VOL. 14, NO. 23, DECEMBER 2019 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

EFFICIENT SPATIAL DATA MANAGEMENT BY APACHE SPARK

R. Shiva Shankar, V. Priyadarshini, J. Raghaveni and Ch. Vinod Varma
Department of Computer Science and Engineering, S.R.K.R. Engineering College, Bhimavaram, Andhrapradesh, India
E-Mail: shiva.srkr@gmail.com

ABSTRACT

Nowadays one of the biggest concerns on different organizations around the digital world being generate enormous data at high velocity, which increase extremely over the last few decades. A more constantly witnessed almost 90% of the world data has been drastically produced from last decade. Today's modern world significantly depends on big data tool such as apache spark as the de facto framework are being able to establish several solutions for the process and analyze enormous structured and unstructured data sets, even though still lack about complete support especially for spatial data analysis. Therefore this work focus an extremely high-overall performance on temporal spatial data corresponding programming framework is designated for its extensibility and high performance to mixed together on big data innovations since which is essential to deal mostly with the rapidly growing spatial information. In this paper, we discuss the impact of one of the known solutions is apache spark, one of extremely fast streaming framework at all for massive-large scale data management, then we really describe an advanced strategy to processing massive spatial information. We also explore the spatial data framework integrate with Resilient Distributed Datasets (RDD) geometric data framework which is specially-designed regarding in-memory distributed and parallel computing framework to extremely support various iterative algorithms.

Keywords: big data, apache spark, resilient distributed datasets.

1. INTRODUCTION

In current decade the digital world applications drastically increasing volume and variety of data sets from past years. We focus the rapid improvements of new technologies especially in big data tools both for cost-effective and efficiently processing and storing the enormous data. In general, big data is significantly useful when it is shared between numerous entities. This means several organizations from diverse fields require accessing this data for various functions. A most important and number of distributed oriented big data storing, processing and management systems and tools have been recently proposed and actually developed and made popular in the basic information technology (IT) economy mostly with apache open source hadoop plus spark. Then we really encompass created a software engineering kit called spark spatial-SDK which mostly takes into account the spatial attributes of geospatial information as well as produces a spatial data framework and API facilitated by spark to also allow users to easily and quickly conduct spatial evaluation mostly with large spatial information[1]. Spatial data framework integrate with Resilient Distributed Datasets (RDD) geometric data framework which is specially-designed regarding in-memory distributed and parallel computing framework to extremely support various iterative algorithms. Such an interface is offered, actually called Spatial-RDD, to gain access to large spatial stored data in decentralized databases such as HBase and also to load raw data into spark processing framework. By using some specific examples of data processing mechanisms like geo-spatial target range plus temporal k-nearest-neighbor (KNN) requests, we demonstrate API apps. Typically, the outcome suggests the usefulness of just using spark spatial-SDK for the processing and storage of large geospatial information [1].

1.1 Big data

Big Data is really a descriptive term used to try to collect massive and complex sets of data, which are complicated to storage plus process, just use conventional database administration techniques otherwise hiring processes. Here the objection involves capture, cataloging, store, search, share, transfer, examining and visualization of this information [2]. The 5 attributes which describes Volume, Velocity, Variety, Veracity and Value as illustrate, in Figure-1.

Volume: which specifically focus to the "data volume," that continues to grow really very quickly each day as shown in Figure-2. The overall size of human machine and social networks information itself is still enormous. Scientists have mostly forecasted that sometimes 40 zetta bytes will be produced by 2020 almost an overall increase of even more 300 times actually over 2015.

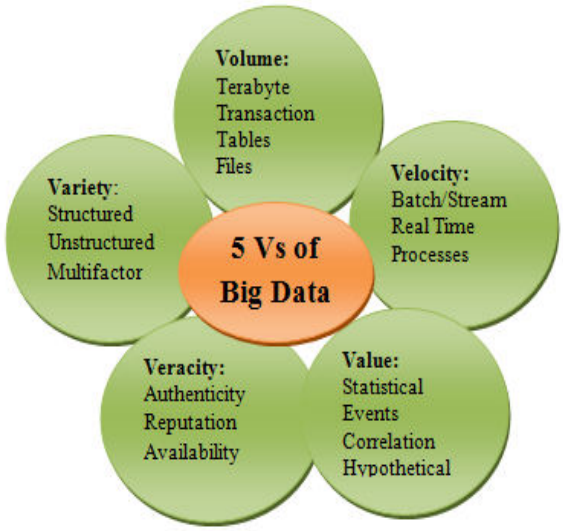

Figure-1. Big Data Characteristics.

