Dart: A Geographic Information System on Hadoop

Hong Zhang[*], Zhibo Sun[*], Zixia Liu[*], Chen Xu[†] and Liqiang Wang[*]
[*]Department of Computer Science, University of Wyoming, USA
[†]Department of Geography, University of Wyoming, USA
Email: {hzhang12,zsun1,zliu5,cxu3,lwang7}@uwyo.edu

Abstract—**In the field of big data research, analytics on spatio-temporal data from social media is one of the fastest growing areas and poses a major challenge on research and application. An efficient and flexible computing and storage platform is needed for users to analyze spatio-temporal patterns in huge amount of social media data. This paper introduces a scalable and distributed geographic information system, called Dart, based on Hadoop and HBase. Dart provides a hybrid table schema to store spatial data in HBase so that the Reduce process can be omitted for operations like calculating the mean center and the median center. It employs reasonable pre-splitting and hash techniques to avoid data imbalance and hot region problems. It also supports massive spatial data analysis like K-Nearest Neighbors (KNN) and Geometric Median Distribution. In our experiments, we evaluate the performance of Dart by processing 160 GB Twitter data on an Amazon EC2 cluster. The experimental results show that Dart is very scalable and efficient.**

Index Terms—**Social Network; GIS; Hadoop; Hbase; Mean Center; Median Center; KNN**

I. Introduction

Social media increasingly becomes popular, as they build social relations among people, which enable people to exchange ideas and share activities. Twitter is one of the most popular social media, which has more than 500 million users by December 2014 and generates hundreds of GB data per day[1]. As tweets capture snapshots of Twitter users' social networking activities, their analysis potentially provides a lens for understanding human society. The magnitude of data collection at such a broad scale with so many minute details is unprecedented. Hence, the processing and analyzing of such a large amount of data brings the big challenge. As most social media data contain either an explicit (e.g., GPS coordinates) or an implicit (e.g., place names) location component, their analysis can benefit from leveraging the spatial analysis functions of geographic information systems (GIS). However, processing and analyzing big spatial data poses a major challenge for traditional GIS[**?**]. As the size of dataset grows exponentially beyond the capacity of standalone computers, on which traditional GIS are based, there are urgencies as well as opportunities for reshaping GIS to fit the emerging new computing models, such as cloud computing and nontraditional database systems. This study presents a systematic design for improving spatial analysis performance in dealing with the huge amount of point-based Twitter data.

Apache Hadoop[2][3] is a popular open-source implementation of the MapReduce programming model. Hadoop hides the complex details of parallelization, fault tolerance, data distribution, and load balancing from users. It has two main components: MapReduce and Hadoop Distributed File System (HDFS). MapReduce paradigm is composed of a map function that performs filtering and sorting of input data and a reduce function that performs a summary operation. HDFS is a distributed, scalable, and portable file system written in Java for the Hadoop framework, which provides high availability by replicating data blocks on multiple nodes.

Apache HBase[4] is an open-source, distributed, column-oriented database on top of HDFS, providing BigTable-like capabilities. It provides a fault-tolerant way and ability of quick accessing to large scale sparse data. Tables in HBase can serve as the input and output for MapReduce jobs, and be accessed through Java API. An HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database, but each row must have a primary key, which is used to access HBase tables.

In big data computing, Hadoop-based systems have advantages in processing social media data[5]. In this study, we use two geographic measures, the mean center and the median center, to summarize the spatial distribution patterns of points, which are popular measurements in geography[6]. The method has been used in a previous study to provide an illustration of social media users' awareness about geographic places[7]. The mean center is calculated by averaging the x- and y-coordinates of all points and indicates a social media user's daily activity space. However, it is sensitive to outliers, which represent a user's occasional travels to distant places. The median center provides a more robust indicator of a user's daily activity space by calculating a point from which the overall distance to all involved points is minimized. Therefore, the median center calculation is far more computing intensive. One social media user's activity space comprises geographic areas in which he/she carries out daily activities such as working or living. The median center thus shows a gravity center of that person's daily life.

We design a spatial analyzing system, called Dart, on top of Hadoop and HBase in purpose of solving spatial tasks like K-nearest neighbors (KNN) and geometric median distribution for social media analytics. Its major advantages lie in: (1) Dart provides a computing and storage platform that is optimized for storing social media data like Twitter data. It employs a hybrid table design in HBase that stores geographic information into a flat-wide table and text data into a tall-narrow table, respectively. Thus, Dart can get rid of the unnecessary reduce stage for some spatial operations like calculating mean and median centers. Such a design not only cuts down users' development expenditures, but also significantly improves computing performance. In addition, Dart avoids load imbalance and hot region problems by using pre-splitting technique and uniform hashes for row keys. (2) Dart can conduct complex spatial operations like the mean center and median center calculations very efficiently. Its methodology layer is a completely flexible and totally extensible module, which provides a better support to the upper analysis layer. (3) Dart provides a platform to help users analyze spatial data efficiently and effectively. Advanced users also can develop their own analysis methods for information exploration.

