GPSInsights: An efficient framework to manage and process masssive GPS vehicle data

[Extended Abstract]

Tobin Institute for Clarity in Documentation
1932 Wallamaloo Lane
Wallamaloo, New Zealand
trovato@corporation.com

G.K.M. Tobin Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

Lars Thørväld The Thørväld Group
1 Thørväld Circle
Hekla, Iceland
larst@affiliation.org

ABSTRACT

Intelligent Transport System (ITS) has seen growing interest in collecting various types of location-based data of transport vehicles in circulation in order to build up high quality real-time traffic monitoring system. Managing those massive data faces BigData challenges. In this paper we propose GPSInsights, a framework that can manage and process massive GPS vehicle data effectively. GPSInsights is built on scalable distributed, open-source software with Geomesa performing the key part. We demonstrate our framework with a scalable map matching implementation and perform experiments with big dataset.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Bigdata

Keywords

GPSInsights, spatio-temporal data storage, distributed data processing, map matching

1. INTRODUCTION

With the widespread adoption of GPS technology, Intelligent Transport System (ITS) has seen growing interest in collecting various types of location-based data of transport vehicles in circulation. This data collection is done with the purpose of being able to deliver not only high quality real-time traffic monitoring but also useful traffic statistics and predictive information. Lee et al. [2] a data mining algorithm to discover traffic bottlenecks. Demiryurek et al. [1] proposed an online computation of optimal traffic route based on traffic data.

According to Decree No. 91/2009/ND-CP of Ministry of Transport of Vietnam, all Vietnamese-licensed cars in must be equipped with a standardised GPS monitoring system (black-box) which reports geo-location, speed and direction every 30 seconds to a centralised data centre. With nearly 200.000 cars in circulation in the near future, those collecting data are enormous and have characteristics of BigData. The first characteristic is big volume that refers to the increasing amount of data (petabytes) need to be stored with a relatively low cost. The second characteristic is big velocity that refers to the data generation speed, at which the underlying system must deliver. The third characteristic is big value because mining those data gives insights about the current situation of the traffic infrastructure, as well as, the predictions.

As BigData create non-conventional challenges, current ITS management systems storing data in relational database system (e.g. via PostGIS) will not be able to adapt to the data ingestion speed, nor being able to be mined efficiently. In this work, we describe GPSInsights: a novel scalable system for manipulating and processing GPS traffic data. First, GPSInsights is able to handle big volume of data at the level of petabytes efficiently in distributed environments. Second, GPSInsights deals with big velocity where millions of GPS messages coming from millions of cars in Vietnam are ingesting into the system every 30 seconds. Third, despite big volume and big velocity challenges, GPSInsights can deliver in-time analytic reports for good uses. We demonstrates GPSInsights with a scalable map matching implementation.

This paper is organised in 7 parts. In Section 2, we discuss an overall architecture of our system framework. In Section 3, we go into the details of the technologies and components of GPSInsights. In Section 4, we establish a simple demonstration map matching algorithm. Section 5 presents experimental results for the algorithm presented in Section 4. Section 6 discusses related works on existed map matching algorithms and on storing spatio-time data. We conclude and discuss future work in Section 7.

2. GPSINSIGHTS: SYSTEM DESIGN

In order to deal with the challenges of big volume and big velocity, it is necessary to construct a scalable system with scalable components as following:

Spatiotemporal datastore: As GPS data consist of geolocation information and timestamp in which the data is recorded, GPSInsights favours spatio-temporal queries. At the storage level of GPSInsights is a distributed spatio-temporal data storage aimed at storing the large datasets dispersedly on a large cluster.

Scalable processing engine: After constructing a system scale to store and manage data, we also need distribution and parallel processing engines. The purpose of these is to communicate with the scale for reading and writing data.

Map engine: Map engine is used to allow plotting location-based data for real-time traffic monitoring and for map-based distance calculations in our mining algorithms.

Data importer-exporter: This two components are API servers that expose interface for data ingestion and egression. Data importer accepts data from various source such as JSON, CSV, relational data tables. Data exporter standadises Restful interface for applications, visualization aims and so on.

