2019 IEEE Symposium on Computers and Communications (ISCC)

Clustering of Spatial Data with DBSCAN: An Assessment of STARK

Paolo Bellavista
DISI
University of Bologna
Bologna, Italy
paolo.bellavista@unibo.it

Mattia Campestri
DISI
University of Bologna
Bologna, Italy
mattia.campestri@studio.unibo.it

Luca Foschini
DISI
University of Bologna
Bologna, Italy
luca.foschini@unibo.it

Rebecca Montanari
DISI
University of Bologna
Bologna, Italy
rebecca.montanari@unibo.it

***Abstract*—The ever-increasing diffusion rate of mobile devices, able to continuously gather sensing data, creates favorable conditions for the development of smart city infrastructures. In this field the analysis of spatial data plays a pivotal role, due to the relevance they assume in urban scenarios. To satisfy this need, the usage of large distributed computing infrastructures comes into play, supported by efficient frameworks, such as Apache Spark, one of the most relevant platforms to date. However, in order to better take advantage of data and computing resources, it is also necessary to have at disposal flexible and easy-to-use specialized instruments, granting domain specific capabilities for the analysis of spatial data. This paper focuses on a novel framework for processing of spatial data called STARK, giving an overview of its functionalities and presenting an in-depth assessment study of its performances when implementing spatial data clustering, namely DBSCAN. In particular, we focus on two implementations, called MR-DBSCAN and NG-DBSCAN. Of the latter we introduced an implementation in STARK, in order to enrich the framework and to test its capabilities.**

Keywords—spatial data, big data, Spark, clustering

I. INTRODUCTION

The widespread availability of sensors and mobile devices enables collecting and subsequent processing of large amounts of data to gain a better knowledge of the needs and trends of the population, in order to be able to create the right conditions for the diffusion of smart city services and infrastructures. The analysis of spatial data is especially important for these scenarios, since most of the collected data also includes geographic coordinates, making it possible to get deeper insights regarding the spatial distribution of the information gathered from mobile devices. In the context of big data analysis, the availability of efficient and flexible processing frameworks and tools becomes crucial. One of the most popular distributed computing frameworks is Apache Spark, an open source cluster computing engine for data analysis. The core aspects of the Spark framework are its in-memory computing model, which allows it to achieve excellent performance through the definition of the Resilient Distributed Dataset (RDD), which is a data structure that is partitioned across the nodes of a cluster for parallel processing, and a high-level API, which grants great usability and easy access to the framework's features. However, despite its great support for general data types, Spark doesn't offer native support for spatial data, thus requiring data analysts interested in this kind of applications to resort to self-made ad-hoc solutions or to other frameworks built on top of Spark. Among these frameworks, a solution of particular interest can be found in a project called STARK (Spatio-Temporal Data Analytics on Spark) [1], a module built on top of Spark which offers a complete set of spatial and temporal functions and abstractions to efficiently and easily process geographic and temporal data. For this work we decided to analyze the platform and its capabilities, with a particular focus on spatial clustering algorithms, which represents one of the main functionalities in spatial data analysis scenarios. After this, we also take into consideration a new clustering algorithm called NG-DBSCAN, introduced by [2], of which we added an implementation to STARK in order to provide the framework with an additional clustering method, capable of achieving good efficiency with large datasets, with the intent to address the possible scalability problems given by the growth of the amount of processed data. We also consider the approach explored in previous works [5] by including a dynamic self-adapting partitioning method for MR-DBSCAN clustering.

II. BACKGROUND ABOUT STARK

Spark's flexibility and performances make it the ideal platform for several different data analysis applications. However, it doesn't offer native support for spatial data, requiring users to adopt additional frameworks, offering support for spatial data and the related functionalities. Among these frameworks, in this work we focus on STARK, which stands for Spatio-Temporal Data Analytics on Spark. STARK was developed by [1] on top of Spark and it provides all the most useful operations in the field of spatial data analytics, such as partitioning, indexing, filtering and join, nearest neighbors, clustering and skyline, as well as support for temporal data.