We evaluate the performance of Dart on Amazon EC2[8] with a cluster of 10 m3.large nodes. We demonstrate that our grid algorithm for the calculation of median center is significantly faster than the algorithm implemented by traditional GIS, and we can gain an improvement of 7 times on a 160 GB Twitter dataset and 9 to 11 times on a synthetic dataset. For instance, it costs 1 minute to compute the mean or the median center for 1 million users.

The rest of this paper is organized as follows. Section II discusses the architecture of Dart, and describes its optimizations based on Hadoop and HBase. We then describe algorithms for calculating the geographic mean, midpoint, and median center in Section III. Section IV details two data analysis methods: KNN and geometric median distribution. Section V shows the experiment results. Section VI provides a review of related works. Conclusions and future work are summarized in Section VII.

II. System Architecture

Figure 1 shows an outline of our spatial analyzing system Dart for social network. Our system can automatically harvest Twitter data and upload them into HBase. It decomposes data into two components: the geographic information, and the text information, then insert them into tall-narrow and flat-wide tables, respectively. Our system targets two types of users: GIS engineers and GIS users. GIS engineers can make use of our system to develop new methods and functions, and design additional spatial modules to provide more complex data analysis; GIS users can build complicated data analysis models based on current spatial data and methods for tasks such as analyzing the geographic mean and median centers. All input and output data are stored in HBase to provide easy and efficient search. It also supports a spatio-temporal search for fine-grained data analysis.

Dart consists of four layers: computing layer, storage layer, methodology layer, and data analysis layer. The computing layer offers a MapReduce computing model based on Hadoop. The storage layers employs a NoSQL database, HBase, to store spatio-temporal data. We design a hybrid table schema for data storage, and use pre-splitting and uniform hashes to avoid data imbalance and hot region problems. Our implementation on Hadoop and HBase has been optimized to process spatial data from social media like Twitter. The methodology layer is to carry out complex spatial operations such as figuring out the geographic median or facilitate some statistical treatments

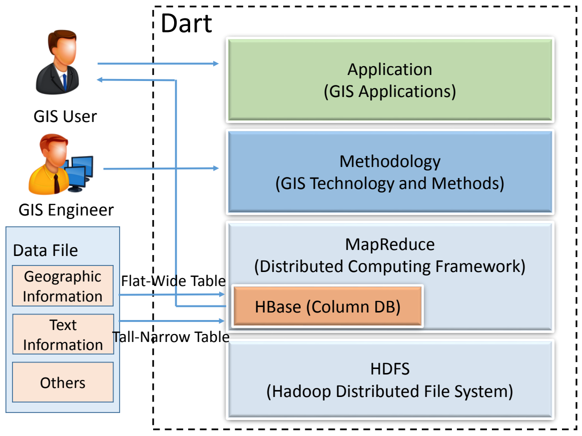
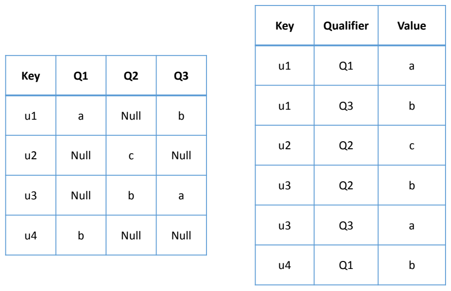


Fig. 1. System architecture of Dart.



Key	Q1	Q2	Q3
u1	a	Null	b
u2	Null	c	Null
u3	Null	b	a
u4	b	Null	Null

Key	Qualifier	Value
u1	Q1	a
u1	Q3	b
u2	Q2	c
u3	Q2	b
u3	Q3	a
u4	Q1	b

Fig. 2. Representations of Horizontal table and Vertical table.

to support the upper data analysis layer. In the data analysis layer, Dart supports specific analytics like KNN and geometric median distribution from customers.

A. *Horizontal v.s. Vertical*

Efficient management of social media data is important in designing data schema. There are two choices when using NoSQL database: tall-narrow, or flat-wide[9]. The tall-narrow paradigm is to design a table with few columns but many rows, while the flat-wide paradigm is to store data in a table with many columns but few rows. Figure 2 shows a tall-narrow table and its corresponding realization in the flat-wide format.