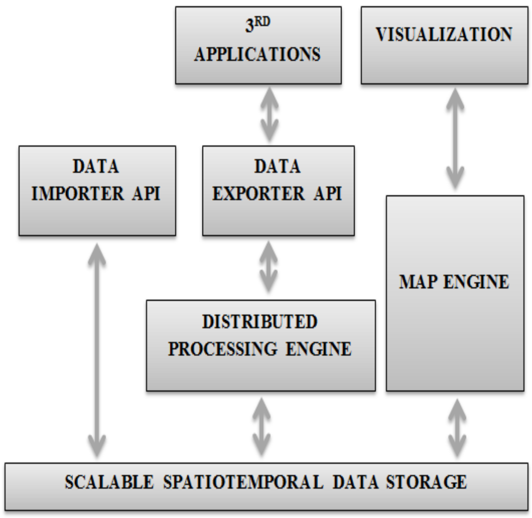

Figure 1: The architecture of the system

3. IMPLEMENTATION

3.1 Hadoop.

The full name of Hadoop is Hadoop Distributed File System (HDFS) [6] is a kind of distributed file system designed to run on commodity hardware. Its construction has a lot of things in common with existing distributed file systems. The fault-tolerant of HDFS is highly assessed and one of the most important thing is that it can be deployed on low-cost hardware. Furthermore, HDFS provide a high throughput access capability to application data which is relevant for applications that have to process large data sets. Besides, streaming access to file system data is made available in HDFS by relaxing a few POSIX requirements.

With our real situation, HDFS is very appropriate with our low-cost hardware.

3.2 Zookeeper.

ZooKeeper [4] is a kind of high-performance centralized coordination services for some form or distributed applications. It provides some traditional services - such as: naming, configuration information management, distributed synchronization, and group services - in a quite low-level. For example, in the configuration management, the actual processing of the configuration changes must be developed as a part of the application. However, ZooKeeper will ensure all clients are notified reliably and the order of configuration messages is maintained. All these kinds of services are distributed and highly reliable.

Zookeeper [5] performs a crucial role in coordination between Hadoop nodes. For example, it makes it easier to:

Manage configuration across nodes. If you have dozens or hundreds of nodes, it is very tough to keep configuration in sync across nodes and quickly make changes. ZooKeeper helps you quickly push configuration changes.

Implement reliable messaging. With ZooKeeper, you can easily implement a producer/consumer queue that guarantees delivery, even if some consumers or even one of the ZooKeeper servers fails.

Implement redundant services. With ZooKeeper, a group of identical nodes (e.g. database servers) can elect a leader/master and let ZooKeeper refer all clients to that master server. If the master fails, ZooKeeper will assign a new leader and notify all clients.

Synchronize process execution. With ZooKeeper, multiple nodes can coordinate the start and end of a process or calculation. This ensures that any follow-up processing is done only after all nodes have finished their calculations.

The functionality provided by ZooKeeper is often developed as part of Hadoop applications. However, these are tricky matters to get right, and it is easy to get errors in the implementation. ZooKeeper provides a solid foundation that helps build higher-level services.

3.3 Accumulo.

Apache Accumulo [7] is an open source database with being a high-performance storage and retrieval system developing distributed and sorted key/value pairs store based on the GoogleâĂŹs Big Table [9] design. It is a system working on top of Apache Hadoop and Apache Zookeeper and having many similarities with an existing database HBase [8]and other no SQL databases such as: Cassandra, MongoDB, Riak and so on in maintaining consistency even as it scales to thousands and more of nodes and petabytes of data, both reading and writing in near real-time.

So what makes Accumulo [1] different from all the rest? Written in Java coding language, Accumulo, hase cell-level access labels (also called cell-level security) that make the Big Data community most excited. Cell-level security allow administrator to assign user access permissions down to the cell-level, which means that administrators can extend the access and functionality of a given database to the maximum figure for users while still remaining in compliance with applicable privacy and security regulations.

3.4 Geomesa.

GeoMesa [10] is an open-source, distributed and spatio-temporal database build on top of the Apache Accumulo column family store. Leveraging a high-performance parallelized indexing strategy, GeoMesa aims at providing as much of the spatial querying and data manipulation - such as storing and transforming spatio-temporal data at scale in Accumulo.

A key-value pair store has only a single index is built with the constraint that all records are ordered lexicographically. It is a problem when we want to map from three dimensions (lat, lon, time) to one (lexicographical ordering of keys in a table). So encoding geo-time information in keys is an essential aim of a geospatial indexing scheme. The indexing strategy of Geomesa is that build index keys by interleaving portions of a point's geohash string that is a 7 character string encoded by Niemeyer's encoding from a 35-bit geohash representing the geometry with part of the datetime string of the format 'yyyyMMdd' and adding a random bin number (between 00 and 99) at the begin of the row to avoid a disproportionate query load and bogging down response times.