As described in [1], all functionalities are seamlessly integrated into the Spark framework and aim to take advantage of its parallel computation model. This integration is obtained through a simple and intuitive DSL (domain specific language), usable in any Spark program written in Scala, which allows to maintain the usual Spark workflow, based on transformations on RDDs. The architecture of the framework is illustrated in Fig. 1.

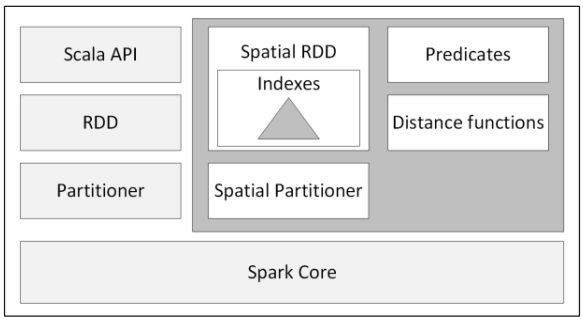

Fig. 1. STARK architecture (as shown in [1]).

978-1-7281-2999-0/19/$31.00 ©2019 IEEE

A. Spatial and Temporal Abstractions

STARK introduces abstractions to represent spatio-temporal data and allow ease of use and interoperability with Spark. The main abstractions are the STObject and SpatialRDD classes. An STObject represents an object with its spatial and temporal components, contained in the respective geo and time fields. To only consider the spatial component of an object, the time field can be left empty. Relationships between instances of STObject can be tested using the following operations, provided by the STObject class: intersect, contains, containedBy, which respectively test if two instances intersect spatially and/or temporally, if the first instance contains the other or if it is instead contained by it, spatially and/or temporally.

The second abstraction is called SpatialRDD and it represents a PairRDD of STObject and the relative payload, consisting in the additional information of one element of the dataset. A SpatialRDD is created by implicitly converting an RDD[(STObject, V)] that can contain objects of different geometries, such as points, polygons or any geometry that can be represented as WKT. The conversion is transparent to the user, so that the only class to be explicitly instantiated is STObject. This way, the RDD now exposes the operations offered by STARK.

B. Operations

The STARK framework offers support for spatio-temporal data analysis applications providing several operations for datasets of spatial or temporal nature. A SpatialRDD exposes functions for filtering and joining of the RDD with another SpatialRDD, according to the specified predicate. The next available operation is nearest neighbor, which allows to find the k nearest neighbors to the given element. The computation of the neighborhood is performed in parallel on different nodes, to each of which is assigned a partition of the dataset to leverage the capabilities of Spark. This results in the computation of a number of k neighborhoods of the element equal to the number of partitions, which finally get sorted to find the actual k nearest neighbors.

Another fundamental operation for spatial data analysis is data partitioning, which consists in dividing the data space into multiple portions and assigning all elements to one of them. This allows to improve the efficiency of computations by exploiting the spatial locality of data items in filter and join operations by discarding partitions which cannot possibly contain join or filter candidates. STARK provides two spatial partitioners, grid partitioner and binary space partitioner. The grid partitioner divides the data space applying a grid of equally sized rectangular cells and assigns each point to the containing cell. The binary space partitioner addresses the problem of data skewness, which can affect the previous partitioning scheme causing only few partitions to contain most of the points leaving other partitions empty or poorly populated. In Spark this results in the overload of those executors who are assigned the most crowded partitions, while other executors perform little work. The binary space partitioning algorithm starts by dividing the data space into small quadratic cells and computing the number of points contained in each cell. Next, the space is divided into two partitions containing approximately the same number of points, which is calculated using the information produced by the first step. The two partitions are then recursively split to produce the most balanced partitions possible until a partition contains less than the maximum cost defined as a parameter of the algorithm or until a partition is as small as a single quadratic cell. This partitioning scheme will create partitions containing almost the same amount of points.

