Exploring Movement Patterns in Geospatial Big Data

By

Mudabber Ashfaq

(NUST201362931MSCEE62513F)

A thesis submitted in partial fulfillment of the requirements for degree of Master of Science in Remote Sensing and GIS

Institute of Geographical Information Systems
School of Civil and Environmental Engineering
National University of Sciences & Technology
Islamabad, Pakistan

August, 2016

CERTIFICATE

Certified that the contents and form of thesis entitled **“Exploring Movement Patterns in Geospatial Big Data”** submitted by Mr. Mudabber Ashfaq have been found satisfactory for the requirement of the degree.

Supervisor: ________________

Dr. Ali Tahir

(Assistant Professor, IGIS, NUST)

Member: ______________

Dr. Muhammad Azmat
(Assistant Professor, IGIS)
SCEE, NUST

Member: ______________

Ms. Quratulain Shafi
(Lecturer, IGIS)
SCEE, NUST

Member: ________________

Mr. Asim Mushtaq
(CEO, Magma Systems)

ACADEMIC THESIS: DECLARATION OF AUTHORSHIP

I, Mudabber Ashfaq declare that this thesis and the work presented in it is my own and have been generated by me as the result of my own original research.

Exploring Movement Patterns in Geospatial Big Data

I confirm that:

1. This work was done wholly by me in candidature for an MS research degree at the National University of Sciences and Technology, Islamabad.

2. Wherever I have consulted the published work of others, it has been clearly attributed.

3. Wherever I have quoted from the work of others, the source has been always cited. With the exception of such quotations, this thesis is entirely my own.

4. I have acknowledged all main sources of help.

5. Where the work of thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

6. None of this work has been published before submission. This work is not plagiarized under the HEC plagiarism policy.

Signed: ……………

Dated: ……………..

DEDICATION

"To my loving mother & father who taught me how to read and write."

“Also to my wife who supported me throughout my thesis. May Allah Bless them All.”

ACKNOWLEDGEMENTS

All praises to Allah to whom all knowledge belongs and who is knower of unknown. All thanks to Allah who blessed me with guidance and knowledge to achieve this. His boundless mercy and blessings are foremost and everything. My all gratitude belongs to Him and only to Him.

My thanks to my family who believed in me and prayed for my success especially, my wife who kept me motivated in my journey to completion. I acknowledge all the sacrifices they made in terms of time, finances and my unavailability for them. My very special gratitude for my mentor Dr. Ali Tahir who not only supervised me for the work but also be kind enough to extend his support in every problem I came across. His farsightedness and visionary judgment made this day possible. Also it was only due to his personal efforts that I was able to submit articles in various conferences. May Allah reward him in return. My thanks to Dr. Azmat, Miss Quratulain and Mr. Asim Mushtaq for guiding me and helping me out in fulfilling the thesis requirements. Their respect, courtesy and availability are highly acknowledged. May Allah bless them for their kindness.

I extend my special thanks to Dr Ralf and Mr. Jiamin, University of Hagen, Germany & his SECONDO research team for the help he extended me in learning SECONDO. My gratitudes to Dr Gavin McArdle, National University of Ireland & Michela Bertolotto , University College Dublin who co-authored SDH conference paper with me and made themselves available in completion and critique of my work. I am also thankful to all those whom I haven’t mentioned here but they were part of their altruistic support directly or indirectly. I acknowledge all of their sincere efforts.

Mudabber Ashfaq

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURES

ABSTRACT

Due to the daily increase in volume of geospatial data, the need for efficient ways of storing and processing such a massive amount of data is becoming a challenge. The geospatial data usually contain raster satellite images, points of locations, linear features like roads, polygon representing area boundary and trajectories of moving objects etc. There is a need to explore different methods to handle and process big geospatial data which is an emerging issue. Different techniques and tools are available to handle large geospatial data. Hadoop platform is among the leaders which provide big data solutions, However Hadoop lacks spatial support. Several extensions are being developed over Hadoop platform for spatial data processing. In some cases, the spatio-temporal sequences, usually obtained from GPS and smart phones moreover contribute to very large dataset. These datasets typically form trajectories. For trajectories, SECONDO, is quite useful as it contains tools to handle trajectories. However when trajectories increase in volume it exceeds the capacity of a single computer processing hence it should be handled in the context of big data. Hadoop based platform of SECONDO, named Parallel SECONDO, which extends the functionality of SECONDO with the support of Hadoop is used to deal with massive trajectory data. This study focuses on exploring the abilities of Parallel SECONDO to analyze the patterns in trajectories of T-drive which could exceed the processing ability of single system. Finally the time effectiveness will be compared for tasks performed by single computer with Hadoop parallel processing using number of nodes. Our approach will be beneficial for testing and proposing geospatial big data solutions.

Chapter 1

INTRODUCTION

"Time is money" a well quoted words that show the importance of time, resembles the recent trends. Everyone wants to get their work done in the minimal time and for this purpose human created machine to perform their tasks which saves time and physical effort. New technologies overcome the deficiencies of previous technologies and improve the time efficiency. This is the era of sharing information and knowledge. This sharing of information produces a massive amount of data which needs to be stored and analyzed to improve daily life style. This brings the challenge for scientists and researchers, how to manage this huge data? how to extract the useful knowledge from this data? how to use the data in order to improve services and human lifestyle? These are the questions that we need to answer because in this time saving era conventional methods are not very helping. Hence we need new technologies like that of big data analytics or parallel processing. Such data that exceeds the computation abilities of the current technology is known as "Big Data" (Chen & Zhang, 2014).

Handling big data requires high performance computing or distributed data processing. The state-of-the-art industrial standard is the MapReduce model (Dean and Ghemawat, 2008). The framework of Apache Hadoop (Murthy et al., 2011) is its open-source implementation. The original aim of the MapReduce paradigm was to process simple text documents. However the implementation of complex algorithms and the management of heterogeneous data structures was a challenging task. To counteract this, several extensions and toolkits have been introduced that operate over the Hadoop platform enabling a wide range of data management, mining and analysis possibilities.

With the growing availability of ubiquitous mobile computing devices such as smart phones equipped with GPS, the amount of mobility data is increasing. Typically mobility data contains both spatial and temporal data which form trajectories representing a time-stamped path of an object through space. Movement of these objects contain hidden patterns which reflect the behavior of these entities. As we know almost everything including people, animals, vehicles etc is in motion. Movement of these objects contains hidden patterns due to some behavior and properties of the object which concludes in a very informative knowledge. Using this knowledge mankind could improve the way they live their life. For example keeping track of person's path, companies could analyze the interests of that person and how to facilitate him with their products could be achieved. In case of vehicles trajectories could be used to identify the best routes to reach a particular destination by using expert driver knowledge and tracking companies could direct other vehicles to follow that perfect path at that particular time.

Trajectories of different objects could be quite beneficial for identifying patterns, hotspots and behavior of the subject. There is a wide range of trajectory data which is available freely to download and perform analysis on it. Movebank[1], which is a free portal for all the trajectories of animals and birds in the world carried out by different researchers (Kranstauber et al., 2011). The data sets are freely available to download and analyze.

1. http://www.movebank.org/

The use of location aware technologies such as GPS in vehicles, bicycles and planes is quite common. This continuous data is ever increasing which may not be processed on a single machine. For example, New York Taxi cab data[1], which is freely available and consist of almost 2.5 GB of trajectories. ESRI's ArcGIS on a simple dual core processor computer could not even plot this data without crashing. Therefore to process such an immense data requires useful tools and distributed framework. There are several datasets of trajectories available online for free downloading such as GeoLife GPS trajectories[1], T-Drive dataset[2] (which are part of Microsoft Asia research) and CRAWDAD[3].

While several datasets are available spatio-temporal queries are commonly used to identify patterns in these datasets. There are several DBMS which provide support for spatial operators however only few specialized ones provide support for both spatial and temporal data processing. Secondo (Guting et al., 2005) and Hermes (Pelekis et al., 2006) are two examples. The nature of movement data means that its size can become very larger, and processing and querying it become slow and inefficient. To handle such instances, there are moving object database platforms which support parallel query processing.

Distributed Geographical Information Processing (DGIP) has many benefit, however, its application are not straightforward. As cloud computing became prominent, several steps were made to advance GIS to the cloud serving as a basis for spatial cloud computing.

Parallel Secondo, a Hadoop based platform is a promising tool to handle big mobility data. It combines the distributed processing author ability of Hadoop and the useful analytical capabilities of Secondo to store and process trajectories. Parallel Secondo

1. http://www.andresmh.com/nyctaxitrips/
2. http://research.microsoft.com/
3. http://crawdad.org/index.html

provides hybrid processing where analysis can be run on both sequential and parallel modes depending on the available distributed architecture. The learning curve of SECONDO is high while Parallel Secondo can be less challenging in comparison to other parallel based systems as this is designed for audience with relatively less technical background (Güting & Lu, 2015). It depends on the Hadoop framework to assign and program tasks running on a computer cluster in parallel, however the tasks' embedded procedures are then managed by SECONDO for the sake of efficiency. Parallel SECONDO inherits the capability from SECONDO to process and analyze spatio-temporal data.

Parallel SECONDO set Data Server (DS) as its basic processing unit on each cluster node and even low end computer could be used as nodes. Among all nodes one is specified as Master Data Server (mDS) and others get to know as Slave Data Servers (sDSs). Parallel queries are processed strictly on the MapReduce standard base. First they are converted into Hadoop jobs by the master database. The tasks are processed in parallel on all sDSs and in order to attain a balanced amount of work on the cluster following the MapReduce model. Every task fetches its essential data from either the local slave database or remotely from the other computers via PSFS (Parallel SECONDO File System).