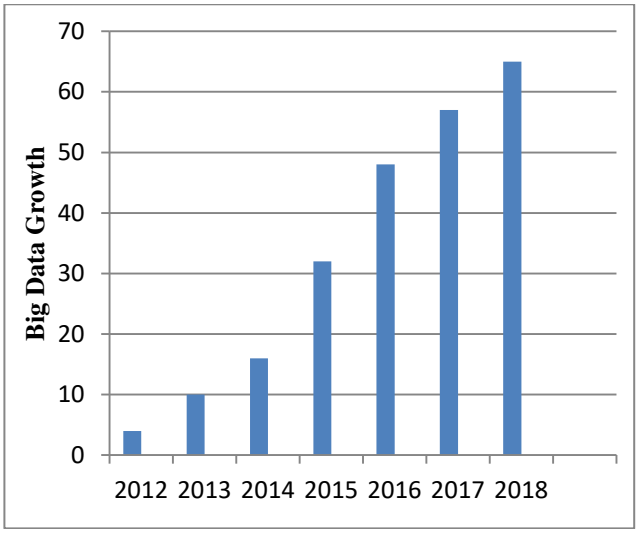

Figure-2. World Data Volume Growth.

Velocity: is the speed at which the data will only be produced by various sources on a daily basis. This flow of information is enormous and rapid. Everyday current (customer Face book DAU) are 1, 03 billion upon mobiles, an improvement of 25% year on year. All of that demonstrates which quickly the amount of customers have been rising upon social-sites and how quick the information is being generated day by day. This can produce unique insights as well as decide mostly on a basis of real time data unless you are able to control the speed [7]. *Variety:* Typically, numerous sources have been contributing to Big Data, the category of information they are produce is diverse. The structure could be structured or unstructured and semi structured. Therefore a huge diversity of information that are generated each and every day. We have previously received information as well from Excel and databases. The data are now displayed as an image, audio, video, sensor, etc. This greater variety of large amounts of data therefore creates issues in data collection, store, mining operations and analysis.

Veracity: This is specifically referring to information collected in uncertainty or ambiguity due to incompleteness on information. Also in the picture below, then you also could see that the table.

Value: Specifically referring to the significance of the extracted information. It really is one major thing to have seemingly endless volumes of data, however it is useless unless it could be did turn into value. However, there is a direct relationship here among information and latest approaches, which may not always actually signify to Big Data has value [2]. The significant important aspect of focusing on a big data strategy is to know the value the also advantages of collecting and storing the data so that the data that is collected can essentially be made available to the public.

1.2 Apache spark stream processing:

Recent decade, apache Hadoop open source Map Reduce was becoming widely popular mostly with big datasets due to its increased efficiency [6]. Map Reduce seems to be supporting only a batch-processing paradigm [5]. However, today is more popular and much successful data processing paradigm called Apache Spark actually addresses all these significant problems. Spark really was introduced to the Apache Software Foundation in 2009 with UC Berkeley, later in 2013. In 2014, Apache Spark was converted into a top-class Apache project. Apache spark is really an advanced technological analytical framework which is capable of processing real world data easily at real-time as represent in Fig.3. It's an efficient and effective as well as much significantly faster in-memory processing and storage paradigm than Map Reduce [5]. In incremental data processing, apache spark seems to be extremely efficient. The memory itself is written with the intermediate and advanced data so that data were already actually present in the memory so that all the data from the disk is not read/ write for each step. which extremely simple and easy to significantly process real live streams of data quickly and accurately[6].