For a flat-wide schema, it is easy to extract a single user's entire information in a single row, which can easily fit into a MapReduce program. For a tall-narrow table, each row contains a single record of a user's entire information to avoid data imbalance layout and too much data stored in just a single row.

For sparse data, the tall-narrow table is a common way and can support e-commerce type of applications very well[10]. In addition, HBase only splits at row boundaries, which also contributes to the users' choice of tall-narrow tables[9]. Using a flat-wide table, if a single row outgrows the maximum of a region size, HBase cannot split it automatically, which makes data stored in that row overloaded.

In our system, we employ a hybrid schema, in which we store geographic data by a flat-wide table, and store other data like text data by a tall-narrow table. Since numerical location information is significantly smaller than text data, each user's location data can easily fit into a region size (256M by default). Our design can make complex geographic operations more efficient due to removing the reduce stage from a MapReduce job.

B. *Pre-splitting*

Usually HBase handles the splitting of regions automatically. That means if a region reaches the maximum size, it will be splitted into two halves so that each new region can handle its own data. This default behavior is sufficient for most applications, but we need to execute insert operation frequently. For example, in social media data, it is unavoidable to have some hot regions, especially when the distribution of data is skewed. In our system, we firstly pre-split the table into 12 regions (the size depends on the scale of cluster) by $HexStringSplit$. The format of a HexStringSplit region boundary is the ASCII representation of a MD5 checksum. Then we employ a hash function on the row key, extract the first eight characters as a prefix to the original row key. In this way, we could make data distribution uniform and avoid the data imbalance problem.

In order to make spatial analysis more efficient on Dart, we also investigate optimizations of system parameters. As shown in [**?**], the number of maps is usually determined by the block size in HDFS, and it also has a significant effect on the performance of a MapReduce job. Smaller block size usually makes system launch more maps, which costs more time and resources. If a Hadoop cluster does not have enough resources, some of the maps have to wait until resources are released, which may degrade the performance further. In contrast, a large block size could decrease the number of maps and parallelization. In addition, each map needs to process more data and could be overloaded. Moreover, failure is an unavoidable situation. If the block size is large, we need longer time to recover it, especially for straggler situation. Furthermore, too big block size may cause data layout imbalance. According to the performance measurement in our experiments, we decide to set the block size to 256 MB instead of the default one (64MB) because the performance with larger size (than 256 MB) does not increase obviously. The region size of HBase is also 256 MB, which makes accessing to the data faster. In the map stage, the sort and spill phase costs more time than other phases. So it is necessary to configure a proper setting in this phase. Since the sort and spill phase happens in the container memory, according to our job features, setting a larger minimum JVM heap size and sort buffer size could affect performance remarkably. Since we need to analyze hundreds of GB data and our EC2 instances has 7.5 GB memory, we set our JVM heap size to 1GB instead of 200MB through "mapred.child.java.opts" and set the buffer size to 512 MB through "mapreduce.task.io.sort.mb", thus we could allocate more resources for this phase.

III. METHODOLOGY

In this section, we describe how to calculate the geographic mean, midpoint, and median. We present a brand-new algorithm for the geometric median calculation that (1) starts the iteration with a more precise initial point and (2) imposes a grid framework to the process to reduces the total iteration steps. These three geographic indicators are important estimators for summarizing location distribution patterns in GIS. For example, it could help us estimate a person's activity space more accurately.

$$\begin{aligned} Lat &= \sum_{i=1}^{n} lat_i/n \\ Lon &= \sum_{i=1}^{n} lon_i/n \end{aligned} \tag{1}$$

A. *Geographic mean*

The main idea of the geographic mean is to calculate an average latitude and longitude point for all locations. The projection effect in mean center calculation has been ignored in this study because the areas of daily activity space of Twitter users are normally small. Equation 1 shows basic calculation steps. The main problem here is how to handle points near or on both sides of the International Date Line. When the distance between two locations is less than 250 miles (400 km), mean is approximate to the true midpoint[11].

B. *Geographic midpoint*

The geographic midpoint (also known as the geographic center, or center of gravity) is the average coordinate for a set of points on a spherical earth. If a number of points are marked on a world globe, the geographic midpoint is at the geographic center among these points.

Initially, latitude and longitude of each location are converted into three dimensional cartesian coordinates after changing unit to radians. We then compute the weighted arithmetic mean of cartesian coordinates of all locations (use 1 as weight by default). After that, the three dimensional average coordinate is changed back to latitude and longitude in degrees.

C. *Geographic Median*

To calculate the geographic median of a set of points, we need to find a point that minimizes the total distance to all other points. A typical problem here is the Optimal Meeting Point (OMP) problem that has considerably practical significance. The implementation of this algorithm is more complex than the geographic mean and the geographic midpoint since the optimal point is approached iteratively.