Parallel SECONDO can still be used in conventional way of single computer due to the front-end SECONDO algebras implemented in new system. To the best of our information, for the mobility data processing Parallel SECONDO could be considered as the first practical and efficient parallel system for movement data analysis. In Parallel SECONDO, parallel queries are only giving in the master database there is no necessity to retype the queries for all the contributing systems. They are listed in SECONDO executable language and later transformed to Hadoop jobs. Parallel SECONDO is freely available for download Like SECONDO. However at the current time SECONDO fails to establish query optimizer supporting the formulation of SQL scripts both for querying and updating the database (Di Felice, 2012). SECONDO is quite in a development phase hence it is neither stable nor easy to use.

1.1 Motivation

Day to day emerging various internet services and social media incorporation results in massive amount of geospatial customer information that is stored and collected continuously. To find patterns that gives useful knowledge and regularity from such massive data could improve the way of dealing various things like navigation, customer feedback and requirement. As an example each day before 2013, 618 million users were active on Facebook contributing new data that exceeds 500 TB of volume (Lu, 2014). This data might not be entirely useful regarding geospatial analysis however extracting our required information is a task that needs attention and efficient cost effective solution for smaller companies to improve their services which could be achieved using parallel processing paradigm on cluster computers. Facebook's cluster scans 105 TB data every 30 minutes and has the ability to store 100 PB data in total.

On the other hand geospatial data has always been big data. These days, large data analysis for geospatial data is receiving considerable attention to allow users to analyze massive amounts of geospatial data. Geospatial big data typically refers to spatial data sets exceeding capacity of current computing systems. McKinsey Global Institute says that the pool of personal location data was in the level of 1 Peta Byte (PB) in 2009 and is growing at a rate of 20% per year. This estimation did not include the data from RFID sensors and those stored in private archives. According to the estimation by United Nations Initiative on Global Geospatial Information Management (UN-GGIM), 2.5 quintillion bytes of data is being generated every day, and a large portion of the data is location-aware. Furthermore, in Google, about 25 PB of data is being generated per day, and a significant portion of the data falls into the realm of spatio-temporal data. This trend will be even accelerated since the world becomes more and more mobile in these days. In India, the internet traffic from mobile devices already exceeded that from desktop computers (Lee & Kang, 2015).

Along with this exponential increase of geospatial big data, the capability of high performance computing is being required greatly than ever, for modeling and simulation of geospatially enabled contents. However, because of limited processing power, it has been hard to fully exploit high-volume or high-velocity collection of geospatial data in many applications. Recently, distributed, parallel processing on a cluster of commodity computers or a cloud such as Amazon EC21 has been becoming widely available for use, breaking the existing limitations on processing power.

However parallel processing is not always a solution and its efficiency becomes valuable only if certain factors are catered before choosing this option. One of the main factor is the data size. Data size drives the necessity to run queries in sequential or parallel mode. Processing queries using more than one node may increase time efficiency; however the extent of efficiency increase includes many factors such as volume of data, nature of query and number of nodes. This paper proposes an appropriate environment to achieve efficiency by using the optimum amount of computation power which can reduce cost.

Another aspect of this study is to analyze the movement patterns and find some useful information which could help the tracking services in providing navigation to customers. This will increase mobility efficiency. Therefore this study could have a huge commercial application. Many techniques have been introduced in this aspect already. However mostly the techniques take in consideration of the road network and its conditions. Few techniques analyze the historical trajectories of vehicles or as in our case taxi's. A massive data is stored in databases of tracking companies without any analysis due to the processing limitations. This data could be utilized to gain information taking into account the expertise of the taxi drivers as generally they are among those people who knew best about the city roads and routes to reach their destination. This analysis has many applications in tracking and navigation systems which needs attention especially in Pakistan as there are very few tracking companies to provide efficient navigation systems. These technologies should be promoted.

Considering all these latest developments, a thorough study is needed on how to utilize these distributed processing efficiently on spatial data and trajectories to analyze the patterns in the data to be used in our daily life.

1.2 National Needs

We are living in an era where information and data is flooding every second. Soon we will be out of technology to process such immense data if we do not start utilizing some recent big data handling techniques like parallel processing. Our study will determine an efficient provide a solution to process big data efficiently within small budget without having to purchase high speed computers as this process could be done with several low end computers. In Pakistan following fields could benefit from it.

- Pakistan National Spatial Data Infrastructure (NSDI) is in process of developing which will link several government departments as well as Non-Government Organizations (NGOs) and Universities. Data exchange between departments will be possible. This could make access to very large amount of data from different departments which will need high performance computing with low cost.

- Trajectories data from different department’s vehicles like Rescue 1122, PIA, Radio Cab Services, or Military could analyze easily with less time to process it. Many patterns can be identified which will help them commercially and improve their efficiency.
- To our best knowledge no such techniques of spatial moving big data are being implemented on research level or commercial level which increases the importance of this study to benefit the countries tracking system in order to develop as a nation.

1.3 Background

Big data’s importance has increased globally with the increase in communication and technology for data sharing; this attracts academic and industrial/organizational attention. “Big Data” term got familiarized in scientific communities during mid 1990’s due to the introduction of internet which provided platform for data sharing and information gathering. This increased the concern about the integrity, continuity and management of the data. Gradually Big Data’s popularity increased until 2010 when it started to become a necessity for Multi-National organizations. Presently this is a buzzword on internet, in trade market, in scientific community and during conferences of every type. According to Manyika et al., (2011), big data provides innovation productivity and competition. Data growth due to technological evolution and invention of multiple sensors created a mash of complexity and opportunities for academic as well as scientific community to present innovative techniques that can ease the situation we deal at present. Big data caused research aspect to shift its paradigm towards data-driven research. Presently Geospatial Big data presents limitless applications in the field of climate change, disaster response, disease surveillance, determining different species of trees/rocks using hyperspectral imagery and transportation. However data's privacy, security and confidentiality limit the benefits of big data towards society.

Different industrial, technological, academic or research point defines big data distinctly. This makes big data term unclear and complex for several fields (Chen et al., 2014). Considering the complexity of massive structured and unstructured datasets presents a challenge to store, analyze, manipulate and visualize keeping in view the available hardware and software technologies. Big data's unique characteristics were first proposed by Laney (2001) describing three dimensions Volume, Velocity and Variety (3Vs) that characterize the opportunities and challenges of increasing data volume. Increasing technologies and data complexity makes addition of more dimensions essential and important. Therefore for data integrity and quality could be catered with adding veracity to the list of dimensions. Some more Vs such as validity, visibility, variability, value, volatility and visualization have been suggested however they lack the ability to express quality of magnitude which brought them at critical evaluation as they do not directly explains the definition of "big" in big data, on the other hand they could be useful on the concept of big data collection, presentation and processing. This argument turned to a new debate when Suthaharan (2014) argued about the original 3Vs support for detection of big data characteristics and classification is not enough and liable on early stages. Therefore instead he proposed 3Cs: continuity, complexity and cardinality. This made defining Big Data or its characteristics an ongoing endeavor. However this hopefully will not affect the processing and handling of big data.

A discussion carried out by Morais (2012) presented an argument according to which “80% of data is geographic”. The basis of this argument was according to the fact that most of the data can be geo-referenced and should be considered as the geospatial data hence handling of geospatial big data gains importance more than we think. Geospatial data usually refers to such objects or relations to which we could assign some coordinates according to a particular reference system of Earth. Sources of geospatial data include ground surveys, remote sensing, photogrammetry, laser scanning, geolocated sensors, mobile mapping, geo-tagged contents, Global Navigation Satellite System (GNSS) tracking. For geospatial data, three main Vs could play vital role of characterization and defining the data however other Vs may have small influence however could be quite useful and relevant for certain characteristics. Therefore the most influential Vs for geospatial big data could be defined as following:

1) **Volume:** The amount of data stored in archives/disks that includes remote sensing imagery/Satellite imagery that is stored since 1960 the start of the first meteorological satellite TIROS-1 and ERTS-1(later named Landsat1) in 1972 makes volume of the images beyond processing ability of regular systems if one wants to analyze time series analysis. Technological evolution produces better resolution as well as big volume Furthermore new sensors which record geospatial data 24/7 creating a massive amount of data like GPS trackers in Vehicles or attached to animals for their monitoring. Storing such data is a big issue that has to be sorted out.

2) **Variety:** Geospatial data consisting of satellite/airborne platform imagery, ground surveyed maps, geotagged imagery/text data, other raster and vector data being structure or unstructured makes a complex structure due to its various data types that exhibits challenges to present a single solution to cater every type of data. Hence efficient models, indexes and data management strategies according to the latest technology are required.
3) **Velocity:** The continuity of the data depends on the intake of the data and the speed of its acquisition like fro remote sensing data the number of satellites and their revisit time makes the incoming of the data a challenge to cater. Contents on Internet are updated every second and must be dealt with accordingly, the real-time GNSS trajectory of million vehicles/animals pose a challenge to process. Therefore an efficient system is needed to meet the requirement of the data processing of commercial or organizational field.
4) **Veracity:** Most of the time geospatial big data's source is unknown and is difficult to verify the authenticity of the data and its level of accuracy varies depending on the source hence quality assessment of the data and its improvement statistically is an issue to be resolved using new technologies like Big Data.
5) **Visualization:** Showing data on maps or through graphic visualization or animation human mind could perform better analysis using its expert knowledge. Visualization supports analyst's perception to identify patterns like outliers in trajectories or clusters of same characteristics which will eventually leads to new hypothesis along with efficient ways to partition the data for further analysis computationally as well as statistically. With the increase in the volume and variety

of the data, visualizing the data has become a challenge due to computation limitations. Therefore an efficient system could be used to improve the visualization which will help end users to determine dominant patterns and relationships that emerge from analysis of big data.

6) **Visibility:** Due to the intervention of the cloud computing and cloud storage the accessibility and processing of the big data has efficiently improved in certain ways which were not possible before. This technology has not been matured enough and its evolution period is still on its way hence we find multiple issues regarding Big Data such as data provenance that is the metadata of the historical data. Once these issues get resolved we could finally experience the best outcomes from the combination of big data and cloud computing as they could be mutually dependent (Lu, 2014).