In Figure-4 MLib: This is really a machine library of low level studying which could be named mostly from computer programming languages Scala, Python and Java. Perform various iteration in order to increase precision. The nine times actually as quick also as Apache Mahout's disk-based design and implementation [8]. Only some methodologies are used: k-means. Graph Frame progress even in GraphX, need to provide the API for all three languages uniformly. Apache open source spark is an extremely useful tool to service large data sets computationally. Through it has own Resilient Distributed Dataset (RDD), spark is indeed suitable for this kind of processing. Here the paper actually entitled RDD: high fault-tolerant, parallel data structures which allow customers to specifically maintain intermediate and advanced memory results, complete control even their own partition to maximize data assignment then distort them through various operators. Solution of in-memory is to simply much faster compare to actually reading a massive disk set of data for each processing iteration.

Fastest framework spark is really a cluster based computing mechanism in memory. Here the RDDs are certainly collections of read-only objects could be subdivided the whole computer device into cluster to simply allow data to be processed quickly [9]. An important advantage of such RDDs is that it could be recreated whenever another data partition is completely lost because if the computer knows how it can be extracted from its own common ancestor RDDs. An unchangeable, decentralized huge collection of objects is the Resilient Distributed Dataset. Resilient: in the situation of such massive failure can indeed be preserved. Distributed: activities are indeed exactly parallel as well as distributed throughout the endpoints. Dataset: Actual data fully loaded and subdivided across several cluster nodes(executors) Spark RDDs are still adaptable or fault tolerant, that allows spark to really regain RDD even in the face of shortcomings.

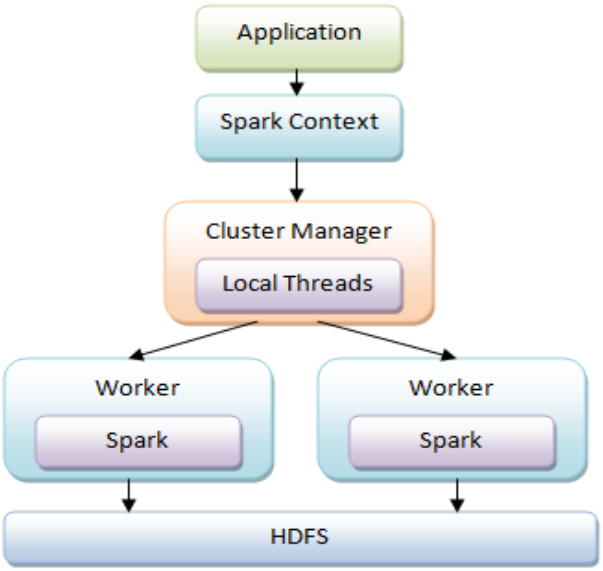

Figure-3. Spark Architecture.

Figure-3 demonstrates Apache Spark's framework. Apache spark is really a driver scheme (Spark Context), employees just called executors, cluster managers, and HDFS. Driver system is the spark's primary program. Spark Context is really the object generated during spark program implementation and is accountable for the complete implementation of the task. The item Spark Context links to the cluster manager,

In fact, the capability to still extremely recomputed RDD. Whenever new machine actually holding to RDD data fails completely, Spark utilizes even this unique ability to recompute the missing obstructions, more open and transparent to every user. Spark's reflection of such of a set of data of which is redistributed across the whole RAM, or working memory, of lots of computers. The "distributed" basic nature of the RDDs operates while an RDD only actually contains a specific reference to the information, while the original data is encompassed only within partitions across the endpoints in it [10]. Mostly by the thought of such an RDD as just a wide variety of integers distributed across many machines. It is indeed simply a cluster-wide divided set of data which could be subdivided mostly from HDFS (Hadoop Distributed File System), HBase table, Cassandra table, Amazon S3.

We've seen a growing quantity of geospatial information mostly from mobile phones, taxis, major types of sensors, social networks, etc in the geospatial realm [3]. Again the information could be used for assessing or even predicting the status metropolitan possessions and providing intelligent decision-making aspects to significantly develop urban development. Therefore, it is important that such a massive amount of spatial information be queried, computed and analyzed by a powerful and effective tool [11]. Though, currently existing extremely high-overall performance computing technologies in spatial data access could not be linked openly and handing simply because of spatial attributes of geospatial datasets. In order to promote spatial data, some configurations are often focused on improving all these mechanisms.