Algorithm 1 The original algorithm for median

```
Input: Location set $S = \{(lat_1, lon_1), ......, (lat_n, lon_n)\}$
Output: Coordinates of the geographic median
 1: Let $CurrentPoint$ be the geographic midpoint
 2: $MinimumDistance =$
        $totalDistances(CurrentPoint, S)$
 3: for $i = 1$ to $n$ do
 4:     $distance = totalDistances(location_i, S)$
 5:     if ($distance < MinimumDistance$) then
 6:         $CurrentPoint = location_i$
 7:         $MinimumDistance = distance$
 8:     end if
 9: end for
10: Let TestStep be diagonal length of the district
11: while ($TestStep < 2 \times 10^{-8}$) do
12:     $updateCurrentPoint(CurrentPoint, TestStep)$
13: end while
14: return CurrentPoint
```

Algorithm 2 The improved algorithm for greographic median

```
Input: Location set $S = \{(lat_1, lon_1), ......, (lat_n, lon_n)\}$
Output: Coordinates of the geographic median
 1: Let $CurrentPoint$ be the geographic midpoint;
 2: $MinimumDistance =$
        $totalDistances(CurrentPoint, S)$
 3: Divide the district into grids;
 4: Calculate the center coordinates for each grid and count
    the number of locations distributed in each grid as weight;
 5: Calculate the total weighted distance between center of
    each grid to centers of other grids;
 6: If any center has a smaller distance than CurrentPoint's,
    replace it with this center, and update;
 7: Let TestStep be diagonal length of grid divided by 2;
 8: while ($TestStep < 2 \times 10^{-8}$) do
 9:     $updateCurrentPoint(CurrentPoint, TestStep)$
10: end while
11: return CurrentPoint
```

1) The Original Method: Figure 1 shows the general steps of the original algorithm. Let $CurrentPoint$ be the geographic midpoint computed above as the initial point, and let $MinimumDistance$ be the sum of all distances from $CurrentPoint$ to all other points. In order to find a relatively precise initial iteration point, we count the total distance from each place to other places; if any of these places has a smaller distance than $CurrentPoint$, replace $CurrentPoint$ by it and update $MinimumDistance$. Let $TestStep$ be $PI/2$ radians as the initial step size, then generate eight test points in all cardinal and intermediate directions of the $CurrentPoint$. That is, to the north, northeast, east, southeast, south, southwest, west and northwest of the $CurrentPoint$ with the same distance of $TestStep$. If any of these eight points has a smaller total distance than $MinimumDistance$, make this point as $CurrentPoint$ and update $MinimumDistance$. Otherwise, reduce $TestStep$ by half, and continue searching another eight points around $CurrentPoint$ by the new $TestStep$ until $TestStep$ meet the precision (2×10^{-8} radians by default).

We can compute the distance using the spherical law of cosines. If distances between each pair of points are small, we then use distance of spatial straight line between two points to replace circular arc. The equation of the spherical law of cosines is shown as follows.

$$distance = \arcsin(\sin(lat_1) * \sin(lat_2) + \cos(lat_1) * \cos(lat_2) * \cos(lon_2 - lon_1)) \tag{2}$$

2) Improved Method: We observe that in order to select a better initial iteration point, the original algorithm has to compute distances between every couple of points. The time complexity of such selection procedure is $O(n^2)$. In fact, if we remove this procedure (from line 3 to line 9 in Algorithm 1), the time complexity for finding the initial point will be reduced to $O(n)$. It shows a significant improvement for total running time of calculating Geographic Median in our experiments compared to the original algorithm, especially when the point set is large (more than 100 points).

The time cost of the initial point detection procedure in the original algorithm outruns its performance improvement, which is the reason why it should be omitted. However, a good initial point selection schema can potentially decrease the number of iterations and ameliorate total performance. Thus, we design a brand-new algorithm that employs grid segmentation to decrease the cost of searching a proper initial point, which can also reduce the initial step size at the same time. Its efficiency and performance improvement are demonstrated by our experiments. As shown in Algorithm 2, after computing the geographic midpoint, we partition the district into grids with the same size. Since our algorithm reduces at least one iteration, in order to search a better initial point, we take the expense of at most one iteration to select a better initial point from centers of grids. We count the number of points located in each grid as weight, and calculate the total weighted distances from the center of each grid to centers of other grids. If any center is better than $CurrentPoint$ in the sense of less total distance, replace $CurrentPoint$. As the centers of eight neighbor grids of $CurrentPoint$ is not better than its own, we can reduce the initial step to half of distance between two diagonal centers.