The STARK framework also offers clustering operations. Clustering consists in the identification of groups of points close to each other. In particular, the clustering algorithm chosen by the authors of STARK is a density-based algorithm called MR-DBSCAN, a distributed version of DBSCAN that leverages Spark's parallel computing model. The algorithm detects densely populated regions and groups close points together to form clusters. MR-DBSCAN partitions the dataset using one of the previously described partitioners and executes the DBSCAN algorithm on each partition in parallel, to finally aggregate the local results into a single global result.

One more useful functionality the framework provides is data indexing, used by STARK to create an index of each partition to further improve performance. The index structure is an R-tree and multiple indexing modes are available: no index, live indexing and persistent indexing. In the first mode no index is used, while the difference between live and persistent indexing is that the latter allows to reuse the index structure for consecutive operations and to persist the index to disk for further usage in later executions, avoiding the creation cost.

Given the completeness and the usability of the framework, we decided to further analyze it, with particular interest in its clustering functionality.

III. Distributed DBSCAN Solutions

DBSCAN [3] is a density-based algorithm which works by classifying the points of the dataset into three different categories: core points, border points and noise, based on the other points in their neighborhood. The classification is defined by two parameters, ε and MinPoints. The former, ε, represents the maximum distance between two points to consider them to be neighbors, while the latter, MinPoints, determines the minimum number of points in one point's neighborhood to recognize it as a core point. The computation is carried out by calculating each point's ε-neighborhood and accordingly classifying each point as core, border or noise. Core points define clusters together with all other core and border points reachable from them. DBSCAN provides some useful properties, such as the ability to find arbitrarily shaped clusters and the notion of noise, but it also has some downsides. Specifically, it becomes more inefficient as the dataset grows in size, making the algorithm little scalable.

A. MR-DBSCAN

To address this problem, the MR-DBSCAN algorithm, introduced by [4], distributes the computation on different nodes of a cluster by partitioning the data and computing the local DBSCAN algorithm on each partition in parallel. This causes a reduction of the size of the data on which DBSCAN is executed, which results in a big performance improvement. To achieve the same clustering as the normal DBSCAN algorithm, some modifications have to be made. In particular, it is necessary to expand the partitions of a distance equal to ε in all directions, in

order to make them overlap with each other. Points in intersection regions will be assigned to all overlapping partitions for local DBSCAN execution. In this way, after the local clustering phase, partial results are aggregated in a merging phase where local clusters are combined together if they have core points in common in overlapping regions. Since the computation is performed in parallel, the total execution time is dominated by the longest local DBSCAN execution, which means that the performance is strictly dependent on the partitioning strategy. A good partitioning scheme manages to produce similarly large partitions, whose execution completes in almost the same amount of time.

B. NG-DBSCAN

NG-DBSCAN is an approximate distributed density-based clustering algorithm, developed by [2] with the intent of providing a more efficient, scalable and versatile implementation of the DBSCAN algorithm. The algorithm was conceived to be suitable for multiple types of data, including spatial and textual data. In this work we are interested in the analysis of its performance on spatial datasets. NG-DBSCAN avoids the cost of calculating the exact ε-neighborhood of each point, as required by the classic DBSCAN algorithm, by calculating approximated ε-neighborhoods. The execution is divided in two phases: the first one iteratively creates an ε-graph, a data structure whose elements are the points of the dataset and the respective approximated ε-neighborhood; the second phase takes this data structure as input and executes the density-based clustering algorithm by researching connected components in the graph, which finally represent clusters.

