Locality-Preserving Spatial Partitioning for Geo Big Data Analytics in Main Memory Frameworks

Isam Mashhour Al Jawarneh, Paolo Bellavista , Antonio Corradi, Luca Foschini , Rebecca Montanari

Dipartimento di Informatica – Scienza e Ingegneria, University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{isam.aljawarneh3, paolo.bellavista, antonio.corradi, luca.foschini, rebecca.montanari}@unibo.it

Abstract*—The easily reachable IoT edge devices have caused the accumulation of vast amounts of geo-referenced data traces that can help in performing deep insightful analytics. Geospatial data in real geometries are normally clumped into batches and has strong autocorrelation properties which can be exploited in discovering interesting insights. Current plain Cloud computing frameworks are not attuned to the shape of data. Most importantly, data splitting is an important precursor in data parallelization mechanisms. Current systems mostly focus on general data workloads, thus are giving attention mostly to load balancing while splitting the data to Cloud computing resources. However, many benefits can be reaped by being attuned to the spatial characteristics while distributing the data, thus striking a plausible balance between load balancing and spatial data locality preservation normally leads to achieving better time-based QoS goals, which then leads to an optimized provisioning of Cloud computing resources. In this paper, we have designed a spatial batch processing engine that comprises a custom spatial data locality aware partitioning method for disseminating spatial data loads in Cloud computing clusters. We have also extended a state-of-art benchmark density-based clustering method that is known as DBSCAN-MR and implemented a standard compliant prototype on top of a best-in-breed de facto Cloud-based main memory processing framework, Apache Spark. Our results show that our partitioning method with the associated spatial query optimizers can achieve gains that significantly outperform baselines

Keywords—* *spatial join, Spark, DBSCAN, data partitioning, smart city

I. Introduction

City planning, urban computing, and participatory healthcare services [1] are just few examples of innumerable dynamic application scenarios that require highly scalable architectures for the management of big data in the arena of IoT. Traditional beefed-up server central-based computing paradigms could not catch up with the exponential increase in the data arrival rates. This has led to the emergence of parallel distributed computing paradigms for the challenging management of avalanches of big data loads, deployed mostly in a Cloud or in on-premises computing Clusters. For example, MapReduce-based [2] systems. Those frameworks unquestionably need to step over two pillars to deliver right insights to decision makers. The first pillar is data partitioning, where huge data loads are split into the distributed computing resources of the Cloud, aiming at speeding up the production process by depending on collaboration by exploiting the processing capacities of multiple computing nodes. What then remains incumbent is applying a query optimizer that selects the best query plan and apply it to the partitioned data. It is then apparent that despite data splitting is not a target per se, it constitutes a pre-stage that is of a paramount importance and has a utility in improving (or conversely deteriorating) the data parallel processing in Cloud computing deployments. It is often highly desirable to lower the cross-node shuffling of data in those deployments, aiming at achieving better time-based QoS goals such as lowering the latency and achieving a higher throughput. The auto-scaling methods that are offered by Cloud-based frameworks such as Apache Spark [3] allow dynamic allocation of computing resources (i.e., overprovisioning computing and/or storage resources). By those measures, they aim basically at keeping up with the fluctuation of data arrival rates. This requires then the redistribution of data between the processing nodes in what is known as *live data migration*. Thus, shuffling, possibly, a huge amount of data in the network. All in all, more data shuffling can lead to high congestion in the network, which can at some point slow down the overall system performance. An important target in Cloud computing environments would then be minimizing data shuffling, which can be achieved by a custom spatial data partitioning method that focuses, most importantly, on achieving a spatial co-locality awareness by then clumping geographically-nearby spatial objects into same Cloud processing nodes. However, the focus of the current literature has been geared so far mostly toward load balancing during the partitioning stage, which, at a loosely manner, can be thought of as sending roughly an equivalent number of data loads to the processing nodes of the Cloud. Therefore, neglecting the real-geometrical spatial distribution and statistics of data, such as the skewness, where geo-referenced points are normally clumped into few areas in the embedding real geometries [4, 5] .