This paper actually wants to introduce to advanced strategy to processing huge spatial information mostly with apache spark as well as provides an excellent an simple-to-use software development tool kit on the spatial data analysis tool called spark spatial-SDK. Then we really design and build and spatial-RDD only in spark spatial-SDK based mostly on the characteristics of both the central part of spark framework and RDDs, which mostly enhances fairly normal RDDs, could be more consistent only with objects of spatial sources. This paper's primary contributions are as follows: (1) Lengthening RDD to Spatial based RDD (Polygon, Point, Poly-line) as just an expanded data framework for easily loading, indexing, managing, and processing large spatial information stored in distributed databases including HBase, (2) Providing spatial evaluation providers in Spatial-RDD, such as range and k-nearest neighborhood services. We are therefore setting up a Spark-Spatial-SDK to make it possible at all for users to really process large spatial data.

The remaining paper could be arranged here as actually follows: Section II incorporates important innovations for the processing and storage of spatial information to improve here the distributed processing system. Section III introduces the advanced tools for using apache spark to really process people large spatial and temporal data. The demonstration plus experimentation presenting in section IV. Lastly the absolute conclusion plus upcoming projects provided in Vth section.

2. RELATED WORK

These world data is actually generated mostly at high speed mostly from numerous sources and which is received from a wide category of sources. Because when actual size of world data hits exabyte’s or even more, it couldn’t be handled efficiently by current tools and methods so probably and significantly requires efficient and reliable storage capacity and recovery management services, such type of data is called the big data.

Extremely technology enhancement on geo-spatial data processing as well as statistical examination is already being leveraged. For case, Hadoop based GIS and Spatial based Hadoop actually attach spatial enhancements to open source Apache Hadoop to reap the benefits from whole Map-Reduce paradigm, encouraging customers to actually make spatial analyzes in distributed manner and fault-tolerant approach. Mostly with the significant improvement on currently existing memory and their demands in today's modern machines, by simply applying memory-based decentralized optimization techniques, Apache spark looks more impressive overall performance as well as fault tolerance than one of Hadoop. Exploring large-scale geospatial data analysis and processing especially in spark is therefore beneficial for achieve improved performance. The massive geospatial data management system that is based mostly on apache spark is really being designed and built with some beginning work. Though for example, GeoSpark tends to create a series of SRDDs mostly with apache spark's RDDs to really start reading temporal data both on text based files including Comma-Separated Values-csv, Geo JSON, as well as Well-known Text- WKT. So then, information will only be divided up in the small cluster by a worldwide grid system and otherwise indexed for each partition table mostly with local R-Trees or Quad-Trees to enhance the performance of spatial query. In order to optimize overall performance, Location Spark expands RDD to really Spatial-RDD as well as creates various global spatial datasets. But only about point information is strictly limited. This also examines query bias, excessively high and maybe even un-optimized communications network, and bandwidth-reducing I / O information exchange. Also in Hadoop Distributed File System (HDFS), Spark-Spatial extremely storing temporal information even in the specific form of WKT and processes everything with generic RDDs [12].

Moreover, even in the text message file system, that the above-described mechanisms successfully manage of their own input and output data sets. By using a decentralized database such as HBase to really store and connect directly a massive temporal information for Spark processing and storage, there seems to be no examination. HBase seems to be more reliable, accurate, efficient and therefore more extensible. The major job at all in this paper provides complete routing and transitions mostly from HBase tables that store large temporal information to Spatial-RDD to enable a complete solution for large geospatial information processing and storage.

3. PROPOSED METHODOLOGY

Spatial data with apache spark exhibits the modern architecture of within our own strategy to apache spark processing large temporal information in Figure-5 and Figure-6. This usually include three kinds of layers: (1) Spatial oriented Information Layer, consisting of a vast number of tables which store large temporal information while in an Apache open source HBase small clustering; (2) Spatial across RDD based Layer, consisting like sequence of RDDs that manage as well as compute various spatial objects, including point , line, and polygon based; and (3) Spatial oriented processing and storage interfacing layer, providing a new sort of incredibly user-friendly interfaces both in favor of large objects to every customers. Also in regards, the spatial analysis organization as well as deployment depends on spatial RDD based layer and spatial process interfacing layer is entirely depends on Geo Star's GIS based kernel.