In the first phase, the graph is calculated iteratively using an auxiliary structure called neighbor graph, a directed graph connecting points to each other. The neighbor graph is initialized by connecting each point to k random points and is modified at each iteration by exploring for each point the nodes which are at most two hops away from it in the graph. If the distance, calculated using the provided distance function, is smaller than the distance between the considered point and one of the k neighbors connected to it, the neighbor is replaced by connecting the point to the new found one. In this way, at each iteration, the neighborhood of each point becomes more accurate, containing increasingly near points than in the previous iteration. Furthermore, when two nodes whose distance is less than ε are found, they are added to the ε-graph. The algorithm also introduces a method to reduce the time required by each iteration, which consists in removing a node from the neighbor graph as soon as it reaches kMax connected points in the ε-graph. This is done because the algorithm has the final objective of finding clusters and therefore core points, so, as soon as a node has a sufficient number of neighbors in its ε-neighborhood it can be considered to have enough information to be identified as a core point and it can be ignored for further computation in the first phase. The first phase continues iterating until the number of iterations reaches the threshold set as parameter or the termination condition is verified. The termination condition is calculated using two parameters: the number of active nodes (the nodes which have not yet been removed from the neighbor graph) and the fraction of nodes which were removed in the last iteration. This mechanism aims to reduce execution times where further iterations would not affect the result in a significant way.

The second phase takes the ε-graph as input and performs the actual clustering operation by executing lookups on the graph. Being the algorithm graph based, a cluster corresponds to a connected component in the ε-graph, where all core points of a cluster are connected to all other core points in their ε-neighborhood. Based on their neighborhood, points are distinguished between core points, border and noise, and the final clustering is created.

Both execution time and result quality are determined by different configuration parameters, the most important of which are k and kMax. The ideal parameter configuration depends on both the dataset and DBSCAN parameters ε and MinPoints, and the best tuning may require some empirical testing with different configurations.

IV. Distributed DBSCAN in STARK

As already mentioned, being clustering one of the main operations needed in a spatial data analysis framework, STARK already provides an implementation of MR-DBSCAN. Additionally, we added an implementation of the NG-DBSCAN algorithm, to extend the clustering capabilities of the framework.

A. MR-DBSCAN Implementation

STARK's implementation of MR-DBSCAN allows the usage of different user-defined distance functions and provides two different partitioning algorithms, a grid partitioner and a binary space partitioner, as described previously.

Additionally, in this work, we introduced a new vertical partitioner, which divides the space into vertical parallel slices. To optimize this partitioning scheme, we also introduced an auxiliary module to automatically balance execution times of different partitions between consecutive runs of MR-DBSCAN. Both additions are based on the work of [5]. The number of vertical partitions, as well as the distance between their bounds can be arbitrarily defined by the user to obtain the desired level of parallelism. The new bounds are calculated in a way that aims to balance execution times of all partitions, iteratively converging to similar times after a certain number of executions of the clustering algorithm. The newly calculated bounds are also saved to a user-defined location, giving the possibility to use previously optimized split configurations for new datasets, which can prove beneficial if the datasets originate from the same sources (e.g. both datasets are relative to the same analysis campaign), since it allows to avoid execution with non-optimized splits by directly using previously optimized splits.

The usage of the algorithm requires the definition of the two parameters ε and MinPoints, as well as the additional parameters regarding the chosen partitioning scheme, which are the number of partitions for each dimension for the grid partitioner, the maximum amount of points per partition for the binary space partitioner, and the vertical splits for the vertical partitioner. The result of the execution is a DBScanModel, containing all elements of the dataset as a collection of ClusterPoint, a class which represents a point and all related clustering information,

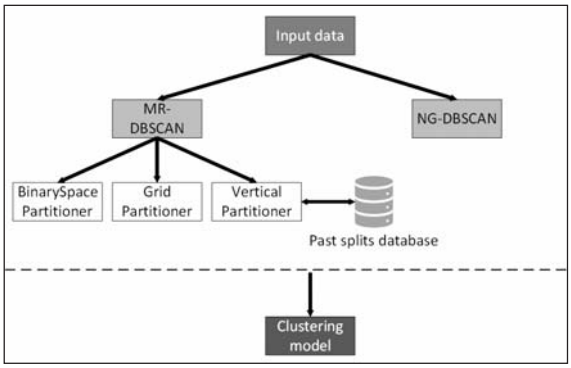

Fig. 2. Clustering system architecture.

such as the cluster id and the category of point (core point, border point, noise).