The contributions of this paper are the following. First, we design a custom spatially-attuned partitioning method that achieves plausible degree of spatial data locality preservation in Cloud cluster computing deployments. It does so by sending geometrically-nearby spatial objects to same partitions in the computing Cluster. We also consider load balancing to a lesser extent. Thereafter, we design a spatial query optimizer that appropriately exploits the newly added partitioning method in responding to some of the most expensive and common spatial analytics in today's highly dynamic and scalable application scenarios. For example, density-based clustering. Also, we have implemented a representative density-based clustering algorithm and transparently incorporated it within the layers of a trending spatial Spark-based library known as Spark's Magellan [1]. We collectively call our system that encompasses those components under its umbrella as SpatialBPE (short for Spatial Batch Processing Engine), indicating the fact that it is optimized for batch in-memory processing frameworks.

The remainder of the paper is organized as follows. We first discuss brief primers and related background. We thereafter step through the system design perspectives, explaining the enclosed algorithms. In what follows, we discuss the results that we have obtained comparing our new methods with representative baselines. We conclude the paper by drawing some remarks and recommending some future research perspectives.

II. Background and Theoretical Grounding

In this section, we discuss noteworthy grounding foundations for the ensuing discussions.

A. Primer on Distributed Data Partitioning

In distributed computing environments such as Cloud computing or an on-premises Cluster computing deployment, data parallelization means splitting data to computer Cluster nodes, thereafter, sending an operator instance to each node so that it is applied there locally to the partitions of data within that node. Local results are then combined into a single piece which is then served to the user or forwarded as an intermediate result to be ingested by sequentially connected operators downstream (i.e., as part of a directed acyclic graph, DAG, that is composed of all the stages required for answering a specific query).

An integral part in the process is data splitting (a.k.a. partitioning). Selecting a highly qualified partitioning scheme has a utility in determining the overall performance of the underlying Cloud-based distributed processing system. Given that current big data is mostly geo-referenced [6] , traditional partitioning schemes should be revisited and revised to consider the spatial characteristics of the data. Most importantly, we consider the spatial data skewness, where for some surveys spatial data are clumped into few areas in real geometries. In the next subsection, we explain why neglecting spatial co-locality while distributing the spatial data in Cloud could potentially bog down the system performance.

B. Spatial Data Partitioning Goals

An intrinsic problem of the current distributed processing frameworks is that they are generic and not specifically designed to handle geospatial data. A fact that can deteriorate the benefits of parallelization at times, especially in cases where spatial data is highly skewed, which is a common case that is not unheard of. A well-performing partitioning scheme should aim at minimizing the network shuffling during query response. This can be achieved by better trading off few partitioning goals. We have identified three conflicting partitioning goals that are related to geospatial data partitioning, which have utilities in determining the overall QoS of big spatial data query processing. Those goals are, i) Load balancing, which alone is insufficient for a QoS aware processing in patchy spatial data loads distributions where data tends to be highly skewed [7, 8]. Also, spatial data often show autocorrelations, where nearby objects are more related than others that may fall farthest [9]. We refer to this spatial characteristic as ii) Spatial Data Locality (SDL). Preserving SDL while splitting data to processing nodes in a Cloud has a utility that determines, at times, the overall performance of the system, by basically minimizing cross-partition spatial data access operations and shuffling. As a case of example, proximity-alike spatial queries (such as k-Nearest Neighborhood, or kNN for short) often seek accessing spatial points that are clumped into patches in real geometries. Sending geographically-close-by points to same worker nodes can reduce the cross-partition access caused by shuffling data over the network. iii) Boundary Spatial Objects (BSO) minimization. Considering a planar geometrical earth as a flat surface and divided into grid cells, then overlayed by an ordering structure such as Z-order curves, which precedes covering it by overlapping Minimum Bounding Rectangles (MBRs). Spatial points that fall exactly in the overlapping areas of the MBRs are known as Boundary Spatial Objects (BSO). Considering those in partitioning methods is specifically expensive.