To store temporal information with all of HBase, we need to use a similar strategy [13]. Each and every row does have a row based master key, a time plus date and new columns are entire trying to represent a temporal object on spatial information as long tables in hadoop open source HBase.

Right key on row seems to be the only ID component even in a row because we utilize the geohash significance of the center point of the space object as the row right key for indexing each and every row. Column extended family is a new column collection; it already includes a WKT row to storage space object communication and attributes data rows [14].

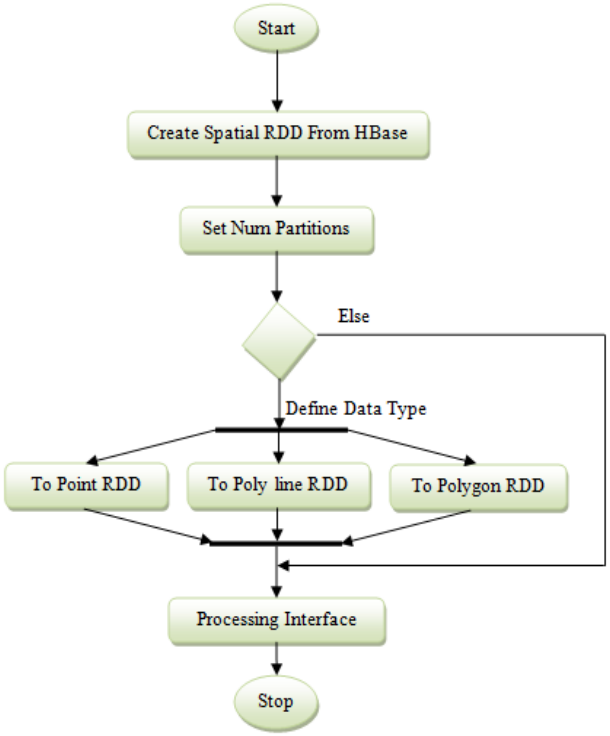

Figure-4. Processing Flow of Spark Spatial SDK.

A. Maps Spatial Information mostly from HBase Table to just Spatial-RDD

First of all, also in the Creating Spatial RDD Of HBase functionality, then we really accept apache Hadoop RDD provided by Apache Spark's absolute core for reading almost all divided row information in HBase Table [13]. Then afterwards, a mapping facility could completely transform apache Hadoop RDD into a Mapping Partition RDD synthesizes row keys as well as space objects mostly from row information within each information partition, and the generated Mapping

Partitioning RDD is really a group of divided records (geo hash real value, space object). Eventually, that is possible to design a Spatial-RDD by simply getting the Mapping Partitioning RDD mostly as a variable of the data set. The Spatial-RDD also can be converted into the pointed and whose fundamental basis is (geo-hash actual value point actually object), the Polyline RDD whose core component was, and the Polygon-RDD whose central component is (geohash value, polygon object) to follow different kinds of temporal information. Such classic novel RDDs were essential components of Apache Spark's strategy to processing large spatial information.

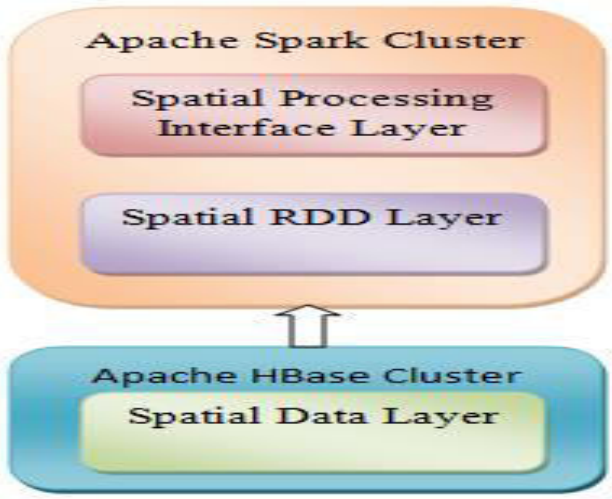

Figure-5. Block Diagram.