These were some of the challenges that users are facing due to these qualities capacity for conventional spatial computing technologies to handle such big data is becoming an anchor which could weigh down the users time if new technologies are not utilized. With the help of these technologies and Big Data encourage people to expect more and think big beyond the box, make bigger hypothesis that increases the statistical strength of data and capacity to analyze that data critically and effectively.

Technologies like conventional RDBMS systems falls short in effectively handling such exploding data. The main technology gap is due to the limitation of computation power of conventional RDBMS of single computer hence it could only carry sequential queries on a single system with limited capabilities. With improving hardware abilities computation power furthermore increases analysis shows that data is growing at a faster rate than hardware efficiency. There are some hardware present which has the ability to handle present big data problem however they will cost the user a fortune which most of the organizations and companies tries to avoid.

Therefore need of parallel system has become imminent. Such systems which could combine the power of low end systems and provide same results as we could get from performance of super computers. These systems will be both processing and cost efficient. However not all parallel systems or database has the ability to handle such amount of data like Vertica for example (Lu, 2014). Some of the reasons for their in ability is due to the fact that parallel databases are generally designed assuming the probability of failure (software or hardware) is rare in clusters. However this probability increases if more systems are added into cluster. Hence the one advantage that parallel systems enjoy becomes limited and that is scalability. Clusters with more than hundred will possess danger of collapsing making the system vulnerable towards failure while exploding big data's requirement could reach to thousands of computer power.

Above cited issues and problems provide opportunities for researchers to head towards a more scalable parallel processing mechanism to remain in the race against big data growth. Eventually new improved platforms like MapReduce, SCOPE, Dryad etc. are introduced which caters the problems facing earlier. They are intended to provide a flexible infrastructure through the network, data storage distributiveness without following the relational data model. Furthermore their main focus is not on providing efficient performance on clusters reaching hundreds in number rather they keep the system's fault-tolerance high in order to maintain integrity. These platforms has successively attracted industries and research communities due to their efficient performance for large scale data analysis. Hadoop MapReduce is particular attention gaining approach worldwide for processing big data.

1.3.1 Hadoop Map-Reduce Environment

Google proposed Map-Reduce Environment in 2003. Whereas the foundation of Hadoop was laid by Yahoo employee Doug Cutting and his co-partner Mike Cafarella a graduate student of University of Washington. They supported the concept of Map-Reduce Environment and introduced in their own search engine which was first named as "Nutch". Now Hadoop is a part of Apache. This system has helped several organizations like Google, Yahoo, Facebook etc. process data in the volume of Peta Bytes using clusters consisting of thousands computers spread across different data centers operating 10,000 distinct programs that includes algorithms for large-scale graph/text processing, statistical machine translation and machine learning (Lu, 2014). This programming model is relatively simple in comparison with other solutions. The functionality of this model starts by first implementing Map function which process a <*key,value*> pair generating a set of intermediate <*key,value*> pairs which is then processed by Reduce function to evaluate all values associated with the same intermediate key. Original MapReduce platform proposed by Google is kept private, however Apache Software Foundation an open source project implemented their technique and released it for general public use.

The Hadoop core does not support spatial data properties hence Hadoop lacks the ability to process spatial data efficiently. Several extensions handle different kind of spatial data and as a result more than one extension of Hadoop could be used in a system.

SpatialHadoop, Hadoop-GIS, GIS Tools for Hadoop, HIPI, MrGeo and AEGIS are some of the Hadoop extensions currently available to handle raster or vector dataset.

- **SpatialHadoop** is a MapReduce extension which enables the processing of spatial data with ongoing development. SpatialHadoop extends functionality with efficient spatial data storage, indexing, and spatial query support. It is implemented in Java, and has no external dependencies. As a library it can be easily integrated into the Hadoop framework. Furthermore it has an integrated spatial extension to Pig, called Pidgeon. SpatialHadoop offers support for vector data in form of shapes. Three vector types (point, rectangle and polygon) can be easily specialized, however introducing new types requires modification of other components (Eldawy, 2014).
- **Hadoop-GIS** is a spatial data warehousing system, which is furthermore available in library form. Initially developed for medical applications, it contains a custom query engine known as RESQUE and integration to Hive. The tool is developed in Java, C++ and Python, with external dependencies for GEOS and libspatialindex. Hadoop-GIS supports vector data using the GEOS library (offering the basic OGC SFA geometries). It does not support modification or introduction of new types (Giachetta, 2015).
- **GIS Tools for Hadoop** provides a toolkit for managing and processing spatial data with the primary aim of fitting ArcGIS with Hadoop support. It provides extensions to Hive, and multiple geoprocessing tools. The toolkit is developed in Java, however some functionalities require ArcGIS software. GIS Tools for Hadoop provide the support for the most spatial storage formats by exploiting proprietary ArcGIS software (Gao et al., 2014).

- **HIPI – Hadoop Image Processing Interface** is a general toolkit for performing image processing on a collection of images. It is furthermore developed in Java. Although the tool does not offer processing operations, it enables the handling of multiple images in a variety of formats, thus providing easier image access and manipulation using standard MapReduce operations. HIPI supports raster imagery in the form of floating point value arrays. Images can be grouped to form image bundles, which serve as the primary input for processing.
- **MrGeo – MapReduce Geo** is the most recently released geospatial raster processing library on Hadoop. It is primarily developed in Java with multiple dependencies, including GeoTools and Apache Giraph. Currently, it provides the most extensive raster processing support over MapReduce with ongoing development. MrGeo relies on Java raster representation with additional support for geographic location.
- **AEGIS** is a platform independent library, implemented using NET/Mono Framework to exploit the wide possibilities and the simple usage of this object-oriented development platform. AEGIS supports both raster and vector data formats (vector and raster). Vector data is modeled using the OGC (Open Geospatial Consortium) SFA (Simple Feature Access) standard. The data model is abstract and can have different realizations, and there are wide possibilities for introducing new types. Additionally, reference system management is supported according to OGC SRC (Giachetta, 2015).

1.3.2 Parallel SECONDO

Another extensible database system developed at the FernUniversität in Hagen that allows the user to handle non-standard data and applications especially ones containing spatio-temporal information, is Secondo (Lu, 2014). This database system has the specialty in handling mobility data (data containing continuous spatial coordinates as well as time information). In addition, Secondo could furthermore be utilized for handling other non-standard data types like vector and raster images. This is an open source system and its source code is freely available for users to access and modify database system according to their use and environment.

At the backend of Secondo, BerkeleyDB is used as a store manager which could be run on any operating system like Windows, Linux and Mac OS. This system has three major components SECONDO kernel, Optimizer and Graphical User Interface (GUI) which are written in different languages.

SECONDO kernel utilize algebra modules for implementation of specific data models and furthermore allow user to query from that model and perform analysis. Its implementation is on top of BerkeleyDB written in C++.

Optimizer has the ability to optimize the queries according to the user's requirement. Currently it is restricted for relational environment. However Conjuctive query optimization has the abilities to handle any data models. SQL like language is supported in Optimization in such a way to adapt PROLOG as optimizer is written in it.

Graphical User Interface (GUI) enables users to visualize the data especially moving data and furthermore has the ability to animate the trajectory of that moving object.

Furthermore it could enable users to plot there spatial data with the background layer of OSM, Google Map tiles or their own imagery. Java is the scripted language of GUI.

Secondo with all its abilities to handle spatial moving data has the drawback as most of the conventional systems, the lack of distributed processing abilities or to handle Spatial Big Data. However as we discussed earlier about the needs of the research community for data models that allow user to increase the efficiency of query execution time for their analysis. Hence for this purpose extended version of Secondo was developed named as “Parallel Secondo”.

A hybrid parallel processing system, parallel SECONDO built upon Hadoop and a set of SECONDO databases. It depend on the Hadoop framework to assign and program tasks running on a computer cluster in parallel, however the tasks’ embedded procedures are then managed by SECONDO for the sake of efficiency. From SECONDO, Parallel SECONDO inherits the capability of processing special data types, like spatial and moving objects.

1.1.3.1 Merits

User can use Parallel SECONDO still like on a single computer because of the new system keeps the front-end and the executable language of SECONDO. To the best of our information, for the mobility data processing Parallel SECONDO is the first practical and efficient parallel system. In Parallel SECONDO, parallel queries are only give in to the master database there is no necessity to retype the queries for all the contributing systems. They are listed in SECONDO executable language and later transformed to Hadoop jobs. Parallel SECONDO is freely available for download Like SECONDO and could be optimized according to the users need. Source code is available for the developers to customize according to their environment.

1.1.3.2 Demerits

At the current time SECONDO's failure of established query optimizer supporting the formulation of SQL scripts both for querying and updating the database (Paolino Di Felice, 2012). SECONDO is quite in a development phase hence it is nor stable neither easy to use.

1.4 Objectives

"To provide an efficient solution to handle big mobility data and analyze the hidden patterns in T-drive dataset to improve tracking and navigation."

- To use T-Drive dataset in order to find movement pattern through customized spatio-temporal queries.
- To run the queries on single node and multiple nodes in order to compare and measure performance.

1.5 Literature Review

Literature review consists of four sections containing big data challenges and opportunities, tools used by different researchers to handle big spatial data, critical appraisal on SECONDO Parallel SECONDO, available trajectory datasets and their case studies.