III. Related Works and Baseline System

A. Related works

The shortcomings of the traditional spatial data partitioning schemes have led to the emergence of custom spatial-aware alternatives. For example, [7, 8] have designed methods that resemble Sort Tile Recursive (STR) [10] partitioning. Their method works by ordering the points horizontally and vertically in reference to their longitude and latitude coordinates. Thereafter, tiles (horizontal and vertical stripes) are employed to split the embedding space into (non)-equally-sized cells. Tiles can be heuristically overviewed as horizontal and vertical lines that cut the embedding space (imagining the earth flattened out) in the designated dimension. The selection of tiles locations (i.e., longitudes and latitudes) may depend on a cost-based model that seek to achieve a balance between the three contradicting spatial data partitioning goals (that are mentioned in section II. B.). Few methods focus on one partitioning aspect and overoptimize it against the other goals. For example, [11] have designed a method that focus on BSO minimization, thus is categorized under the stripe (tile) partitioning family, where a cost model seeks optimized tiles locations. In the same vein, other frameworks simply employ traditional spatial methods. For example, SpatialSpark [12] supports grid-based partitioning that simply flattens the earth out and overlay it with a grid network.

[1] https://github.com/harsha2010/magellan

They also support an STR-alike method. The problem with traditional hierarchical schemes such as the grid-based and quadtree representations is that they do not consider SDL preservation.

B. Usage Model and Baseline System

There are innumerable manners at which big amounts of geo-spatially tagged data need to be clustered to reveal the dynamics of IoT devices mobility. Imagine a scenario where millions of mobility data traces are collected monthly from IoT edge devices such as GPS in taxicabs, shared bikes and handheld devices. In a metropolitan city, there may rise a desire to understand the dynamicity at which people and vehicles are commuting daily. Municipalities' administrations may aim at revealing interesting insights that can help in making strategic decisions. For example, deciding where to put new traffic lights or surveillance cameras. So that they cut the costs by only putting them in locations with a real need. This need applying clustering algorithms and costly proximity-alike spatial queries such as kNN. A well-performing density-based clustering algorithm depends on a spatial partitioning scheme that most importantly considers SDL preservation. This is attributed to the fact that clustering means clumping geometrically-nearby objects into groups (known as clusters). This means that spatial data loads that are apportioned based on their spatial characteristics will end up in same or nearby partitions in the computing cluster, simplifying the process of clustering at run time and significantly cutting off the cost associated with a voluminous network data shuffling. Our baseline system is a custom partitioning method known as SASAP (short for Spatial Aware Self-Adaptive Partitioning) that we have designed in previous works [7, 8]. Also, belongs to the baseline, a spatial query optimizer method that exploited SASAP in implementing DBSCAN-MR [13]. SASAP results in a grid with a sorted order (in both directions, longitudes and latitudes) and applying some median statistics. The process resorts to a recursive halving in one-dimension or quartering in two-dimensions. However, we have found that the computations involved in SASAP induce additional overheads that, at times, are hardly mitigated by the benefits the method provides. SASAP requires sorting spatial data, which is an expensive operation because it is based on sorting parametrized fields (longitudes and latitudes representing spatial points), which then requires applying expensive geometrical operations such as the distance calculation by Haversine formula [14]. Another limitation in SASAP and the associated query optimizer is that they were implemented atop a spatial framework known as GeoSpark [15] that exploits the traditional Spark abstraction (based on the RDD [16] abstraction). Thus, missing an important optimization that is offered by adopting the novel Spark's DataFrame abstraction instead [17], which can further improve the time-based QoS goals (such as lowering the latency) by employing an important optimizer known as Catalyst in Spark terms. To close those voids, in this paper, we alternatively have designed a novel system that overcomes those limitations by being engineered over an optimized SQL-based spatial processing framework atop of Spark (known as Spark's Magellan). Also, we avoid introducing any costly spatial locality preservation method within the layers of the novel partitioning scheme. We discuss SpatialBPE in details in the next subsection.

Algorithm 1 SCAP partitioning scheme

```
     /* points: coordinates in longitude/latitude format,
     neighbourhoods: polygons representing neighbourhoods of
     the embedding space, geoPrec: geohash precision */
1:   tsl = [] //each element → tuples of a neighbourhood
     finalist = []
2:   coveringGeo ← retrieveCoveringGeo (neighbourhoods,
     geoPrec) /* List of geohashes covering each neighbourhood*/
3:   GeoCodedTuples ← geoEncode(tuples)
     /* performing an inner join on geohash by applying the
     'filter' stage in the filter-and-refine spatial join approach */
     /* tsl: list of tuples assigned to neighborhoods. we do this join
     using Spark's Magellan*/
4:   tsl = GeoCodedTuples.join(coveringGeo, GeoCodedTuples
     ("index") == coveringGeo ("index"))
5:   For i = 0 until tsl.size
6:   //currently 'threshold' is calculated by pre-profiling the data
7:       If (tsl[i]. count > threshold)
8:           split_ tsl = split (tsl[i])
9:           finalist.append (split_ tsl)
10       Else
11:          finalist.append (tsl[i])
12:      End
13:  End
14:  For j = 0 until finalist.size
     //materializing data chunks in partitions
15:      partition[j]. populate (finalist[j])
16:  End
```