B. Interface aspects used to Processing Big Spatial Data

Then we really create a particular set of decentralized geo-databases user interfaces for all of these RDDs as well as coordinate them further into the Spark-Spatial-SDK to promote querying as well as computing large temporal information only with Spatial-RDD and many other RDDs only in a distributed network. A latest Apache Spark-Spatial-SDK outline is shown in Figure-4, and maybe even extra functionalities are now under remodelling [14].

To Point RDD functionality: though if objects even in Spatial-RDD reside to a point point actually class, each common temporal item will just be transformed into a relevant point entity as well as a Point-RDD returned back.

Interface to Polyline-RDD: though if attributes in Spatial-RDD actually belong to a polyline group, each fairly normal temporal item will only be transformed into another relevant point entity and a Polyline-RDD returned.

Interface to Polygon-RDD: because if attributes in Spatial-RDD actually belong to a polygon group, each completely normal temporal item will just be transformed into another relevant point entity and maybe a Polygon-RDD returned.

Set Numpartitions interface: Whenever customers setting the amount of new dividers, again the partitioner could repartition massive temporal information in the workforce small cluster obviously, it depends on the spatial equity index and eventually return the reformatted Spatial-RDD.

Interface save As HBase Table: this little functionality can mapping spatial information mostly from Spatial RDD once again to HBase table in order to indefinitely want to save the outcome data.

Interface range Query: it actually provides temporal information in Spatial-RDD mostly with decentralized spatial effective range list of questions

Cluster: By utilizing a spatial junction techniques offered on GeoStar kernel, a temporal distance query window would be sent to each information partition as well as filter temporal objects that do not overlap with it anyway. Eventually, again the remaining objects will just be extracted as both the outcomes of the list of questions in each partition.

KNN Query interface: whenever customers describe the center main point and hence the total number K to actually call the user interface, the center point's k-nearest neighbors will only be found in the cluster's spatial objects. It's first, the center main point as well as actual quantity K could be transfer from the mastering node to every divider in the working nodes also to sort queue can be manipulated by compare mathematical distance within each partition to catch more knn of the core point. Then after, just as the true value (length, (geo-hash real value, temporal entity) normally takes the minimum distance to either the start point, almost all information will be shuttled mostly on the cluster and requested mostly from the simplest to the highest. Maybe at least here as the outcomes, top in terms-K information records will only be collected.

C. Data Flow on Spatial Processing

The processing flow exposes about spatial data about the way of process massive temporal information only with apache spark by using spark-spatial-SDK in Fig. 5. First, when after establishing the apache spark and HBase setting variables, customers could use the Create Spatial-RDD from HBase functionality to gain access massive spatial information stored in HBase tables just to get a Spatial-RDD [1]. Second, mostly using set num partitions interface, utilizes can really repartition cluster information to achieve better overall performance. Finally in order to calculate, store and make the query about large spatial information over the distributed cluster, a sequence of spatial information proceeding strong interface could make available for the above reveal RDDs [13].

4. RESULT ANALYSIS

In the present tests, we concentrate on a micro benchmark analyzing performance rates with distinct environments for separate operators. An Intel Core i8 processor has performed the benchmarks and 8 GB RAM for each node on our 16-node cluster. The nodes have been linked to a network of one Gbit / s. The Spark tasks have been performed to each executor of 32 executors and 2 cores. We would first have to solve problems in GeoSpark3 to operate our experimental work. The first

and most significant issue was because the applicant set received by the index (R-tree) performed activities using an index for querying. To get the right outcomes, we incorporated the applicant-pruning phase.

The very first test performs a spatial screen controller over an information set of 60, 000, 000 polygons (900 GB, uniform distribution) with such a predicate containing certain polygons containing a specific query point. They only used required spatial practitioners to each paradigm and, if necessary, performed the activity without indexing and with the development of live (on - the-fly) indexes. Throughout this test, hardly 48 (BSP, active index) or 53 secs (BSP, no index) conducted best by STARK. The usability of Spatial Spark is restricted, as spatial partitioning in conjunction with a screen is still not permitted. It required 3860 secs (more than one hour) to operate without geographic partitioning and indexing.

