometric data structures," *Theoretical Computer Science,* vol. 181, pp. 3-15, 1997/07/15/ 1997.

[15] I. Kamel and C. Faloutsos, "Hilbert R-tree: An Improved R-tree using Fractals," presented at the Proceedings of the 20th International Conference on Very Large Data Bases, 1994.

[16] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," Ottawa, Canada, IBM1996.

[17] J. Yu, J. Wu, and M. Sarwat, "GeoSpark: a cluster computing framework for processing large-scale spatial data," presented at the Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, Washington, 2015.

[18] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data," *Frontiers of Computer Science,* vol. 8, pp. 83-99, 2014.

[19] G. Nam, D. Kim, J. Lee, H. Y. Youn, and U.-M. Kim, "Efficient batch processing of proximity queries with MapReduce," presented at the Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication, Bali, Indonesia, 2015.

[20] M. Ester, H.-P. Kriegel, r. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," presented at the Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, 1996.

[21] A. Garaeva, F. Makhmutova, I. Anikin, and K. U. Sattler, "A framework for co-location patterns mining in big spatial data," in *2017 XX IEEE International Conference on Soft Computing and Measurements (SCM)*, 2017, pp. 477-480.

[22] G. Cardone, A. Corradi, L. Foschini, and R. Ianniello, "ParticipAct: A Large-Scale Crowdsensing Platform," *IEEE Transactions on Emerging Topics in Computing,* vol. 4, pp. 21-32, 2016.

[23] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*: Morgan Kaufmann Publishers Inc., 2011.

[24] A. Aji, F. Wang, and J. H. Saltz, "Towards building a high performance spatial query system for large scale medical imaging data," presented at the Proceedings of the 20th International Conference on Advances in Geographic Information Systems, Redondo Beach, California, 2012.

[25] R. T. Whitman, M. B. Park, S. M. Ambrose, and E. G. Hoel, "Spatial indexing and analytics on Hadoop," presented at the Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Dallas, Texas, 2014.

[26] R. C. Nelson and H. Samet, "A consistent hierarchical representation for vector data," *SIGGRAPH Comput. Graph.,* vol. 20, pp. 197-206, 1986.