IV. Efficient Spatial Processing System for Main Memory Frameworks

SpatialBPE constitutes basically two main parts. A spatial custom partitioning method that we term as SCAP (discussed shortly) and an improved query optimizer that exploits SCAP in optimizing a density-based clustering algorithm (known as DBSCAN-MR [13]). We explain those in detail in the next subsections.

A. Spatial Co-Locality-aware partitioner (SCAP)

In this paper, we have designed a spatial aware adaptive big data partitioning scheme for distributed in-memory big data batch processing frameworks. More formally, the workflow of our method is listed in algorithm 1.

The method starts by geocoding the spatial points. We focus in this paper on reducing the dimensionality by using the geohash[2] encoding (which belongs to the family of z-order curves). We then apply an efficient relatively cheap join method readily available through Spark's Magellan for joining the spatial points with a file containing neighbourhoods (imagining the earth flattened out into a two-dimensional planar geometry representation, those are polygons that represent the city administrative divisions put by municipalities).

[2] http://geohash.org/

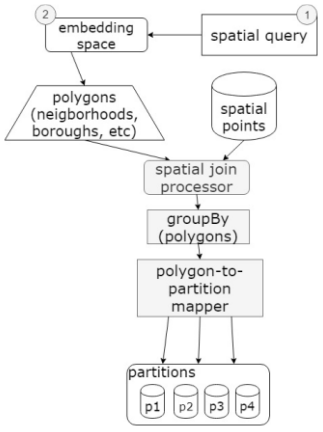

Fig. 1. Spatial co-Locality-aware partitioner (SCAP)

The result of this process is a list of neighbourhoods, where each neighbourhood list contains all the points that geometrically belongs to it. This stage constitutes the strategy we do for clumping geometrically co-located objects into batches that can then be sent to same partitions for processing. However, since geohash is an approximation and since we are applying only the 'filter' stage of the 'filter-and-refine' spatial join approach, then the result of the approximate join is susceptible to generating 'false positives'. Those are the objects that belong to the geohash covering of a neighbourhood, but however in real geometries they do not. We call such objects in this paper as BSOs. In this paper, we focus mainly on preserving spatial data locality as this has shown to reduce data shuffling significantly during a response to a proximity-alike spatial query. In addition, we focus to a lesser extent on load balancing and BSO minimization. We also achieve a good degree of BSO minimization by tuning the tweakable geohash precision. The workflow of SCAP is schematically shown in Fig. 1.

The output of the SCAP algorithm are partitions distributed to the computing resources of the Cloud for parallel processing. Each partition contains roughly fair amount of geometrically-nearby spatial tuples.

To understand the way at which our method replicates BSOs, consider the heuristic overview in Fig. 2. This figure represents the administrative neighbourhood divisions for part of the city of Milan in Italy. All objects (tuples) that belong to g12, g14 and g22 will be replicated in partitions that host tuples from N1 and N2. This is so because, geohash encoding is an approximation and by such representation, tuples with same geohash value may belong to different neighbourhoods. Such points need then to be replicated to bordering cells because at the time we do not know to which neighbourhood (polygon) the point belongs by simply looking at its geohash representation.

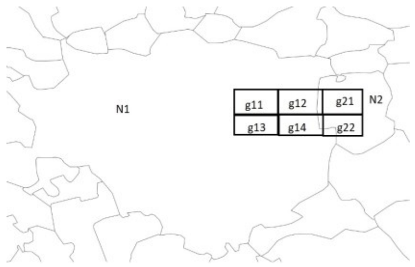

Fig. 2. Part of a heuristic geohash covering for part of the city of Milan in Italy. In the legend, N1 is the first neighbourhood while N2 is the second. g11 is the first geohash covering for N1, while g21 is the first geohash covering for N2 and so on.