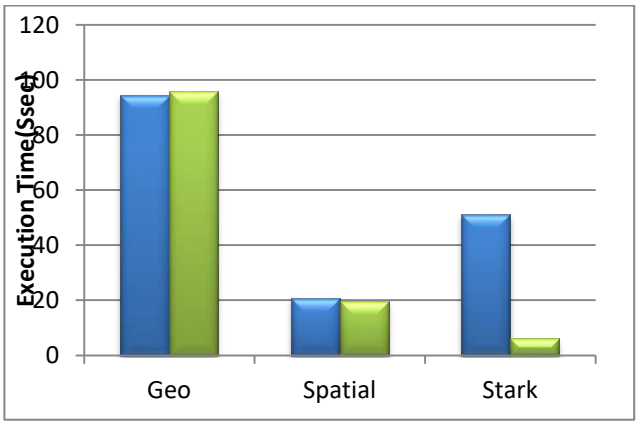

Figure-6. Execution Times for queries.

Geo Spark took 1236 secs (20 mins), yet couldn't handle such a information set with such a space partitioner as well as collapsed on each partitioner after many hrs. In researching the above issue, we carried out the program with 1, 000, 000 requests (17, 6 GB) using a narrower polygon collection of information. Throughout the best scenario, Hilbert's partitioning did take 53 secs. STARK required 900 GB to handle it simultaneously. We have been using their command line scheme for Spatial Hadoop (as the delegate of Hadoop-based schemes), but could not obtain a right outcome: Since 38 mins, the program ended with null outcomes.

The issue is that there is no spot query choice accessible, but as a query scope we supplied a point. In our GitHub database, you can find a visual representation of the processing intervals with a much more thorough evaluation that involves the partitioning costs.

Our next experiment looks at the implementation duration impact of the query distance length. We using a point data of 50,000,000 points for this purpose and performed a filter operator with such a contained by referent to discover all points found in a given polygon. Whereas the information set does have a value variety of [-180, 180] forx and [-90, 90] for y, we perform the filter with 5 distinct polygon tiles.

Its border ranges of such squares are 1, 5, 10, 50, and 100. There is also a request space, which includes the entire information storage. Figures 1 to 3 indicate all partitioner / indexing components implementation times. Practitioners requiring the sum of dividers to be placed beforehand both use the same quantity (8100). This demonstrates the effect the paradigms could have on the pruning phase. Because the area of request is tiny, just one or few partitions could involve items of consequence as well as other disks need not be inspected. Wherever feasible, STARK uses the whole partition trimming strategy and is therefore able to best others who do not appear to conduct the whole activity because they want the same speed across all six requests. We evaluated the spatial join procedure on two top sets of data (1,000,000 points, uniform distribution) over the last test we demonstrate here, which discovers maximum points (same precise place) in the two data sets. Figure-4 demonstrates the outcome of joining even without to use an index for the Spark-based paradigms with their finest partitioner. With Spatial Hadoop, we have been unable to conduct such an experiment as when the command line scheme collapsed with such an issue and thus no beneficial paperwork was found. The very same partitioner proven to be successful across both instances for Geo Spark and Spatial Spark. Moreover, as in the ultimate consequence 1 and ca, Geo Spark does have a defect of Grid and R-tree partitioning. 10,000 data types have been lost (of each iteration of the test we still found distinct outcome dimensions). It could also be used to see that you could not profit from active indexing for such paradigms. This could be because the velocity of querying the index of the stage is not large enough yet to accommodate for the moment needed to construct the index. The Grid partitioner done much better to STARK instead of using an index, but it was lighter than Spatial Spark. Moreover, the BSP has been strongest to live indexation and exceeded the others. The motivation for this may be that the BSP produced partitions with such an equal amount of components on the executors and therefore equal workload.

5. CONCLUSIONS

With the latest big data's revolutionary development, different data processing frameworks have been developed. A few of these data processing frameworks are excellent for streaming data, whereas others could be utilizing to batch processing.