Lee and Kang (2015) discuss the challenges and opportunities of big data. Of the 15 global challenges faced by humanity they consider, 7 challenges are directly related to spatial information acquisition and planning. These challenges include energy, clean water, health, population and resources, transnational organized crimes, and peace and conflict. These challenges require the analysis of massive spatial datasets and analytical information. They discussed how spatial data could be used efficiently and in what applications. Furthermore they pointed out some data repositories like Movebank, Foursquare and OpenStreetMap etc. These data sets could be useful in extracting valuable information. Moreover they mentioned new technologies like Spatial on Line Analytical Processing (SOLAP). They discussed about the efforts researchers are putting in this field and challenges they are facing. On the other hand Vitolo et al., (2015) discussed the implementation of environmental big data analysis through web technologies which will help the environmentalists to analyze daily created weather reports and satellite images of weather situations.

Recognizing the role of spatial data, the framework of Hadoop (Murthy et al., 2011) has been adapted and extended to process big geospatial data and several efforts in this area are ongoing. For example, ESRI, a supplier of Geographic Information Systems has added Hadoop support in their ArcGIS products via the ESRI Geometry APIs which spatially enable the Hadoop cluster for scalable processing of geo-tagged data. Using this approach Gao et al. (2014) used Volunteered Geographic Information (VGI) sites and automatically linked the results with ArcGIS Desktop for visualization. The authors further constructed a gazetteer from the geospatial information collected from social media such as Flickr through geo-tagged images. Spatial Hadoop is an example of another extension of Hadoop to process spatial data (Eldawy, 2014).

SOLAP which is a powerful decision support system for exploring the multidimensional perspective of spatial data, was introduced by Li et al. (2014) for environmental monitoring which requires spatio-temporal analysis. The author used this system with the Hadoop MapReduce framework for processing large remote sensing data. Similarly, in another study, the feasibility of Hadoop MapReduce in a parallel model was explored with a broader domain of scientific data and installed in a cluster of high-end computers (Golpayegani and Halem, 2009). In their research the authors proposed that Hadoop may be used to improve scientific satellite data processing time.

All the Hadoop based big data platforms mentioned above can handle typical vector and raster datasets. However, there are very few big data platforms which can support moving objects such as trajectories. Parallel Secondo is one such system which is an open source moving object database developed by Lu and Guting (2014). According to the authors, Parallel Secondo is useful for handling big mobility data since it provides Secondo functionality in a distributed environment. Parallel Secondo uses Hadoop for parallel processing. Lu and Guting (2014) compared the processing efficiency of Secondo and Parallel Secondo. They performed different queries on OpenStreetMap data as well as GPS data of personal trajectories. In a similar study, Orakzai (2014) compared the time efficiency of Parallel Secondo to his own approach for processing big mobility data in a parallel environment using HBase and achieved better performance than Parallel Secondo. However, the system is only a prototype and is not available for general public to experiment.

The use of mobility data using SECONDO is not common due to the new technology. Not much work has been done in mobility data using SECONDO. There is a dire need to promote the effectiveness of mobility data and the information gathered after analyzing such data could be used in multiple fields.

Di Felice et al.,(2013) used SECONDO to analyze the trajectories of sex offenders in USA counties. They used example data of 20 subjects and 20 sensible areas. The trip of these subjects are then analyzed in SECONDO. This study is useful for managing crime and reducing it through proper law enforcement on the data analyzed with the help of the trajectories of these sex offenders.

There are several studies which used movement data sets such as T-drive. Yu et al. (2015) proposed a system called Cludoop and used T-drive with the Hadoop MapReduce based algorithm for clustering the large dataset. A comprehensive experimental evaluation on 10 network-connected commercial PC's was carried out using both huge-volume real and synthetic data. The authors demonstrated the effectiveness of the algorithm in finding correct clusters with arbitrary shape and the scalability of their proposed algorithm was better than state-of-the-art methods. Furthermore they compared the effectiveness of the algorithm against other density-based clustering algorithms.

Liu et al., (2011) proposed an effective method to detect outliers from the taxi trajectory dataset of Beijing, the same one we are going to use. They did not only formulated an algorithm to detect outliers but also tried to find the causal interaction between them which had not been tested before. Detecting outliers from a massive data of trajectories is an important task as these outliers could disrupt the analysis that we want to perform. When the data becomes massive the chances to effectively remove outliers visually becomes very low and requires excessive effort and time. Therefore an efficient system was built by the team to not only detect the outliers but also find the reason of its existence and production in the data. For this purpose they used road network of Beijing to partition the urban area according to road framework into different regions which created region graph that allows them to analyze the traffic flow among these regions and detect outliers. They proposed two algorithms: (1) STOTree which could uncover the relations between outliers and their causal interactions (2) frequent Subtree that could reveal recurrent abnormalities present in the data and furthermore propose inherent problems in the road networks. This makes the work an exemplary as always the first step of any analysis contains data pre-processing and removing every anomaly data contains. The segmentation of urban areas using road networks was further detailed analyzed by Yuan (2012) and proposed an image-processing based technique to achieve their goal of segmenting the urban areas into regions using road network. They used simple morphological operators like *dilation, thinning and connected* component labeling to efficiently segment the urban area. Furthermore they displayed a case study to show implementation of this segmentation and its application regarding trajectory analysis. For this purpose they used same Beijing taxi cabs dataset but of 3 months duration.

Another study uses the same data however in another aspect. That is to infer the status of the taxi, whether taxi is occupied or available for passengers or either it is parked somewhere. Zhu et al., (2011) used probabilistic decision true in order to get the job done in which they first used identified features so that local probabilistic classifiers could be trained. After training the classifiers they carried out Hidden Semi Markov Model (HSMM) for long term travel patterns. This results in an estimation of taxi status (occupied or not). However for parking regions they carried out simple algorithms working on the displacement values of taxi moreover some geographical landmarks like signals or parking areas to separate events like traffic jam or traffic signals. This method could allow city's transportation system and land use planning bodies to improve their assessment and services. Whereas another study benefits directly the taxi drivers in finding passengers and vice versa. J. Yuan et al., (2011) proposed a recommender system for people and taxi drivers which would allow them to find their ride or passenger in a nearby place and furthermore process the fastest route to reach their. They too used the probabilistic model in order to find suitable locations where taxi drivers could find passengers and passengers could find vacant taxi. For this purpose they used trajectories of 1200 taxis for duration of 110 days in Beijing.

For improving the navigation and routing system with the help of drivers expertise, J. Yuan et al., (2010), J. Yuan et al., (2011) and J. Yuan et al., (2013) work is recognizable. Using time-dependent landmark graph and Variance Entropy Based Clustering approach they suggested fast routes to reach destination. Their main focus was to use the expertise of the taxi drivers to help general public find fastest routes as local Taxi drivers of any area know best routes for a particular destination as it is their job. They used the same T-drive data for their analysis and proposal as we are going to work on. J. Yuan et al ., (2011) proposed a cloud based system computing for an end user. Through this technique anyone could utilize the services on a go and finds the fastest route which taxi drivers are utilizing in real time.

Lee et al., (2014) focused on applying spatial queries like containing, contained in, intersects and within Distance on a Hadoop based distributed system using H-Base for storage purpose. They used GeoLife GPS trajectory dataset and San Francisco taxi cab traces. After applying spatial queries they compared effectiveness of distributed system with single computer processing. GeoLife dataset was used again by Uddin et al., (2011) who focused on finding Regions Of Interest (ROI). In addition they used San Francisco

Taxi Cab data and Starkey, Oregon DeerElk Data. Conditions for region of interest were set as (1) a range of speed that an object maintains while in an ROI (2) a minimum duration of staying in an ROI area and (3) the density of objects in that area. They modified DBSCAN method for density base clustering. First they measured the minimum duration of stay point inside region of interest then identified point wise dense regions.

Yu et al.,(2013) deals with the uncertainty problem in the continuously changing data objects CCDOs due to the fact that databases can only be discretely updated. This problem was dealt in a Hadoop Map/Reduce environment using GeoLife GPS trajectory dataset. Whereas Mathew et al., (2012) proposed a hybrid method for human mobility on the basis of Hidden Markov Models (HMMs). The approach was to cluster location histories according to their properties, and after that they trains an HMM for each cluster. The HMMs is used to allow the users to account with location characteristics as unobservable parameters, and also to account with the effects of each individual's previous actions. They report through their experiment using GeoLife real-world location history dataset, that a prediction accuracy of 13.85% can be achieved when taking data from regions of roughly 1280 square meters. Yuan et al., (2010) used T-Drive dataset to mine smart driving directions from the historical GPS trajectories of a large number of taxis, and provides a user with the route which takes minimum time to a given destination. A Variance-Entropy-Based Clustering approach has been proposed to estimate the distribution of travel time between two landmarks in different timestamps. They designed a two-stage routing algorithm to compute the practically fastest route.

All the cited studies are very recent which shows that geospatial big data is an emerging discipline. The topic we selected has both commercial and research potential.

For processing of very large movement data, Parallel Secondo is a good choice, however there is a high learning curve due to its own Secondo executable language that requires certain amount of expertise to master all the operators. There are very few studies in literature which reports the real performance of Parallel Secondo that gives us the advantage to apply this system using real data therefore it can provide an efficient tool to handle big mobility data and perform certain analysis. Moreover the use of open source geospatial big data platforms such as Parallel Secondo is a good choice for running and executing realtime analysis.

Chapter 2

MATERIALS AND METHODS

2.1 Study Area/Dataset

There are several datasets of trajectories available online for free downloading for example:

- New York Cab trajectory dataset: This data set contains information about all the New York cab fares along with coordinates of locations from where taxi picked a passenger and location of dropping point. (http://www.andresmh.com/nyctaxitrips/)
- Movebank: A free online infrastructure initiated to help researchers manage, share, analyze, and archive animal movement data, which has been hosted by the Max Planck Institute for Ornithology. The database of Movebank has been designed primarily for data sets that include consecutive locations of individual animals, which are generally called tracking data. The status of Movebank, at the end of 2013, is as follows: 1,000 user-created studies, 350 taxa, and more than 61 million unique animal locations. (http://www.movebank.org/)
- Community Resource for Archiving Wireless Data at Dartmouth (CRAWDAD) holds wireless network data for research community to utilize this free data and develop new and efficient tools to extract information that could be useful commercially or from research point of view. (http://www.crawdad.org/)

- Some of the data is available on SECONDO official website in the format which is supported by the software (http://dna.fernuni-hagen.de/secondo/). Table 2.1 contains the list of all the datasets.