B. NG-DBSCAN Implementation

The implementation of the NG-DBSCAN algorithm we added reflects the one introduced by [2] and is simplified to be specialized for spatial data. The main classes used to represent the graph are Neighbor and NeighborList, in which nodes are sorted based on the distance value and only the k nodes with the lower distance value are kept. Execution parameters are loaded from a configuration file on start and then maintained in an ENNConfig class. Computation is carried out in the same way as described before and the final result is returned in the form of an NGDBScanModel, where the clustered points are collected in the form of ClusterPoint, in the same way as in the MR-DBSCAN implementation. This allows for good integration with STARK's environment and abstractions.

The architecture of the resulting system is represented in Fig. 2, which shows the possibility to choose between MR-DBSCAN and NG-DBSCAN and to define the partitioning scheme to be used in the case of MR-DBSCAN. As mentioned, both algorithms return a clustering model containing the resulting clustered points.

V. Experimental Results

In this section we present a series of tests aimed at comparing the performances of the clustering algorithms previously described. All tests were executed in a distributed environment, using Amazon's AWS EC2 service. The tests were run on a cluster of 5 nodes (1 master and 4 workers), each equipped with 8 cores, 32GB of memory and a 30GB EBS volume. We used two different datasets: the first one is relative to the ParticipAct [7] project, a crowdsensing project by the University of Bologna, and contains the geographic positions of users, collected from mobile devices in the area of Bologna, while the second one represents the positions of taxis recorded in the city of Shanghai [6]. For both datasets we considered subsets of different dimensions to evaluate the scalability of the algorithms as the size of the processed points increases.

The parameters we consider for this evaluation are the total computing time needed by the algorithms to complete their execution and the resulting recall between the clustering obtained with NG-DBSCAN and with MR-DBSCAN. Recall measures the fraction of point pairs belonging to the same cluster in NG-DBSCAN's clustering result and in MR-DBSCAN's result, assuming MR-DBSCAN as reference. Given the nature of the data, as distance function we opted to apply the Haversine formula, which represents the great-circle distance between points on a sphere.

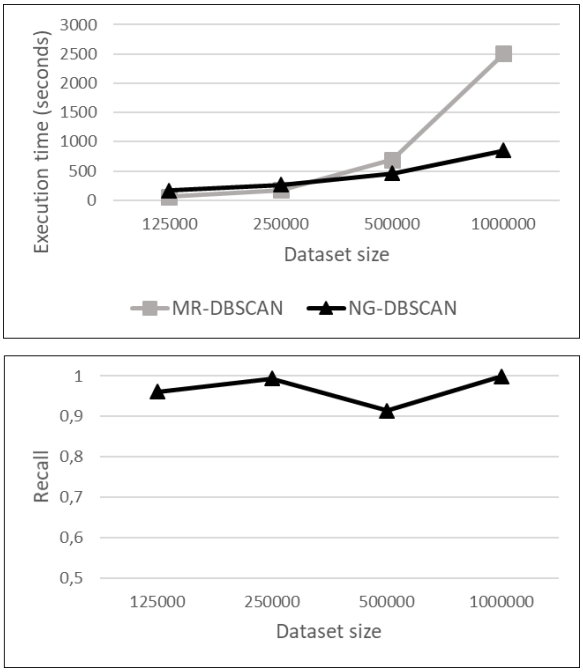

Fig. 3. Execution time and recall for ParticipAct dataset (first test set).