B. Spatial Query Optimization in Main Memory Systems

Query optimizers in the current state-of-art systems are not attuned to the shape of the processed spatial data, causing substantial amounts of data to be shuffled around by the underlying optimizers, thus deteriorating the benefits of parallelization. A spatially-attuned query optimizer seeks at selecting efficient query plan that minimizes the shuffling of data over the network.

Density-based clustering algorithms cluster spatial objects by considering high-dense regions as clusters, whereas considering others as noise. Accounting for SDL preservation and BSOs minimization while applying those algorithms in Cloud is challenging.

A well-performing Cloud-based density-based clustering algorithms is DBSCAN-MR [13] , which operates as follows. First, it partitions input data to worker nodes of Cloud deployment. This is followed by applying a local version of the operator instance of the plain DBSCAN for data in each worker node. It then combines local results into unified final clusters. An intrinsic challenge with parallelizing DBSCAN-MR is the need to replicate BSOs to overlapping cells (e.g., cells that result from overlaying the grid representation of the embedding space with approximate Minimum Bounding Rectangles (MBR)), thereafter a post-replication refinement stage is needed to discard extra points.

We have implemented a retrofitted version of DBSCAN-MR over Spark's Magellan which is not natively supported. Our retrofitted version exploits our SCAP partitioning scheme, where we apply SCAP as a frontstage for splitting data to achieve a lower latency during the application of the retrofitted DBSCAN-MR. For our method SCAP, we simply duplicate the BSOs that belong to overlapping geohashes. The size of the BSOs space then depends on two factors. The data skewness and the geohash precision (a multiple of 5). On the contrary, for the SASAP baseline from our previous work [7, 8] , the mathematical model that have been used for replicating BSOs was expensive despite at times leading to less replication. A tension then between some QoS goals is always present. Stated another way, less BSOs means a desirable higher Cloud computing resource utilization, but an undesirable high latency as the replication method used is expensive, while more BSOs but depending on a cheap replication method yields a lower resources utilization in bid for a highly desirable lower latency, which is the benefit we reap by applying SCAP. Since in this paper we are focusing on time-based QoS goals (e.g., lowering the latency), then SCAP is favorable.

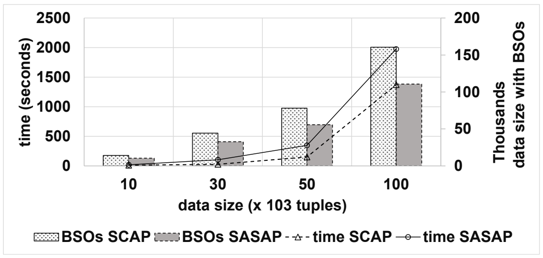

Fig. 4. Running times and number of BSOs of our retrofitted version of DBSCAN-MR over SCAP against SASAP-based using the ParticipAct dataset. Parameters: eps 0.15, minPts 300, geohash 30.

V. Implementation Insights

To show how SCAP and the associated optimizers (forming together our system SpatialBPE) are adept in achieving time-based QoS goals, we have built a standard-compliant prototype on top of Spark [3] . We have basically extended Spark's Magellan, which is a spatial-aware library that is optimized on top of Spark. Spark cores offers an abstract module that allows the creation of custom partitioning methods. We have overloaded that module to implement our SCAP scheme.

A. Deployment Settings and Benchmarking

This section discusses deployment settings that we have selected to evaluate the ability of SCAP and the associated query optimizer.

Dataset. For benchmarking, we use the NY City taxicab trips datasets [3], from which we choose a cohort of around 150k points representing a portion of data captured through taxi rides for the first half of 2016. We have selected the green taxi trips that include fields registering pick-up/drop-off locations. The other dataset is a cohort of 150k spatial points collected during the ParticipAct project , which is a project that has been started at University of Bologna in Italy aiming at achieving the People as a Service (PaaS) vision, where people act as active collectors of data that can be exploited and applied to interesting scenarios such as DBSCAN. Every spatial point has a user locational data (in planar GPS coordinates longitude/latitude) in addition to timestamps pointing to the times of collection.