Table 2.1: Open Source datasets available from Secondo Home page.

Data Set	**Description**
berlindb	The city map of Berlin
geodb	Zip codes of German cities
metrodb	Some trains of the Berlin metro represented as moving points
opengeodb	German states and districts. For further information visit www.opengeodb.de
optdb	Some relations used by the examples in the optimizer documentation
osnabrueck	City map of the German city Osnabrück
transobj	A java program converting data from www.dict.cc
vorwahlobj	Area codes of German cities

To perform spatial pattern analysis and exploring big data solutions we selected T-Drive taxi trajectory dataset. T-drive dataset was created by Microsoft Research Asia. The average sampling interval is about 177 seconds with a distance of about 623 meters. The data used in this research are taxi tracks of 7 days only. The size of T-drive dataset is 1.6 GB. This does not qualify for big data however the possibility to scale up the analysis provides an insight into the integrity of the system. This dataset contains the GPS trajectories of 10,357 taxis within Beijing, China. Dataset contain trajectories during the period of Feb. 2 to Feb. 8, 2008. Each file of this dataset, which is named by the taxi ID, contains the trajectories of one taxi. Full details and properties of dataset is presented in Table 2.2.

However the data is not big enough to make much of a difference in query processing as this data is of 7 days duration only, however the possibility to scale up the analysis and results could provide a better opportunity to understand the integrity of the system we are going to propose and demonstration could be done accordingly. This pilot phase could enable commercial organizations to attain valuable information by utilizing these Big Data techniques. This data presents real time challenges of handling big mobility data including outliers anomaly, trajectory discontinuity and map matching.

Table 2.2: Characteristics of T-Drive dataset.

Properties	**T-Drive**
Time Span of the Collection	02/02/2008 - 08/02/2008
Number of Users	10,537
Number of trajectories	10,359
Number of points	15 million
Total distance	9 million km
Total duration	144 hour

2.2 Hardware/Computer System

For computation analysis and performance measurement two different types of hardware were required both with powerful computation abilities to be able to handle such big data and contain multiple processors in order to process tasks in parallel. The system we used for single node analysis has following specifications:

- Processor Intel® Core™ i7-4790 CPU @ 3.60GHz × 8
- Random Access Memory (RAM) 8GB
- OS Ubuntu 14.04 LTS 64 bit

For multi node analysis a powerful system with specifications mentioned below was used:

- PowerEdge R720/R720xd Motherboard TPM
- 2x Intel Xeon E5-2680v2 2.8GHz, 25M Cache, 8.0GT/s QPI, Turbo, HT, 10C, 115W, Max Mem 1866MHz
- 256GB (16x 16GB) RDIMM, 1600Mhz, Low Volt, Dual Rank, x4 Bandwidth

To continue with distributed system a virtual machine of Linux Ubuntu 14.04 64 bit was deployed on Institute of Geographic Information System and Remote Sensing (IGIS), National University of Sciences and Technology, Islamabad machine.

A powerful virtual machine was created with 20 core processors and assigned 200 GB RAM so that it could contain multi nodes and be useful to test the ability of Parallel Secondo. Having 20 processors at hand it was optimal to deploy no more than 10 nodes of Parallel Secondo system on the machine so that each node could have at least 2 processors at their disposal all the time. After deploying the system with all ten nodes cluster, a simple query was processed to test the efficiency of the system. Figure 2.1 illustrates a flow chart of methodology.

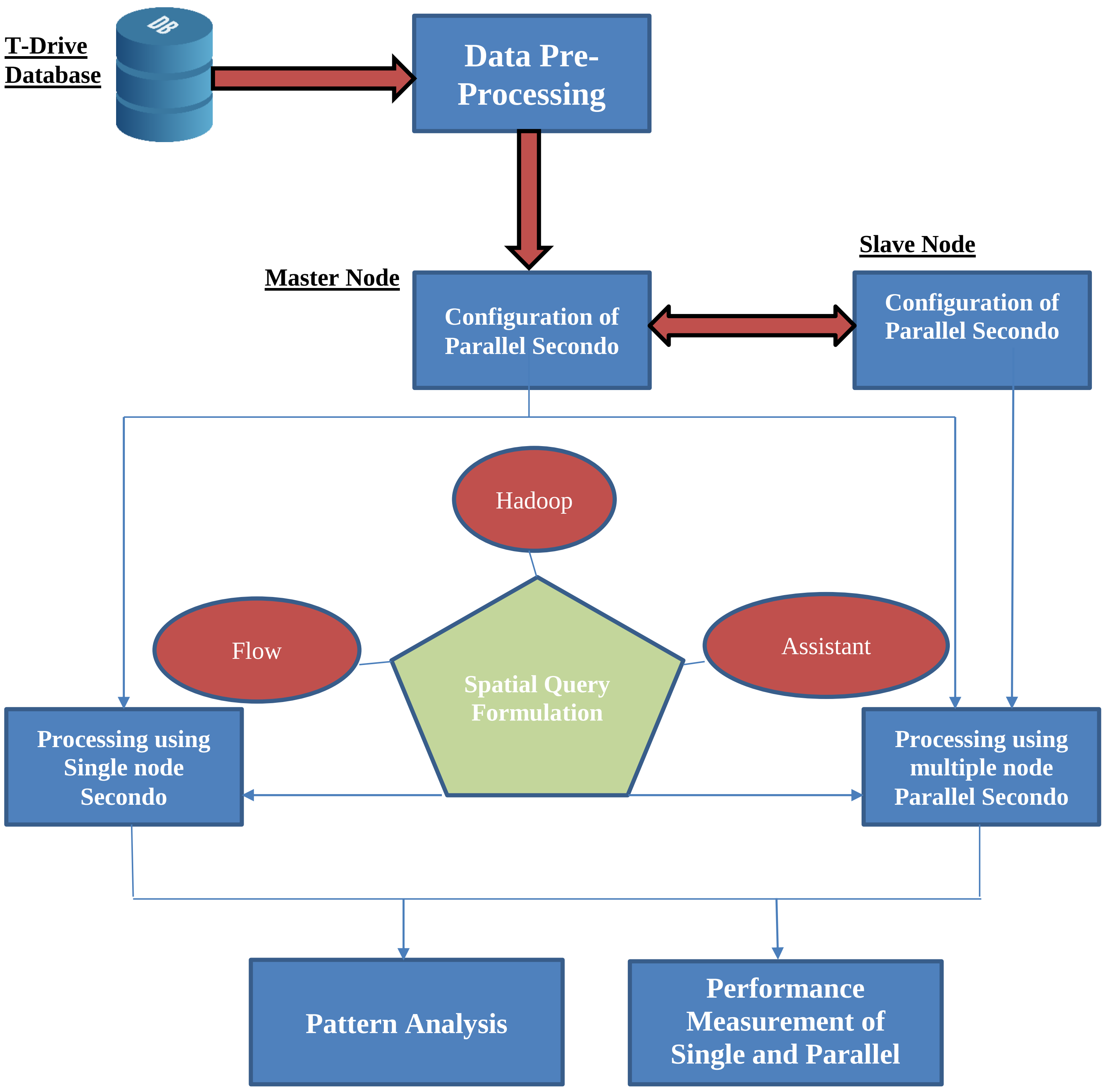

T-Drive Database
DB
Data Pre-Processing
Master Node
Configuration of Parallel Secondo
Slave Node
Configuration of Parallel Secondo
Hadoop
Flow
Spatial Query Formulation
Assistant
Processing using Single node Secondo
Processing using multiple node Parallel Secondo
Pattern Analysis
Performance Measurement of Single and Parallel

Figure 2.1: General Flow Chart of Methodology.

2.3 Pre-Processing

The data of taxi trajectory is available in a raw form which means it contains several anomalies due to GPS error or system unavailability hence this data first needs to be pre-processed and required to remove all the uncertainties and error points also known as outliers. Figure 2.2 shows sample trajectories of T-Drive dataset in raw form containing several outliers.

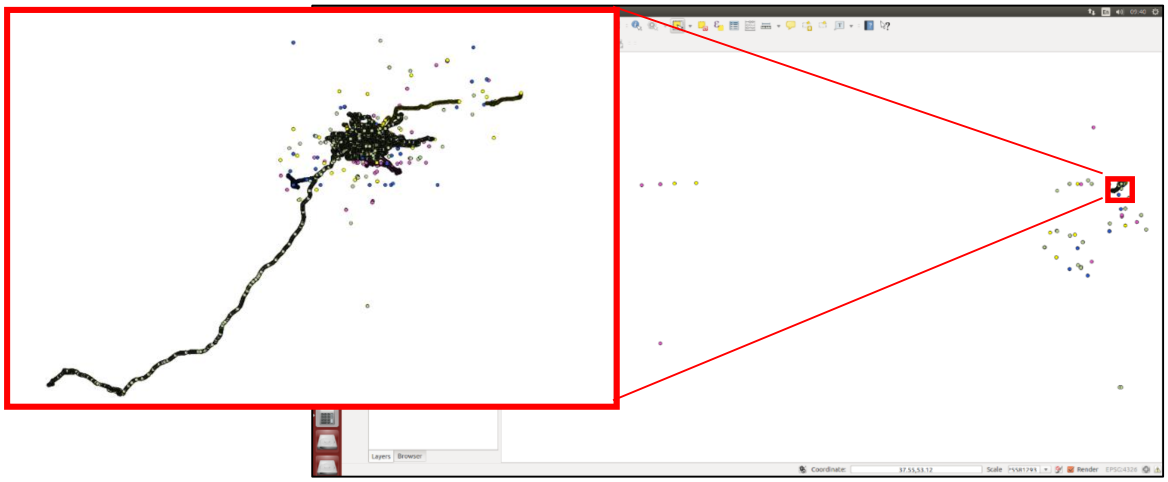


Figure 2.2: Outliers in the data.