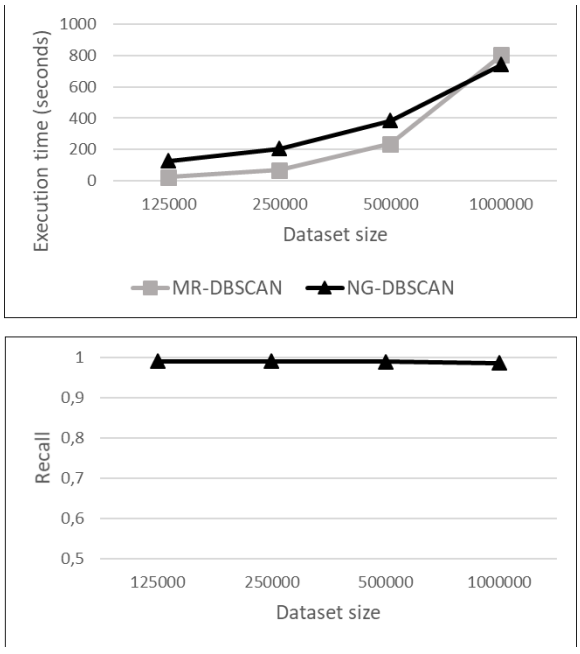

Fig. 4. Execution time and recall for taxi dataset (first test set).

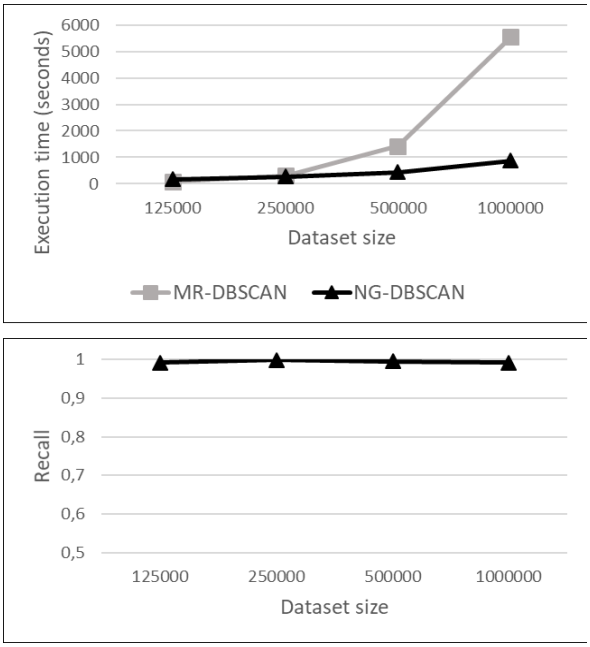

Fig. 5. Execution time and recall for ParticipAct dataset (second test set).

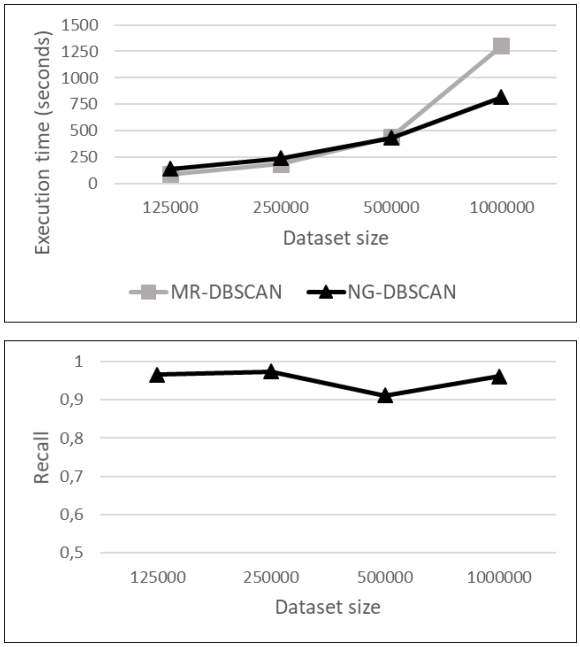

Fig. 6. Execution time and recall for taxi dataset (second test set).