Deployment and experimental settings. We run our system, SpatialBPE, on a Microsoft Azure HDInsight Cluster hosting Apache Spark version 2.2.1. It consists of 6 NODES (2 Head + 4 Worker) with 24 cores. Head (2 x D12 v2) nodes , and Worker (4 x D13 v2) nodes. Each head node operates on 4 cores with 28 GB RAM and 200 GB Local SSD memory, and quantities are double those figures for worker nodes.

Parameter settings. We have selected to intermix the tweakable parameters in a way that enables us to measure the capabilities of SCAP and its correlated optimizers in achieving a wide choice of qualities. I) Varying the DBSCAN-MR parameters values. We first profile the datasets by applying a *k*NN distance histogram method in order to select the values for epsilon and minPoints, which are the parameters required by DBSCAN-MR, where epsilon is the maximum distance of a point to be considered for a specific cluster. minPoints is the minimum number of points that altogether form a cluster. From the histogram, the best epsilon/minPoints combination are those that fall on the so-called 'knee' of the graph. We have found that there are multiple applicable values, from which we choose the combinations of 0.09 epsilon, 200 minPoints, and the other combination is 0.15 epsilon and 300 minPoints. By this parametrization, we aim at measuring the ability of our methods in achieving low-latency against the baseline method. II) Fixing DBSCAN-MR parameter settings and varying the geohash precision. By this setting, we aim at measuring the ability of our method in striking a balance between the spatial partitioning goals, which then transfers an effect that is reflected on the QoS goals (e.g., latency, resource utilization).

B. Performance Metrics

We use the end-to-end latency for comparing a time-based QoS. Latency follows a low-is-better trend. Latency is the total time required for processing all tuples that arrive during a running session in an end-to-end fashion (i.e., passing through all the operators of a DAG operator graph), from the moment data is ingested by the system until results are served to the user.

We also apply another performance metric that calculates the data size with BSOs. By this metric we aim at measuring the resource utilization QoS, considering that it is a tradeoff which collide with the lowering latency QoS goal.

Furthermore, to quantify the gain of the geohash precision adaptation, we define and apply

$$\text{adapGain} = (((\text{BSO}_{30} - \text{BSO}_{35})/\text{BSO}_{30}) . 100)\,\%$$

Where BSO_{30} is the number of BSOs resulted through the geohash precision 30, whereas BSO_{35} is the number of BSOs resulted through the geohash precision 35.

C. Results and Discussion

We have tested SCAP against SASAP by using the first parameter enumeration. As shown in figures 3 and 4, respectively, our retrofitted version of DBSCAN-MR over SCAP is adept in achieving QoS time-based goals (specifically a lower latency) better than the baseline with which we compare that alternatively exploits SASAP partitioning method from the literature. Notice how an increased number of bordering replicated points (i.e., BSOs) implies a near-linear similar-pattern increase in running times of both implementations. This applies also for the case of value of epsilon that is equal to 0.09

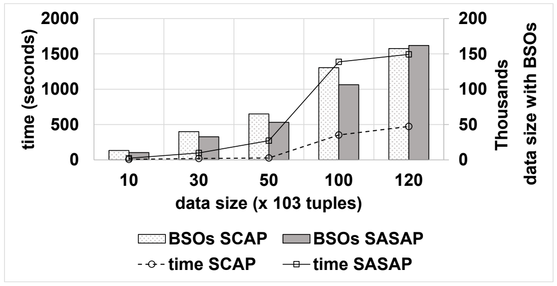

Fig. 5. The effect of tweaking geohash precision on the number of BSOs generated by SCAP on NYC taxicab dataset. Parameters: eps 0.15, minPts 300, geohash 35

of DBSCAN-MR over SCAP against SASAP-based version tested on NYC taxicab datasets. Parameters: eps 0.15, minPts 300, geohash 30.

[3] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

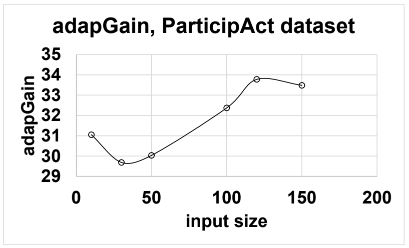

Fig. 6. Adaptation gain we reap by tweaking geohash precision, from 30 to 35 in this case.

and minPoints equals to 200. However, in both cases it negatively affects the running time of our new method SCAP version to a lesser extent as opposed to SASAP baseline counterpart.