Due to the massive amount of points in the dataset, visualization of whole data at the same time requires lot of computing power and visually removing all the outliers becomes inefficient requiring much of the man power as well. Furthermore due to the dense cloud of points in center of the city makes it impossible to remove outlier lying inside city being removed and clean the data anomalies. Hence an efficient system is required to process the data and maintain its integrity.

An automated system is required which could get rid of the outliers logically and requires less human interaction. Furthermore the data needed to be arranged in a format that could be easily read by the Secondo system without causing too much effort to process. For this purpose a script was designed in python language to do the required task.

2.3.1 Python Script for Outlier Removal

Python, an interactive high level programming language is widely used and popular among programmers with its wide variety of libraries and functions that gives freedom to develop variety of application using fewer lines of codes due to its relatively easy syntax compared to its competitors such as C++, Java etc. Due to the availability of python interpreters on almost every version of OS, programmers are independent to choose their platform and implement the code accordingly.

For the development of script that enables us to achieve our goal of removing outliers several libraries were required like *RE*, *OS*, *datetime*, *subprocess*, *math*, *glob* and *numpy*. Most of these libraries were available as default in Linux Ubuntu 14.04.1 LTS OS. First task for the script was to collect all the files containing trajectory datasets from the directory and read them one by one in which glob library was useful for gathering all files with extension of .txt as there were 10,356 files present representing each unique taxi trajectory.

The logic used for detection and removal of outliers was distance measurement. As we know there are certain speed limits inside city and its outskirts. These speed limits varies according to the type of the roads for example, for single lane road 30 Km/h (19 Mp/h), 70 Km/h (43 Mp/h) for multi-lane roads whereas National highways could go up to

80Km/h (50 Mp/h) and the highest achievable speed could be attained on expressway of 120 Km/h (75 Mp/h). Hence taking account of some speedsters the threshold limit was set as 150 Km/h (93 Mp/h). The threshold speed will thus decide the acceptable distance between the points.

The Python script converts Latitude and Longitude coordinates of points and the distance achieved through computing consecutive points to linear distance according to the latitude value of Beijing and its respective 1 mile distance.

The data consist of unique Taxi ID, its coordinates in latitude longitude and corresponding date time format. Secondo supports particular type of date and time format to be read as instance hence the script has to be able to change the format accordingly. Although Secondo gives functions to change the format it will require double computation power. Hence it was better to change the format on the run through python script that will save time. Putting all the clean data in one file as .csv format would be more efficient. The script could automatically upload the data in Secondo database by creating .sec file containing some useful commands to import data in Secondo environment. This was done using OS and subprocess library that enable us to access terminal of the system from where we could start Secondo text interface and complete our task without any human interaction. Figure 2.3 shows the flow diagram of processes script followed whereas Figure 2.4 displays the output of the script after removing outliers.

2.4 Parallel Secondo

After pre-processing, T-Drive queries performance is evaluated on both single and parallel nodes of SECONDO. Some spatial queries are applied to find patterns. Parallel

SECONDO is a hybrid system and in order to measure the capability of handling extensible data model in SECONDO to a group of computers, it's built up by powerfully coupling the Hadoop framework with a set of SECONDO databases.

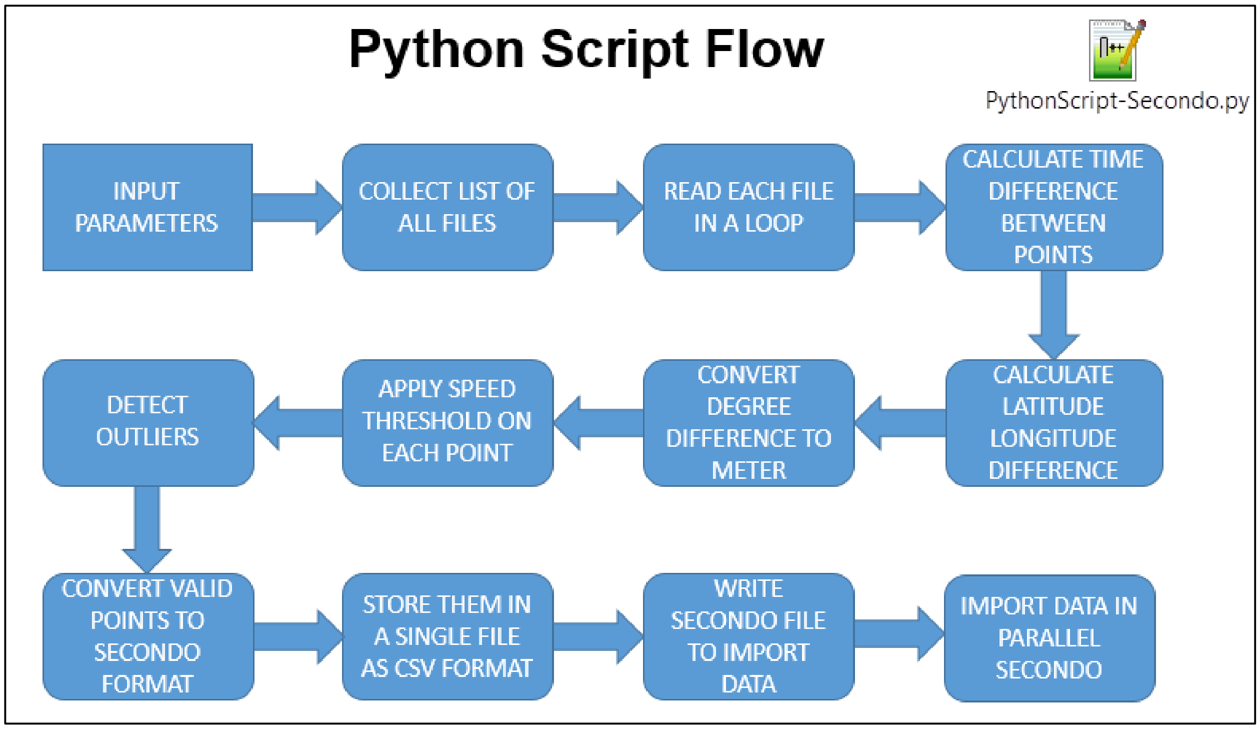

Python Script Flow
PythonScript-Secondo.py
INPUT PARAMETERS
COLLECT LIST OF ALL FILES
READ EACH FILE IN A LOOP
CALCULATE TIME DIFFERENCE BETWEEN POINTS
CALCULATE LATITUDE LONGITUDE DIFFERENCE
CONVERT DEGREE DIFFERENCE TO METER
APPLY SPEED THRESHOLD ON EACH POINT
DETECT OUTLIERS
CONVERT VALID POINTS TO SECONDO FORMAT
STORE THEM IN A SINGLE FILE AS CSV FORMAT
WRITE SECONDO FILE TO IMPORT DATA
IMPORT DATA IN PARALLEL SECONDO

Figure 2.3: Flow Diagram of Python Script for Outlier Removal.

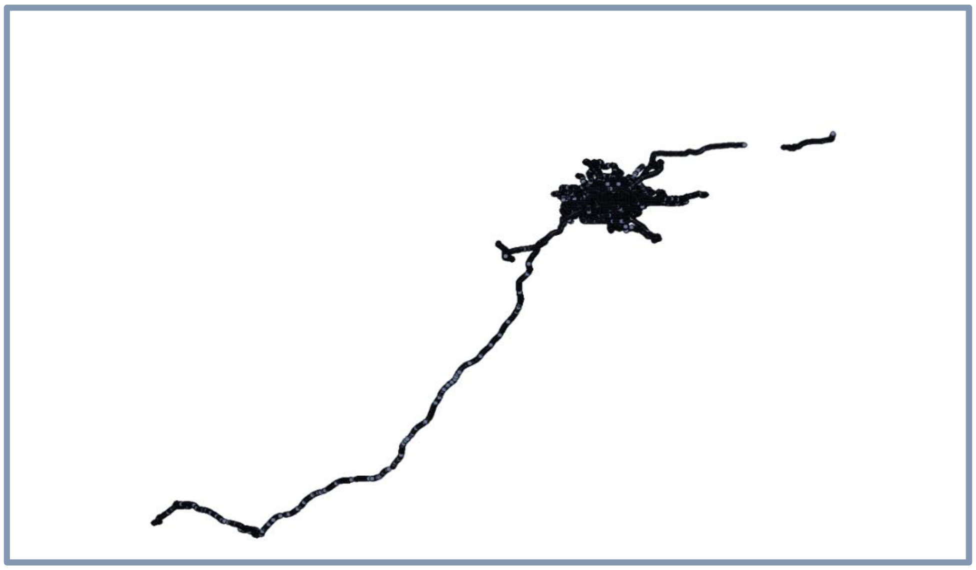


Figure 2.4: Point Result after removing outliers from the trajectories using python script.

First of all Parallel SECONDO will set up of several processing units on each cluster node and even low end computer could be used as nodes. Afterward Parallel SECONDO will set DS as its basic processing unit. Each DS containing a compact SECONDO system called Mini-SECONDO and its affiliated database and each DS will be deployed on one hard disk. For that reason, in computers with numerous hard disks, we can set several DSs on the same computer in order to reduce the disk interference to the minimum. Similarly the simultaneous tasks containing different Mini SECONDO databases can read their data individually. Using several DSs set, we specify one of them as the Main Server (MS) that containing the configuration data and management scripts that work for the rest of them. Moreover it consists of a Hadoop node being used to receive the assigned tasks from the Hadoop framework.