Query Planning of Geomesa is described as follows. First, the query polygon is decomposed into GeoHash ranges to scan. Second, the temporal bounds are used to further refine the row range.Finally, we can exactly dertermine every row that must be inspected by using the bin number.

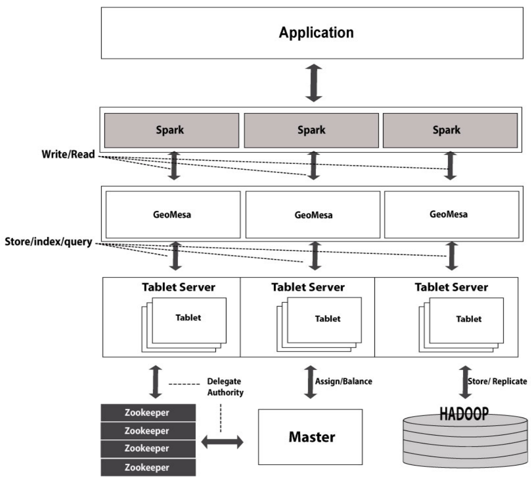

Figure 2: The architecture of the system

3.5 Spark.

Apache Spark is an open-source cluster computing framework [2] [12] processing large-scale data quickly and effectively.

One of the principal reasons for us to use Spark is speed [3]. In stead of running programs and reading data from disk as HadoopâĂŹs two-stage MapReduce paradigm, SparkâĂŹs in-memory primitives will enhance performance up to 100 times faster for certain applications. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is very relevant with our algorithm. Furthermore, Spark offers more than 80 high-level operators with a very simple way to program in Java, Scala or Python, making it easier for us to build up our parallel app. And, finally, Spark has capability to run on Hadoop stably, thereby, being well suited to our system.

4. DEMONSTRATION: SCALABLE MAP MATCHING

4.1 Dataset

Our data include about 12,565,521 GPS records collected by vehicles equipped with a GPS receiver from 22/03/2014 to 22/04/2014 in Ho Chi Minh city. Every record consists of speed, GPS coordinate and state of the vehicle and the period of time between two records is 15 minutes. The following table shows the format of the data.

time_stamp	car_id	lon	lat	speed

4.2 Algorithm: Road Reverse Geocode Algorithm Using K-D Tree.

4.2.1 Process OSM raw data.

The Open Street Map's raw data consist of a mass of tag almost covering the whole world. Every node tag has some tags inside to determine its attributes (e.g. type, way's name, coordinate, ..). Way tag contains one or more node tags that used to define the shape of it.

Before going to the details, vertices, links and segments should be defined clearly. Link is a section of road between intersections. In most digital maps, a real road is digitized and is described with a set of many straight lines. Vertices are points which separate these straight lines, and each straight line is a segment. The road AD in Fig. 3 is formed by 3 intersections (black points), 2 vertices (white points), 2 links (AD, DE) and 3 segments(AB, BC, CD).

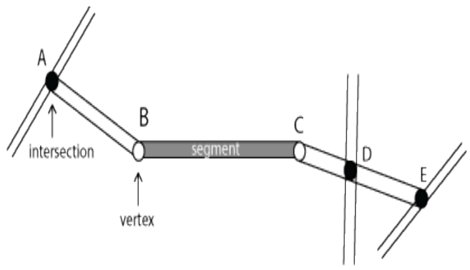

Figure 3: The composition of the road network

Our purpose is to determine all links and their information (name of roads which contain these links). Firstly, we traverse the tag list of OSM, read all Way tags, ignore irrelevant ways (roads that are unfit for transports such as cars, taxis and buses) and then store all associated node ids (we use BTree for this task). If a node occurs once, it is a vertex. Otherwise, it is an intersection. Secondly, we store all Node tags for later uses.

4.2.2 Link matching.

Because the transport data are collected from GPS tracking device, the input consists of a sequence which contains a time stamp and a geographic coordinate. Our task is to assign a GPS point to a relevant link. Kd-tree is chosen to tackle this problem [3]. It is an efficient searching method to quickly find nearest points and this algorithm only takes O(log n) average time per search in a reasonable model.

Despite the fact that the standard deviation of GPS data could be quite low in the best case, around 3 meters, it can increase several fold due to tree cover, tunnel and other problems. The limited sampling polling time intervals is the second source affecting the accuracy. There are many methods to solve quite effectively this, including vertex-based and segment-based map matching, map matching using the geometric relationship, map matching using the network topology, the data history and so forth. This paper does not focus on this problem, we only use the vertex-based map matching for the simplicity of the process.