A. *First Test Set*

In this first set of tests, we set DBSCAN's parameters as follows: ε = 0.05, MinPoints = 40. This results in a large number of clusters composed by points at a relatively close distance. Because of the small value of the ε parameter, using the BSP partitioner results inconvenient, since it has to divide the whole data space into a grid of cells of size equal to 2*ε and then calculate the amount of points contained in each cell, which adds an overhead to the execution time in the case of very large geographic areas, such as the one relative to the taxi dataset, and a small ε value. We therefore employed the vertical partitioning strategy, dividing the data into 64 partitions (2 for each core). As regards NG-DBSCAN, the parameters were set as follows: k = 10, kMax = 120, maximum number of iterations = 6, termination active nodes = 0.7, termination last removed nodes = 0.01. The results we obtained are represented in the following charts.

As shown in Fig. 3 and Fig. 4, both algorithms obtain good results, with MR-DBSCAN achieving a better execution time than NG-DBSCAN with smaller datasets, being outperformed by NG as the number of processed points increases over a certain amount, with NG-DBSCAN maintaining a less steep increase in completion time. It can be observed that for subsets of the same size, MR-DBSCAN achieves better times with the Shanghai taxi dataset then with the ParticipAct dataset. This can be attributed to the fact that in the latter dataset points are more densely concentrated in a single area, resulting in smaller and closer partitions, to the point where the ones covering the densest area, by getting extended, cover a great portion of the neighboring partitions and therefore come to contain a large number of points, which increases the total execution time. The taxi dataset, on the other hand, covers a larger geographic area and points are more evenly distributed in space, which causes the resulting partitions to be wider and overlapping areas cover a smaller percentage of the neighboring partitions. NG-DBSCAN, on the other hand, achieves similar times for datasets of equal size, as it doesn't apply a spatial partitioning strategy.

As shown in the charts representing recall values, NG-DBSCAN manages to obtain very good results, being able to find almost exactly the same clusters as MR-DBSCAN.

B. *Second Test Set*

To better compare the two algorithms, we also tested them in a different scenario, setting DBSCAN's parameters to: ε = 0.2, MinPoints = 40. This results in fewer clusters of bigger size and containing a larger amount of points. The larger value of ε allows to use the binary space partitioner, since it will create a smaller number of cells, therefore reducing the overhead introduced by the partitioning phase. The maximum number of points per partition was set so that the resulting number of partitions is between two and three times the number of cores present in the cluster, in order to continuously process partitions of smaller size. NG-DBSCAN's parameters were kept the same. Fig. 5 and Fig. 6 show how the algorithm performed in this case.

As regards execution times, the results confirm the trend of the first set of tests, with MR-DBSCAN performing slightly better than NG-DBSCAN until a certain number of elements is reached, after which NG proves more efficient, thanks to its more gradual increasing trend. Once again, the more even distribution of the taxi dataset, along with its larger geographical extension, allow for better partitioning, granting better results compared to the ParticipAct dataset.

As mentioned before, the larger value of ε causes clusters to be larger and more populated. In this scenario, NG-DBSCAN still achieves a good recall, which always stays above 0.9, confirming the viability of the algorithm.

VI. Related Work

Many different methods have been proposed in literature for clustering of spatial data. Among these, one of the most widely adopted is MR-DBSCAN, introduced in [4]. The algorithm leverages the capabilities offered by large distributed computing frameworks such as Spark and has allowed to increase the scale of the quantity of processed data.

New approaches, such the method adopted by [2] in NG-DBSCAN, further push towards the improvement in performances and scalability. The addition of an implementation of NG-DBSCAN to the STARK framework aims therefore at addressing the need for a highly efficient clustering algorithm for increasingly large datasets. As regards the introduction of the new vertical adaptive partitioning method, we referred to the previous work conducted in [5], which aimed at creating an efficient partitioning method, capable of optimizing execution time by adapting throughout subsequent iterations of the algorithm.