Another tunable parameter in our settings is the geohash precision, which then has a utility in determining the number of BSOs that will be marked for replication. Fig. 5 shows that changing the tweakable geohash from 30 to 35 precision yields less BSOs for SCAP. This is because a wider geohash precision implies a smaller size of the cells that this geohash order pass through, which then reduces the overlapping areas between bordering cells, thereby reducing the BSOs count.

Fig. 6 shows that we obtain roughly 32% gain by changing the geohash precision. We sometimes term such gain as the design effect as it is carried over as a result achieved by applying the design of our SCAP scheme.

VI. Conclusions and Future Works

Research efforts in communication service provisioning are currently geared toward designing communication networks that serve as robust infrastructures for the management of huge amounts of data with QoS guarantees. Clouds are built with such design primitives being prioritized. However, at times, the way that parallel data processing frameworks act in such settings may deteriorate the benefits we reap by the elasticity provided by such infrastructures. To assist those efforts, we are tackling the problem from the perspective of optimizing the deployed data processing systems so that they synergistically achieve the goals envisaged by such infrastructures. A special attention in this paper has been given to the pivotal problem of spatial data partitioning in Cloud computing frameworks. Specifically, we focus on in-memory systems that need to process data in fast memory with parsimonious resources. We have designed a locality-preserving spatial partitioning scheme for quality spatial analytics in distributed main memory frameworks. Our method termed SCAP, basically focuses on spatial locality problem to help in minimizing the data shuffling around the network while processing costly proximity-alike queries. In this paper, we specifically focus on a costly density-based clustering algorithm. As a future potential research frontier, we could offload a portion of the data partitioning to IoT devices near the edge. Thus, further assisting novel network designs in achieving high QoS.

Acknowledgment

This research was supported by the IDEHA project funded by PON "RICERCA E INNOVAZIONE" 2014-2020 (no. J46C18000440008).

References

[1] I. M. Aljawarneh, P. Bellavista, C. R. De Rolt and L. Foschini, "Dynamic identification of participatory mobile health communities," in *Cloud Infrastructures, Services, and IoT Systems for Smart Cities*. Springer, 2017, pp. 208-217

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Commun ACM, vol. 51, (1), pp. 107-113, 2008.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster computing with working sets." HotCloud, vol. 10, (10-10), pp. 95, 2010.

[4] 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, (13), pp. 1565-1568, 2016.

[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, (3), pp. 11-18, 2015.

[6] I. M. Al Jawarneh, P. Bellavista, F. Casimiro, A. Corradi and L. Foschini, "Cost-effective strategies for provisioning NoSQL storage services in support for industry 4.0," in 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 1227.

[7] 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," in 2017 IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 851-856.

[8] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, R. Montanari and A. Zanotti, "In-memory spatial-aware framework for processing proximity-alike queries in big spatial data," in 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2018, pp. 1-6.

[9] I. M. Al Jawarneh, P. Bellavista, L. Foschini and R. Montanari, "Spatial-aware approximate big data stream processing," in 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1-6.

[10] S. T. Leutenegger, M. A. Lopez and J. Edgington, "STR: A simple and efficient algorithm for R-tree packing," in Proceedings 13th International Conference on Data Engineering, 1997, pp. 497-506.

[11] H. Vo, A. Aji and F. Wang, "SATO: A spatial data partitioning framework for scalable query processing," in Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2014, pp. 545-548.

[12] S. You, J. Zhang and L. Gruenwald, "Large-scale spatial join query processing in cloud," in 2015 31st IEEE International Conference on Data Engineering Workshops, 2015, pp. 34-41.

[13] B. Dai and I. Lin, "Efficient map/reduce-based dbscan algorithm with optimized data partition," in 2012 IEEE Fifth International Conference on Cloud Computing, 2012, pp. 59-66.

[14] C. C. Robusto, "The cosine-haversine formula," The American Mathematical Monthly, vol. 64, (1), pp. 38-40, 1957.

[15] J. Yu, 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, pp. 70.

[16] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2012, pp. 2.

[17] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin and A. Ghodsi, "Spark sql: Relational data processing in spark," in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 1383-1394.