We go through the list of ways in OSM and add some equidistant vertices in segments (illustrated in Fig. **??**) and build a KD-Tree based on those vertices. Note that a node in the Kd-Tree consists of a coordinate and information of a link which associates it. Therefore, by taking a GPS point into the Kd-Tree and using the vertex-based map matching, we can determine its nearest vertex as well as a link it is matched to.

4.2.3 Describe our algorithm.

In this section, we focus on explaining our proposed algorithm in traffic volume statistic. With this algorithm, we query vehicle object q in a specific time period and a GPS bound from Geomesa aimed at achieving following attributes of q: qâĂŹs latitude, qâĂŹs longitude and qâĂŹs speed. All the information we will store in a data structure list âĂŞ input. After getting input data from Geomesa, now, we review how Spark is used in our algorithm. Firstly, we build up a kdTree with OSM data preprocessed and then broadcast it to the nodes in our system (Line 14). By doing this, we can keep a read-only variable cached on each machine instead of shipping a copy of it with tasks, thereby, providing every node a copy of the kdTree which make up a lot of memory and construction time. Secondly, it is necessary for our algorithm to utilize a collection in-parallel in order to construct a distributed dataset from input data list which can be operated on in paralle. Because the parallel collection can be used many times, we will cache it on memory accelerating our program (Line 15).

As we have a parallelized collection formed, our algorithm will be divided into two phase: a data mapping phase and a data collecting phase.

4.2.4 Data Mapping Phase:

All the elements of the parallelized collection (object q) will be passed on the maptopair method of spark which can operate on in-parallel. As every object q go into the method, the nearestPoint method of copies of kdTree cached on each machine can be utilized aimed at finding what point of a road segment (p) is nearest point of object q and how long distance between q and p (Line 19). Because of the fact that coordination GPS sent from satellites will have some deviations in comparison with the real coordinate, so that we have to choose a threshold distance in order to determine whether object q belong to the road segment including p or not (Line 20 - 27). If the distance is smaller than the threshold distance, it will be much easier for us to conclude that the object q belongs to the road segment and vice versa. As a result, our algorithm can remove the object q that its distance with the road segment is too far from, therefore, we can enhance the accuracy of our algorithm to some extent. Finally, in the Line 29, after every object q is matched to a road segment by road segmentâĂŹs ID, we will group the matched pairs by road segmentâĂŹs ID, hence, we can obtain <key, value> pairs with key being a roadID and value being a iterable of vehicleID and speed.

In the next step in the phase, we will continue to process the <key, value> pairs in-parallel by using the maptopair method of spark. The number of vehicle moving on a road segment also the average speed of the road segment will be calculated in the step (Line 33 - 34).

4.2.5 Data Collecting Phase:

In data mapping phase, statistic numbers of the road segments will be processed in-parallel on the nodes of our system, so that collecting the data perform a decisive role in our algorithm. After the figures are processed, we will use the collect method of spark to convert the parallel collection into a simple list in order to summarize the information and, therefore, we will achieve statistic numbers of traffic volume and average speed on each road segments (Line 38 - 40).

The Pseudocode of Algorithm

5. EXPERIMENTAL EVALUATION

Our system set up on a cluster of HPCC super computer which has one master node and three compute nodes . The configuration of each node include an Intel Xeon E5-2670 Processor (20M Cache, 2.6 GHz, 8 Core), 32 GB DDR3 RAM and FDR 56Gbps Infiniband.

With this configuration, we have conducted several experiments on our data set shown in section 4.1 aimed at appreciating execution time and performance of the architecture. After querying data from Geomesa according year, month, day of month and hour of day and removing records that its speed attribute is 0, we obtained 1,200,000 records. We compared the running time of the system on the whole records and the half records with various numbers of nodes. The result is illustrated by Figure 6.