Secondly based on the Hadoop division Parallel SECONDO will separate the DSs into different roles. SECONDO will show the MS on the Hadoop master as its mDS, while the left over DSs will be observed as sDSs. The task of these DSs will be to retain in the DS Catalog and share it with every cluster node. Therefore, we will set the Mini-SECONDO database in mDS as the master database of the entire system, while the others will be slave databases. The master database will cover worldwide data like the cluster's size, global index structures and the metadata of distributed objects while the slave databases will cover only the local objects belonging to the corresponding sDSs and its being hidden from users. This gives users the impression of still handling a separate SECONDO database.

Thirdly Parallel SECONDO will be fully compatible with the current SECONDO system and both will be combined seamlessly. In principle, the master database can still be used as an ordinary SECONDO system, where all consecutive queries can be handled as usual.

Fourth, in Parallel SECONDO, parallel queries will be processed strictly on the MapReduce standard base. These queries will be converted into Hadoop jobs by the master database, then will additionally be divided into Map and Reduce tasks by Hadoop. Parallel tasks are then processed by all sDSs following MapReduce Model in order to attain balanced amount of work. Figure 2.5 shows flow of the structure of Parallel Secondo file system and how it processes the data.

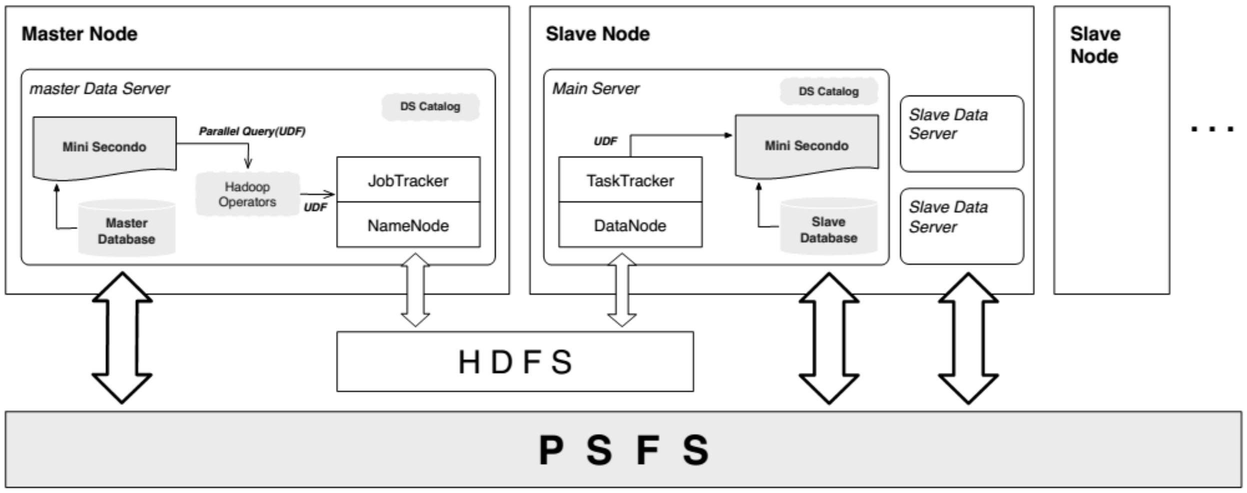


Figure 2.5: Architecture of parallel secondo (Jiamin & Guting, 2014).

During the installation of Parallel SECONDO, Hadoop will be installed by unpacking the software and setting HDFS nodes to all MSs. According to Parallel SECONDO preferences, its configuration parameters will furthermore set automatically. No extension will be made in Hadoop core functionality and without the participation of any Parallel SECONDO components the framework will work all by its own mechanism.

Furthermore we will perform the same operations on simple SECONDO to compare the effectiveness of the system.

2.4.1 Parallel Secondo Configuration

As Parallel Secondo is a hybrid system based on Secondo and Hadoop platform, it requires first the installation and configuration of Secondo on every node whether Master or Slave. Secondo being an open source provides users with its source code that could be deployed on users system and requires some steps to configure. The installation on Linux and Macintosh operating systems is much easier as compared to Windows system. Moreover virtual disk images deployed on Linux system are also available online for users to quickly test the performance of the system without having to deploy on their own system. Likewise Parallel Secondo also offer users with virtual disk images having Linux Operating system to explore the abilities and varieties of functions system has to deliver.

However users need to follow guidelines according to their operating systems provided by the developing team in order to configure their own system. As our system resides with the Linux Ubuntu 14.04.1 LTS Operating system, hence first step was to update the system and install some additional software or libraries mentioned in the Table 2.3.

An online bash script is provided for the users to easily deploy all these additional software and also make a configuration file accordingly. However users could install all these libraries one by one on their own. If they don’t trust the script or have fear that script might alter their systems internal files which could cause problems for other software’s. At the end of the script a configuration file with the name .secondorc is created at the root of the system. Users need to link this file with the systems environment by adding following line in the file *.bashrc.*

```
        source $HOME/.secondorc $HOME/secondo
```

Next step is to download the source code of Secondo available at http://dna.fernuni-hagen.de/secondo/ for free. Deployment of the software is made simply by unzipping the software at $HOME location and making the software by typing *make* in terminal accessing secondo folder. This will install Secondo on your system along with all its algebras and libraries. These steps should be repeated on every system user intended to include in the cluster.

The next part is to make Secondo available for distributed processing or making Parallel Secondo. There are two more utilities required in case of Parallel Secondo and that includes *ssh* and *screen* which are the main source of communicating and accessing different systems in the clusters. This could be arranged using following commands in terminal:

```
        $ sudo apt-get install ssh
        $ sudo apt-get install screen
```

Another requirement is to enable *ssh* passphraseless hence to access each system without having to enter password every time any function wants to access other nodes on the slave nodes. This could be achieved by executing following commands:

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

This will generate a key for every IP of systems in the cluster. Having done with ssh user need to download Hadoop source of specific version at the below mentioned address http://archive.apache.org/dist/hadoop/core/hadoop-0.20.2/hadoop-0.20.2.tar.gz . The tar file of Hadoop downloaded should be placed in secondo/bin directory for Secondo algebra to easily access and install according to its environment.

Next step is to prepare configuration file of Parallel Secondo for the cluster or for single node itself. An example configuration file is provided in the source code of Secondo inside secondo/Algebras/Hadoop/cluster Management/ with the name Parallel Secondo Config.ini. We have to give the address of Java directory as it will be needed for Hadoop installation and processes. Then we have to declare the IP addresses of the systems network. One system is made the Master node and all the other systems as Slave nodes. It is important to give “1234” as default port when configuring SECONDO. The example of the configuration changes user have to make is as follows:

```
Master = 192.168.1.1:/disk1/dataServer1:11234
Slaves += 192.168.1.1:/disk1/dataServer1:11234
Slaves += 192.168.1.2:/disk1/dataServer1:11234
Slaves += 192.168.1.1:/disk2/dataServer2:14321
Slaves += 192.168.1.2:/disk2/dataServer2:14321
```

Above mentioned configuration represents two systems on which four different slave nodes were deployed. The first system was furthermore assigned Master node that will assign all the Hadoop jobs to each node respectively. This configuration will create two dataservers on each system. Furthermore when making cluster containing more than one system, user needs to give IP of Master node to Hadoop operators as mentioned in an example.

```
        core-site.xml:fs.default.name = hdfs://192.168.1.1:49000
        hdfs-site.xml:dfs.http.address = 192.168.1.1:50070
        hdfs-site.xml:dfs.secondary.http.address = 192.168.1.1:50090
        mapred-site.xml:mapred.job.tracker = 192.168.1.1:49001
        mapred-site.xml:mapred.job.tracker.http.address =
192.168.1.1:50030
```

For users wishing to use simple Secondo side by side of Parallel Secondo and that the databases could be shared, last line should be activated which allows user to access databases residing in simple Secondo. This configuration file should then be copied to $HOME/secondo/bin/ directory.

```
        NS4Master = true
```

Additionally we have to alter one line of SecondoConfig.ini at $HOME/secondo/bin/ location to turn off the transaction feature which would improve the data exchange efficiency between Data Servers (DS), by uncommenting the following line:

```
        RTFlags += SMI:NoTransactions
```

At the end *makefile.algebras* should furthermore be altered by adding following lines that enables Secondo to have abilities of Hadoop Algebras.

```
        ALGEBRA_DIRS += Hadoop
        ALGEBRAS += HadoopAlgebra

        ALGEBRA_DIRS += HadoopParallel
        ALGEBRAS += HadoopParallelAlgebra
```

After making all the changes access through terminal the directory

```
$HOME/secondo/Algebras/Hadoop/clusterManagement/
```

Here enter the command that will deploy all the functions of Parallel Secondo on each node:

```
$ ps-cluster-format
```

As we had changed algebras file hence there is a need to reboot Secondo by again running make command through terminal. This would made all the necessary files for running Hadoop jobs and algebras. This modified Secondo is now in need to deploy mini-Secondo on every node in the cluster which could be achieved by:

```
$ cd $SECONDO_BUILD_DIR/Algebras/Hadoop/clusterManagement
$ ps-secondo-buildMini –co
```

For users wishing to install on single node should replace –co with –lo where c represents cluster and l represents localhost. Now Parallel Secondo system is ready to be used.

Initially all nodes have to be started. After that all the monitors of each Parallel Secondo node should be activated that could be achieved by using following commands in terminal:

```
$ start-all.sh
$ cd $SECONDO_BUILD_DIR/Algebras/Hadoop/clusterManagement
$ ps-start-AllMonitors
$ ps-cluster-queryMonitorStatus
$ ps-startTTYCS –s 1
```

The last command will open the text interface of Parallel Secondo where user can create databases and use the maximum abilities of the Parallel Secondo system. GUI could furthermore be used for querying and display purposes by running sgui file through terminal in $HOME/secondo/Javagui/ directory. User will need to give the master node IP and port. The general flow chart of Parallel Secondo configuration is displayed in Figure 2.6.