Algorithm 3 MapReduce program for KNN

```
function: Map(k,v)
1: $p = context.getPoint()$
2: for each cell c in value.rawCells() do
3:     if column family is "coordinates" then
4:         if qualifier is "minipoint" then
5:             $rowKey = c.row()$
6:             $location = c.value()$
7:             $distance = calculateDistance(location, p)$
8:         end if
9:     end if
10: end for
11: emit (1, $rowKey$ + "," + $distance$)
function: Combine, Reduce(k,v)
12: $K = context.getK()$
13: for (each $v_i$ in v) do
14:     $(rowKey_i, distance_i) = v_i.split(,)$
15: end for
16: Sort all users by distances, and choose K smallest distance
    locations to emit;
```

IV. Data Analysis

In this section, we introduce two common spatial applications, namely, KNN and geometric median distribution. KNN is a method for classifying objects based on the closest training examples according to some metrics such as Euclidean distance or Manhattan distance. KNN is an important module in social media analytics to help user find other nearby users. Geometric median distribution is to count users' distribution in different areas, which might be useful for business to promote products. Due to the mobility of users, the geographic median is one of the best values to stand for users' geographic positions.

A. K Nearest Neighbors

Algorithm 3 shows the process of MapReduce on Hadoop. In the map function, we extract *point* for KNN search and K value from *context*. Then we calculate the distance from each point to *point*, and send top K points to the combine function and then the reduce function. Both of them sort users by distances computed by the map function. The difference between the combine function and the reduce function is that the former sends results to the reduce task, but the later one uploads K nearest neighbors to HBase.

B. Spatial distribution

Algorithm 4 describes how to calculate the distribution of users in a district. In the map function, we obtain the grid length (0.01 degree by default) from *context* to build a mesh on the region of interest, and compute which grid each point locates in. Then the key-value pair of grid index and the number of users is sent to the combine function and afterwards reduce function. The combine and reduce functions sum the number of users in each grid, and finally upload results into HBase.

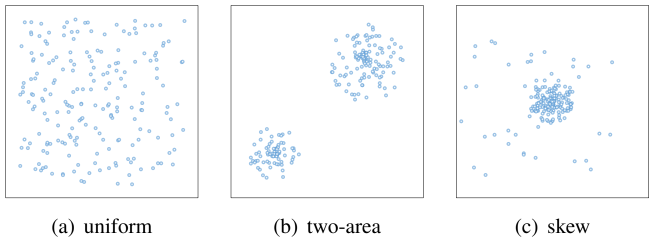

(a) uniform (b) two-area (c) skew

Fig. 3. Data distributions.

Algorithm 4 MapReduce program for geometric median distribution

```
function: Map(k,v)
1: $gridLength = context.getGridLength()$
2: for each cell c in value.rawCells() do
3:     if column family is "coordinates" then
4:         if qualifier is "minipoint" then
5:             $(lat, lon) = c.value()$
6:             $latIdx = (lat - miniLat)/gridLength$
7:             $lonIdx = (lat - miniLon)/gridLength$
8:         end if
9:     end if
10: end for
11: emit $((latIdx, lonIdx), 1)$
function: Combine, Reduce(k,v)
12: $total = 0$
13: for (each $v_i$ in v) do
14:     $total$ += $v_i$
15: end for
16: emit $(k, total)$
```

V. Experiments

In this section, we describe the experiments to evaluate the performance of Dart based on Hadoop and HBase, including the computations on mean center, median center, KNN, and geometric median distribution.

A. Experimental setup

Our experiments were performed on Amazon EC2 cluster that contains one Namenode and nine Datanodes. Each node is an Amazon EC2 m3.large instance, which provides a balance of compute, memory, and network resources. EC2 m3.large instance has Intel Xeon E5-2670 v2 (Ivy Bridge) processors, 7.5 GB memory, 2 vCPUs, SSD-based instance storage for fast I/O performance, and runs CentOS v6.6. Our Hadoop cluster is based on Apache Hadoop 2.2.0, Apache HBase 0.98.8, and Java 6.