This paper presents in order to offer a new strategy to loading, maintaining, computation and trying to access large temporal information mostly on the apache spark and HBase-based purely decentralized computer cluster mechanism. Typically utilize the RDD characteristics, central part like apache sparks, then really programmed Spatial-RDD as well as other highly specific RDDs (e.g. Poin-RDD, Poly-line-RDD and Polygon-RDD) also with a complete set of user-friendly workflows for processing temporal information stored only in decentralized databases including HBase. Throughout this paper, we implemented motors depending on Hadoop and

Spark that enable Big Spatial Data to be processed. As our function comparative and micro benchmark outcomes demonstrate, they all vary in both assisted activities and execution and efficiency. Moreover, only a reference point for a much more comprehensive spatial benchmark must be the executed micro benchmark. To generate clustered information of distinct dimensions readily, a much more dynamic data generator is required. In addition, as seen in the past section, a healthy benchmark should provide micro benchmark experiments and also a macro benchmark that performs real-world questions.

REFERENCES

[1] Aji. Ablimit, Vo. Hoang, Liu. Qiaoling, Wang. Fusheng, Saltz. Joel, Lee. Rubao. 2013. Hadoop-GIS: A high Performance Spatial Data Warehousing System over MapReduce. Proceedings of the VLDB Endowment. 6: 1009-1020.

[2] Lu. Jiamin, Güting. Ralf Hartmut. 2012. Parallel Secondo: Boosting Database Engines with Hadoop. IEEE 18th International Conference on Parallel and Distributed Systems. pp. 738-743.

[3] Shoji. Nishimura, Das. Sudipto, Agrawal. Divyakant, Abbadi. Amr El. 2011. MD-Hbase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. IEEE 12th International Conference on Mobile Data Management, pp.7-16.

[4] R. R. Vatsavai, A. Ganguly, V. Chandola, A. Stefanidis, S. Klasky, S. Shekhar. 2012. Spatiotemporal data mining in the era of big spatial data: Algorithms and applications. in: Proceedings of 2nd ACM Sigspatial International workshop on analytics for big geospatial data, redondo beach, California. pp. 1-10.

[5] Y. Bu, B. Howe, M. Balazinska, M.D. Ernst. 2010. Haloop: efficient iterative data processing on large clusters. Proc. VLDB Endow. 3: 285-296.

[6] J. Ekanayake, B. Zhang, T. Gunarathne, S.H. Bae, J. Qiu, G. Fox. 2010. Twister: A runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM, USA. pp. 810-818.

[7] A. Fernndez, S. Del Ro, V. Lpez, A. Bawakid, M.J Del Jesus, J.M. Bentez, F. Herrera. 2014. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. In Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 4: 380-409.

[8] Assefi. Mehdi, Behravesh. Ehsun, Liu.Guangchi, Tafti. Ahmad P. 2017. Big Data Machine Learning using Apache Spark MLlib. In IEEE International Conference on Big Data (BIGDATA). 3492-3498.

[9] A. Abbasi, S. Sarker, R. Chiang. 2016. Big data research in information systems: Toward an inclusive research agenda. Journal of the Association for Information Systems. 17: 1-12.

[10] J. Archena, E. M. Anita. 2016. Interactive big data management in healthcare using spark. in Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC 16). Springer. pp. 265-272.

[11] R. Pita, C. Pinto, P. Melo, M. Silva, M. Barreto, D. Rasella. 2015. A spark-based workflow for probabilistic record linkage of healthcare data. in EDBT/ICDT Workshops. pp. 17-26.

[12] M. S. Wiewi´orka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, M. J. Okoniewski. 2014. Sparkseq: fast, scalable, cloudready tool for the interactive genomic data analysis with nucleotide precision. Bioinformatics. pp. 343.

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica. 2010. Spark: Cluster computing with working sets. Hot Cloud. 10: 89-95.

[14] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin. 2016. Apache spark: a unified engine for big data processing. Communications of the ACM. 59: 56-65.

[15] M. Armbrust, T. Das, A. Davidson, A. Ghodsi, A. Or, J. Rosen, I. Stoica, P. Wendell, R. Xin, M. Zaharia. 2015. Scaling spark in the real world: performance and usability. Proceedings of the VLDB Endowment. 8: 1840-1843.

[16] C. Y. Lin, C. H. Tsai, C. P. Lee, C. J. Lin. 2014. Large-scale logistic regression and linear support vector machines using spark. in Big Data (Big Data), 2014 IEEE International Conference on. IEEE. pp. 519-528.

[17] A. G. Shoro, T. R. Soomro. 2015. Big data analysis: Apache spark perspective. Global Journal of Computer Science and Technology. Vol. 15.