As referred in that work, also LocationSpark [8] introduced a novel framework incorporated between Spark layers that provides an adaptable partitioning method, which gathers statistical information from partitions, including running times of previous sessions, to then build a cost model which is used for better repartitioning. Also related to [5] is the research for an efficient and easy-to-use framework for spatial data analysis, such as STARK, with a focus on the usability in the scenario of smart cities and crowdsensing, as described in the work of [7] presenting the ParticipAct project.

Finally, several works in the literature have provided various optimizations for spatial-aware in-memory big data processing. [9] designed a custom density- and spatial-aware big data partitioning approach for digital pathology imaging atop Hadoop. Their model also balances loads by employing a mathematical cost model that calculates the data load of each partition in order to balance execution times. [10] introduced a Hadoop-based model that incorporates a custom spatial-characteristics-aware partitioning method. Compared to such Hadoop-based solutions, the adoption of the Spark framework allows to achieve better performance, making it possible to create more efficient frameworks.

VII. Conclusions and Future Work

The capillary diffusion of mobile sensors today, together with the availability of distributed computing infrastructures, made it possible to create new solutions in the field of smart city applications. New powerful tools for data analysis and processing have emerged, along with useful domain specific frameworks, allowing analysts to obtain insight from data in an efficient and flexible way.

With the great interest gained by spatial data in smart city infrastructures, a specialized and optimized framework targeting domain specific problems becomes crucial in order to take advantage of all available data and resources in the best way possible. Focusing on this kind of frameworks, we found STARK to be a satisfying candidate for spatial and temporal data analysis, granting the most requested functionalities in the domain of interest, in a well-integrated and easily usable way. By focusing, in particular, on clustering capabilities, we conducted a series of tests in order to evaluate the usability of both MR-DBSCAN and NG-DBSCAN. Our results prove the efficiency and usability of both algorithms, showing their applicability with different datasets and different parameters.

Encouraged by the obtained results, we are now pursuing new ongoing research directions. On the one hand, we intend to further analyze the possibilities to adopt novel and increasingly efficient clustering methods and related optimizations. On the other hand, we are continuing to follow the highly interesting evolution in the area of spatial data analysis in order to take the best advantage possible from the opportunities provided by the advancements in this field by bringing them into Smart Cities management support systems.

References

[1] S. Hagedorn, P. Götze, K.-U. Sattler, “The STARK framework for spatio-temporal data analytics on Spark,” BTW, pp. 123-142, 2017.

[2] A. Lulli, M. Dell’Amico, P. Michiardi, L. Ricci, “NG-DBSCAN: scalable density-based clustering for arbitrary data,” PVLDB, vol. 10, no. 3, pp. 157–168, 2016.

[3] M. Ester, H. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD’96), 1996, pp. 226–231.

[4] 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, no. 1, pp. 83–99, Feb. 2014.

[5] I. M. Aljawarneh, P. Bellavista, A. Corradi, R. Montanari, L. Foschini, and A. Zanotti, “Efficient spark-based framework for big geospatial data query processing and analysis,” 2017 IEEE Symposium on Computers and Communications (ISCC), Jul. 2017.

[6] Dataset from Wireless and Sensor networks Lab (WnSN), Shanghai Jiao Tong University:

http://wirelesslab.sjtu.edu.cn/taxi_trace_data.html

[7] G. Cardone, A. Corradi, L. Foschini, and R. Ianniello, “ParticipAct: A Large-Scale Crowdsensing Platform,” IEEE Transactions on Emerging Topics in Computing, vol. 4, no. 1, pp. 21–32, Jan. 2016.

[8] 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,” Proceedings of the VLDB Endowment, vol. 9, no. 13, pp. 1565–1568, Sep. 2016.

[9] A. Aji, F. Wang, and J. H. Saltz, “Towards building a high performance spatial query system for large scale medical imaging data,” Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL ’12, 2012.

[10] R. T. Whitman, M. B. Park, S. M. Ambrose, and E. G. Hoel, “Spatial indexing and analytics on Hadoop,” Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - SIGSPATIAL ’14, 2014.