In our experiments, we extract Twitter data from 38 degrees North latitude and 73 degrees West longitude to 41.5 degrees North latitude and 77.5 degrees West longitude with a total size of 160 GB and more than 1 million users. This area mainly includes the metropolitan areas from New York City, Philadelphia, to Washington DC. In order to show the performance of our algorithm on a single machine, we generate random points to simulate three common scenarios shown in Figure 3: (1)

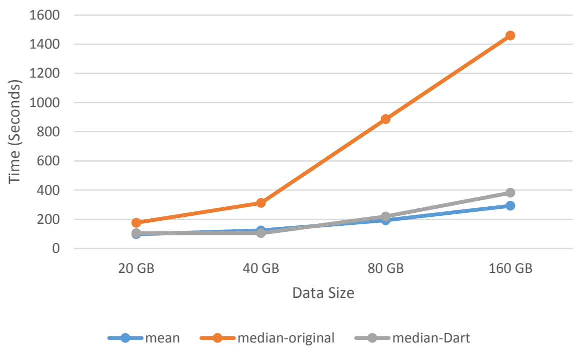
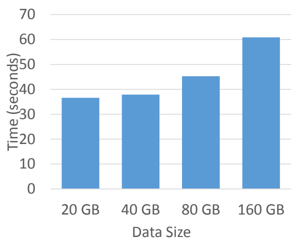
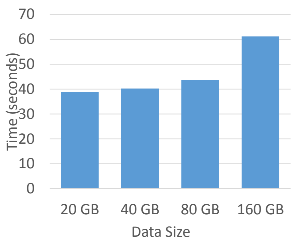


Fig. 5. Performance comparison between mean and median.



(a) knn



(b) distribution

Fig. 6. Results of KNN and Distribution.

Uniform is a scenario where all points are scattered uniformly in the district; (2) Two-area is that all points are clustered into two groups, which is very common in real world. (3) Skew is a scenario where most of points gather together but few points scatter outside.

B. Experimental results

Figure 4 shows the time of calculating the geographic median in three different scenarios. These experiments were conducted on a single m3.large node in Amazon EC2 cluster, and the number of points vary from 100 to 12800. We evaluated three algorithms: (1) The original algorithm most commonly used in geography[11]. (2) The algorithm without-initial-detection, which removes the procedure of selecting a better initial point from the original algorithm as we mentioned in Section III-C. This algorithm reduces the time complexity of selecting initial point from $O(n^2)$ to $O(n)$. (3) Our own grid algorithm, which utilizes grid technique to improve the accuracy of initial point and reduce the step size dramatically at a reasonable time cost. This algorithm reduces the overall iteration number. As Figure 4 shows, when the number of points increases from 100 to 12800, the performance of Dart increases gradually. In all scenarios, our algorithm can outperform the original one by 9 to 11 times when the number of points is 3200. When the number of points is 12800, our new algorithm in Dart gains an improvement of 38%, 26%, and 15% compared to the without-initial-detection algorithm in the uniform, two-area, and skew scenarios, respectively. We find that the without-initial-detection and the grid algorithms cost less time in the skew scenario compared to the uniform and two-area scenarios because the midpoint is closer to the median center.

Figure 5 shows the performance comparison of calculating the geographic mean and geographic median on Hadoop cluster, where the input data size varies from 20 GB to 160 GB. When the input data size is 160GB, our algorithm achieves an improvement of 7 times compared to the traditional one. The calculation of geographic median consumes 30% more time than the calculation of mean on 160 GB data since it is more complex and requires more iterations to approximate the optimal point.

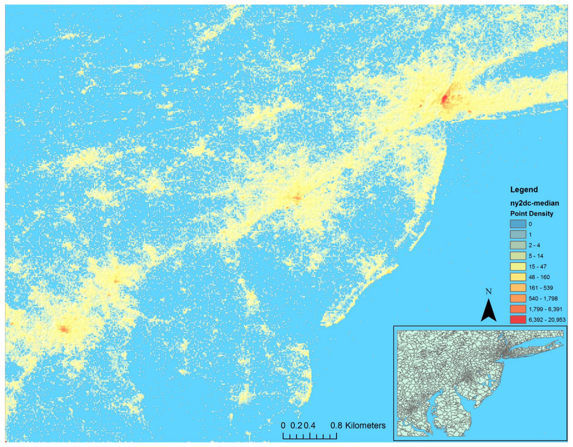
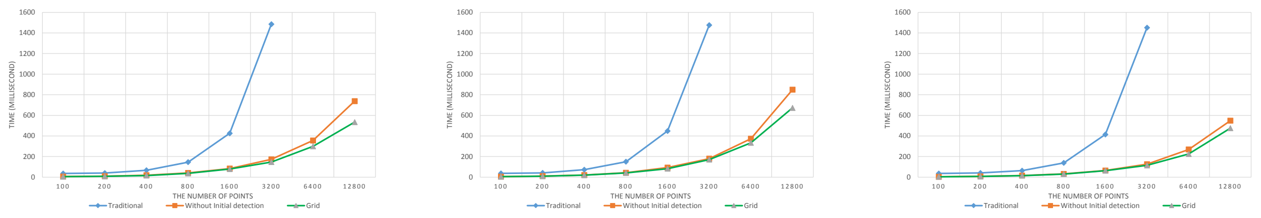


Fig. 7. Results of Distribution

Figure 6(a) measures the performance of computing KNN, where k value is 10, when increasing the input data size from 20 GB to 160 GB. As it shows, the time does not grow linearly, but relatively slowly. Figure 6(b) shows the performance trend of geometric median distribution, which is similar to KNN. We only spend 1 minute to finish KNN and geometric median distribution on 160 GB dataset, which demonstrates that the Dart system is quite scalable.