It is worth noting that as the figure for nodes was increased, the execution time of the system reduced steadily due to the parallel procedure of spark. While effective indexing strategy of Geomesa enable the architecture to quickly

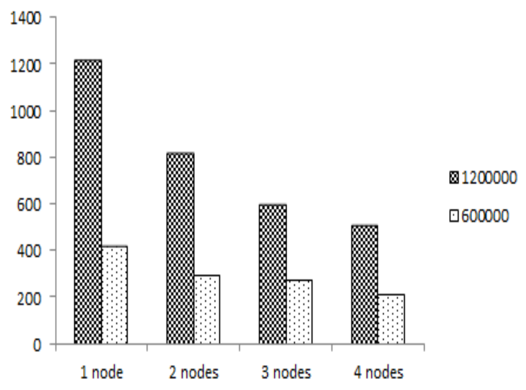

Figure 4: The execution time of the system

query data from Geomesa, spark help the algorithm to accelerate computation by processing in-parallel. As shown in Figure 6, similarity to 600,000 records, with 4 nodes established, the system take a few more than 500s for 1,200,000 records and less than halving of the running time of the system since we only run on 1 node - a considerable decrease. Besides, we installed a common system running without parallel and kind of indexing strategy like Geomesa such as: using Hash Map with normal database in order to have a comparison with our system. With 1,200,000 records, after 20,000s running, there was no signs of stopping of the system.

6. RELATED WORK

PostGIS [5] is a spatial database extender for PostgreeSQL object-relational database, adds support for geographic objects allowing location queries to run in SQl. Remember that the relational database has the ability to use information from multiple indexes to determine how best to search for the records that satisfy all query criteria, so it is good for multi-dimension data rather than a key-value store. It is undeniable that PostGis completely meets most of the spatial-temporal query and there are a large number of software products that can use PostGIS as a database backend [6]. However, spatio-temporal data sets have seen a rapid expansion in volume and velocity due to the rapid increase of means of transport and GPS device, which is about hundred millions and more records per day. With the reason that the capacity of a PostGIS is only about five hundred millions rows, we have to buy more hardware to meet the demand. However, we will have to cope with many issues such as data management, hardware cost and performance. It is clearly that Accumulo and Geomesa are good for tackle big transport data set.

There are a number of studies on matching GPS observations on a digital map. We can generally classify them into two categories [7] Map-matching algorithms using only geometric relationships between GPS data and a digital map and Map-matching consider not only geometric relationships but also the topology of the road network and history of GPS data.

Map-matching in first category can be classified by Noh and Kim(1998) [4] into the map matching algorithm using the distance of point-to-curve. One using the distance of curve-to-curve and one using the angle of curve-to-curve.

Map-matching using the network topology and the data history match the present point to a link based on the result of matching the previous point. However, when the distance between them is higher than a threshold then this method will use only the geometric relationships to match the GPS points to the nearest link.

Map-matching using the network topology and the data history are not appropriate for our GPS data which is described in the section 4. Neither of those two categories has demonstration with a scalable algorithm building with a distributed processing engine.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed an efficient framework to manage and process massive amount of GPS vehicle data. With the framework, we can conduct vehicle data statistics efficiently. Thus, we believe that GPSInsights can address an increasing number of the problem classes relating massive GPS vehicle data. We intend to pursue this framework in three directions. Firstly, we plan to use the framework in order to re-imitate routine of a vehicle with a dataset that the period of time between 2 times recording GPS coordinates is significant. Secondly, we research to use this framework for transport state prediction. And finally, the framework also can be used to build up the fastest path finding system.

8. ADDITIONAL AUTHORS

Additional authors: John Smith (The Thørväld Group, email: `jsmith@affiliation.org`) and Julius P. Kumquat (The Kumquat Consortium, email: `jpkumquat@consortium.net`).

9. REFERENCES

[1] U. Demiryurek, F. Banaei-kashani, and C. Shahabi. *A Case for Time-Dependent Shortest Path Computation in Spatial Networks*. Advances in Spatial and Temporal Databases Lecture Notes in Computer Science, 2010.

[2] W.-H. Lee, S.-S. Tseng, J.-L. Shieh, and H.-H. Chen. Discovering Traffic Bottlenecks in an Urban Network by Spatiotemporal Data Mining on Location-Based Services. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1047–1056, 2011.

[3] M. Otair. Approximate k-nearest neighbour based spatial clustering using k-d tree. *International Journal of Database Management Systems*, 5(1), 2013.

[4] Noh and Kim. A comprehensive analysis of map matching algorithms for its. *Hongik Journal of Science and Technology*, 9:pp. 303–313, 1998.

[5] PostGis. http://postgis.net/.

[6] Wikipedia. User section: http://en.wikipedia.org/wiki/postgis.

[7] Y. Jae-seok, K. Seung-pil, and C. Kyung-soo. The map matching algorithm of gps data with relatively long polling time intervals. *Journal of the Eastern Asia Society for Transportation Studies*, 6:pp. 2561–2573, 2005.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations

Display Equations

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The acm_proc_article-sp document class file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LATEX, you may find reading it useful but please remember not to change it.