Figure 7 shows the median centers of all mobile Twitter users discovered from the real tweet dataset, which covers a geographic area from Washington, DC to New York City. The distribution pattern of Twitter users' daily activity locations reveals that Twitter users are more likely to live in urban areas.

The main contribution of this study is in the development of a system for rapid spatial data analysis. The crafting of this system lays a solid foundation for future research that will look into the spatio-temporal patterns as well as the socio-economic characteristics of a massive population, which are crucial inputs for effective urban planning or transportation management.



(a) uniform (b) two-area (c) skew

Fig. 4. Performance comparison of calculating mean and median.

VI. Related work

Some GIS over Hadoop and HBase have been designed to provide convenient and efficient query processing. However, they do not support complex queries like geometric median. A few systems employ Geographic Index like grid, R tree, and Quad tree to improve the processing, which are, unfortunately, not helpful for calculating geometric median efficiently. SpatialHadoop[12] extends Hadoop and consists of four layers: language, storage, MapReduce, and operations. The language layer supports a SQL-like language to simplify spatial data query. The storage layer employs a two-level index to organize data globally and locally. The MapReduce layer allows Hadoop programs to exploit index structure. The operations layer provides a series of spatial operations like range query, KNN, and join. Hadoop-GIS[13] is a spatial data warehousing system that also supports a query engine called REQUE, and utilizes global and local indexes to improve performance. MD-HBase[14] is a scalable data management system based on HBase, and employs a multi-dimensional index structure to sustain an efficient insertion throughput and query processing. However, these systems are incapable of offering a good data organization structure for social network like Twitter, and their index strategies cannot calculate the geometric median efficiently due to an extra load on data management, index creation and maintenance.

CG_Hadoop[15] is a suite of MapReduce algorithms, which covers five different geometry spatial operations, namely, polygon union, skyline, convex hull, farthest pair, and closest pair, and uses the spatial index in SpatialHadoop [12] to achieve good performance. Zhang et al. [16] implements several kinds of spatial queries such as selection, join, and KNN using MapReduce and proves that MapReduce is appropriate for small scale clusters. Lu et al. [17] designs a mapping mechanism that exploits pruning rules to reduce both the shuffling and computational costs for KNN. Liu et al. [18] employs the MapReduce framework to develop a scalable solution to the computation of a local spatial statistic ($G_i^*(d)$). GISQF[19] is another spatial query framework on SpatialHadoop [12] to offer three types of queries, Longitude-Latitude Point queries, Circle-Area queries, and Aggregation queries. Yet, there is no operation or discussion for calculating geometric median on top of Hadoop and HBase.

VII. Conclusion

In this paper, we introduce a novel geographic information system named Dart for spatial data analysis and management. Dart provides an all-in-one platform consisting of four layers: computing layer, storage layer, methodology layer, and data analysis layer. Using a hybrid table schema to store spatial data in HBase, Dart can omit the Reduce process for operations like calculating the mean center and the median center. It also employs pre-splitting and hash techniques to avoid data imbalance and hot region problems. In order to make spatial analysis more efficient, we investigate the computing layer configuration in Hadoop and the storage layer configuration in HBase to optimize the system for spatial data storage and calculation. As demonstrated in Section V, Dart achieves a significant improvement in computing performance in contrast to the performance of traditional GIS. The improvements have been illustrated by carrying out two typical geographic information analyses, the KNN calculation and the geometric median calculation.

For the future work, we plan to extend Dart to implement more spatial operations like spatial join and aggregation. We also plan to extend Geohash or R-tree and incorporate them into our system to speed up spatial search and prune unnecessary location information.

VIII. Acknowledgement

This work was supported in part by NSF-CAREER-1054834 and NSFC-61428201.

References

[1] Twitter from Wikipedia website. http://en.wikipedia.org-/wiki/Twitter.

[2] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2012.

[3] Apache Hadoop website. http://hadoop.apache.org/.

[4] Apache HBase website. http://hbase.apache.org/.

[5] P. Russom. *Big Data Analytics*. TDWI Research, 2011.

[6] D.W. Wong and J. Lee. *Statistical Analysis and Modeling of Geographic Information*. John Wiley & Sons, New York, 2005.

[7] C. Xu, D.W. Wong, and C. Yang. Evaluating the "geographical awarenes" of individuals: an exploratory

analysis of twitter data. In *CaGIS 40(2)*, pages 103–115, 2013.

[8] Amazon EC2 website. http://aws.amazon.com/ec2/.

[9] L. George. *HBase: The Definitive Guide*. O'Reilly Media, 2011.

[10] R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In *VLDB Endowment*, pages 149–158, 2001.

[11] Website for calculating mean, midpoint, and center of minimum distance. http://www.geomidpoint.com/.

[12] A. Eldawy and M. Mokbel. SpatialHadoop: towards flexible and scalable spatial processing using mapreduce. In *the 2014 SIGMOD PhD symposium*, pages 46–50, New York, NY, USA, 2014.

[13] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop-GIS: A high performance spatial data warehousing system over mapreduce. In *VLDB Endowment*, pages 1009–1020, 2013.

[14] S. Nishimura, S. Das, D. Agrawal, and A. Abbadi. MD-HBase: A scalable multi-dimensional data infrastructure for location aware services. In *MDM*, pages 7 – 16, 2011.

[15] A. Eldawy, Y. Li, M. Mokbel, and R. Janardan. CG-Hadoop: Computational geometry in mapreduce. In *SIGSPATIAL*, pages 294–303, 2013.

[16] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Feng. Spatial queries evaluation with mapreduce. In *GCC*, pages 287 – 292, 2009.

[17] W. Lu, Y. Shen, S. Chen, and B. Ooi. Efficient processing of k nearest neighbor joins using MapReduce. In *VLDB Endowment*, pages 1016–1027, 2012.

[18] Y. Liu, K. Wu, S. Wang, Y. Zhao, and Q. Huang. A mapreduce approach to Gi*(d) spatial statistic. In *HPDGIS*, pages 11–18, 2010.

[19] K. Al-Naami, S. Seker, and L. Khan. Gisqf: An efficient spatial query processing system. In *CLOUD*, pages 681 – 688, 2014.

[20] C. Lam. *Hadoop in Action*. Manning Publications, 2010.

[21] E. Sammer. *Hadoop Operations*. O'Reilly Media, 2012.

[22] C. Bajaj. *Discrete and Computational Geometry*. 1988.

[23] Defination of geometric median from Wikipedia website. http://en.wikipedia.org/wiki/Geometric_median.

[24] L. Wang, B. Chen, and Y. Liu. Distributed storage and index of vector spatial data based on HBase. In *GEOINFORMATICS*, pages 1–5, 2013.

[25] K. Wang, J. Han, B. Tu, J. Dai, W. Zhou, and X. Song. Accelerating spatial data processing with mapreduce. In *ICPADS*, pages 229 – 236, 2010.

[26] P. Bajcsy, P. Nguyen, A. Vandecreme, and M. Brady. Spatial computations over terabyte-sized images on hadoop platforms. In *Big Data*, pages 816 – 824, 2014.

[27] L. Duan, B. Hu, and X. Zhu. Efficient interoperation of user-generated geospatial model based on cloud computing. In *GEOINFORMATICS*, pages 1–8, 2012.

[28] A.Aji and F. Wang. High performance spatial query processing for large scale scientific data. In *SIGMOD/PODS*, pages 9–14, 2012.

[29] C. Zhang, F. Li, and J. Jestes. Efficient parallel kNN joins for large data in mapreduce. In *EDBT*, pages 38–49, 2012.

[30] Y. Zhong, X. Zhu, and J. Fang. Elastic and effective spatio-temporal query processing scheme on Hadoop. In *BigSpatial*, pages 33–42, 2012.

[31] L. Alarabi, A. Eldawy, R. Alghamdi, and M. F. Mokbel. Tareeg: A mapreduce-based web service for extracting spatial data from openstreetmap. In *SIGMOD*, pages 897–900, 2014.

[32] M. Trad, A. Joly, and N. Boujemaa. Distributed knn-graph approximation via hashing. In *ICMR*, 2012.

[33] Y. Vardi and C. Zhang. The multivariate L1-median and associated data depth. In *National Academy of Sciences of the United States of America*, 1997.

[34] S. Khetarpaul, S. K. Gupta, L. Subramaniam, and U. Nambiar. Mining GPS traces to recommend common meeting points. In *IDEAS*, 2012.

[35] D.Jiang, B.C.Ooi, L.Shi, and S.Wu. The performance of mapreduce: An in-depth study. In *PVLDB*, 2010.