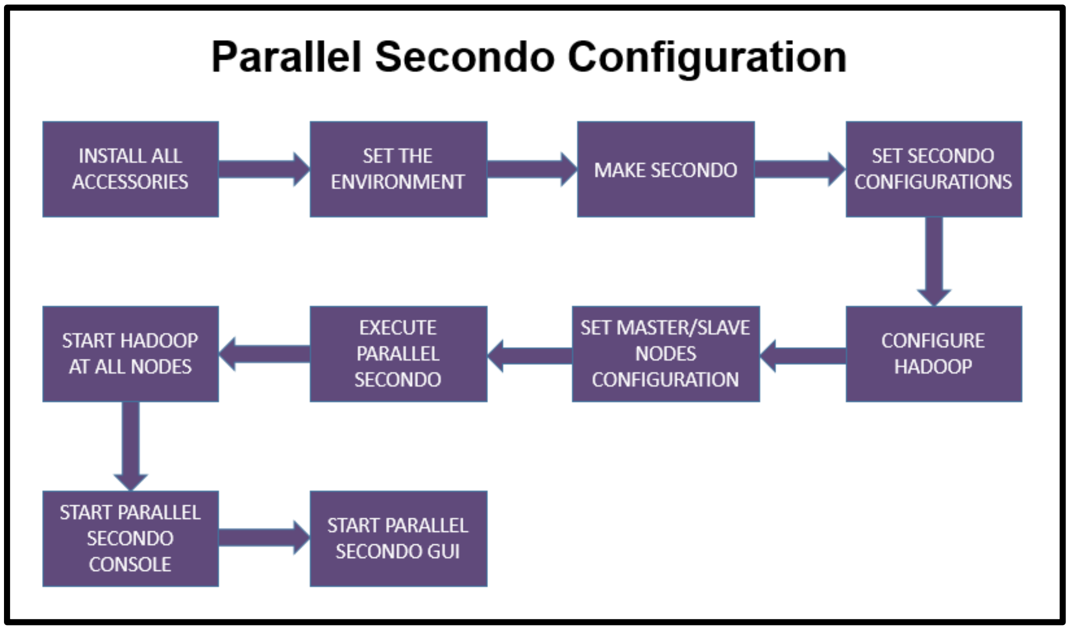

Parallel Secondo Configuration
INSTALL ALL ACCESSORIES
SET THE ENVIRONMENT
MAKE SECONDO
SET SECONDO CONFIGURATIONS
CONFIGURE HADOOP
SET MASTER/SLAVE NODES CONFIGURATION
EXECUTE PARALLEL SECONDO
START HADOOP AT ALL NODES
START PARALLEL SECONDO CONSOLE
START PARALLEL SECONDO GUI

Figure 2.6: Flow diagram of Configuration steps for Parallel SECONDO.

Table 2.3 Additional Utilities for Parallel SECONDO.

Software package	**Version**	**Project home page**
Berkeley DB	>4.3.29	Oracle Berkeley DB
SWI-Prolog	5.6.37	http://www.swi-prolog.org
GCC	3.4 - 4.1	http://gcc.gnu.org
Java SDK	1.7	http://www.oracle.com/technetwork/java/index
Bison	2.1	http://www.gnu.org/software/bison/bison.html
Flex	2.5.33	http://www.gnu.org/software/flex/flex.html

Make	3.79.1, 3.80	http://www.gnu.org/software/make/make.html
BASH	2.x	http://www.gnu.org/software/bash/bash.html
GSL	1.12	http://www.gnu.org/software/gsl/
Libjpeg	6.2	http://www.ijg.org/
libreadline	5.2	http://tiswww.case.edu/php/chet/readline/rlto
Hadoop	0.20.2	http://hadoop.apache.org/

2.5 Spatial Query Formulation

Formulating spatial queries to extract useful information from the given data is an important procedure in our research. As we are dealing with moving data or trajectories hence we would need spatio-temporal queries that could analyze patterns in moving objects datasets. We would explore some of these special queries like flock, swarm, convoy, leadership and motion etc. Furthermore we will find some hidden patterns inside the data that gives us advantage in understanding and creating a new direction system. We would check its implementation in Parallel SECONDO and analyze patterns in the taxi dataset we are going to use. These spatial queries are defined by Li, (2014) is as follows:

- Flock: A set of moving objects form a flock if every object is contained in a disc with specified radius and timestamp.
- Convoy: A set of moving objects form a convoy if every object belongs to the same density-connected cluster at a particular timestamp.
- Swarm: A set of moving objects form a swarm if every object are in the same density-connected cluster where timestamps could not be necessarily consecutive.
- Leadership: Leadership pattern identifies a leader among all the moving objects which is directing a flock, convoy or swarm.

- Convergence: A set of moving objects passing through the same circular region of radius at any timestamp.

As discussed earlier Parallel Secondo supports same language as that of Secondo however with some additional operators and algebras that allows query to process in parallel environment along with the integration of Secondo conventional environment. Table 2.4 shows the list of some of the additional operators for parallel Secondo:

Table 2.4 Additional Operators for Parallel Processing.

Kind	Name	Signature
Flow	spread	stream(T) → flist(T)
	collect	flist(T) → stream(T)
Hadoop	hadoopMap	flist x fun(Map) x bool → flist
	hadoopReduce	flist x fun(Reduce) → flist
	hadoopReduce2	flist x flist x fun(Reduce) → flist
Assistant	para	flist(T) → T

Flow operators are responsible for distributing and collecting data from nodes whereas Hadoop operators process the queries in parallel environment according to Map-Reduce paradigm whereas Assistant operators are used for those functions which are not yet supported by Hadoop operators hence they convert the data back to simple form on runtime.

2.5.1 Simple Parallel Queries

We will start with some simple queries and some necessary steps needed to create a database containing trajectories dataset. After opening the text interface or GUI of Parallel Secondo first step requires creating and opening a new database which was accomplished using following commands:

```
creating database TDrive;
open database TDrive;
```

Defining a schema according to our dataset attributes and their types is the next step which would allow us to import data according to the schema we have defined:

```
let TDPSchema = [const rel(tuple([
  TaxiID: string,
  UTC: string,
  Date: string,
  Speed: real,
  Longitude: real,
  Latitude: real
]))
value()];
```

Here we have created an empty tuple relation with all the attributes our dataset contains. There has been an additional attribute of speed which we had calculated through python script so that it could be useful in further analysis. Importing the data directly from CSV file into the empty relational tuple we had just created was achieved by executing following command:

```
      let Raw = TDPSchema
csvimport['../bin/CompleteOutliersRemoved.csv', 1, "#", ","] consume;
```

Here first parameter of *csvimport* function as could be noticed is the address of the file we want to import data from whereas next parameter asks for number of line user want to skip that contains description about the data or the labels of the attributes. In our case its only one row that contained the labels of the attributes. If user is not sure of how many rows to skip or some comments are present inside the data which should not become the part of the data could moreover be skipped by declaring the symbol in third parameter with which every comment starts. Last parameter asks for the separator symbol between attributes. Some constant declaration for parallel processing would be useful for user like defining the size of clusters (number of slave nodes or DS) and furthermore the number of parallel processes user wishes to make for the system.

```
let CLUSTER_SIZE = 1;
let PS_SCALE = 8;
```

As we were using at that time only single node system containing one master and one slave node. Whereas PS_SCALE value was defined 8 as there were 8 processors present in the system we were using hence it would be appropriate to run 8 parallel processes or tasks.

Till now the data has been successfully imported into the system in a raw form hence next step is to generate points according to the attributes Latitude and Longitude so that we could plot them if we like or perform analysis on point data. The conventional way of doing therefore in simple Secondo is as follows:

```
let Points = Raw feed extend[ I: str2instant(.Date + "-" + .UTC),
P: makepoint(.Longitude, .Latitude)]
consume
```

Here the feed operator will produce the stream of values from Raw tuple so that each value could be read one at a time, the extend operator will add a new attribute in the table for new values to be stored without disturbing the other values whereas str2instant operator is converting date and time value to instant according to Secondo format and at last make point is responsible for generating points in the database. However we could run this same query in parallel however first we have to distribute the data to each slave node or DS according to a unique ID in our case TaxiID. Spread operator is responsible for such operation:

```
        let  taxi_id_dlf_raw  =  Raw  feed  spread[;TaxiID,  CLUSTER_SIZE,
TRUE;];
```

Here the data has been spread into 8x1 matrix with the Distributed Local File (DLF) flist. Two file types are supported in parallel (1) DLF (2) DLO (Distributed Local Object). DLF will divide the data into R x C matrix where R represents rows and C represents columns of the data whereas DLO will divide the data into N X 1 matrix where N is the total number of nodes in the cluster. By giving TRUE as the third parameter files will be divided as DLF otherwise FALSE will make it DLO flist. Now we can execute parallel query according to our requirement:

```
        let PointsHMap = taxi_id_dlf_raw
        hadoopMap[DLF, TRUE
        ; . extend[ I: str2instant(.Date + "-" + .UTC),
        P: makepoint(.Longitude, .Latitude)] ]
        collect[] consume;
```

Here basic operator hadoopMap will let the slave nodes process the data as a Map operation and at the end collect operator will gather the results as a simple tuple. Same function could be done using hadoopReduce operator which will process Reduce task of the Map-Reduce paradigm.

```
let PointsHRed = taxi_id_dlf_raw
hadoopReduce[TaxiID, DLF, PS_SCALE
; . extend[ I: str2instant(.Date + "-" + .UTC),
P: makepoint(.Longitude, .Latitude)] ]
collect[] consume;
```

Unfortunately there is no operator yet defined by Secondo that could provide the both functionalities of Map and Reduce task as one operation however users could use hadoopMap and hadoopReduce operators in same query where hadoopMap will just perform Map operation without processing the internal function and hadoopReduce will carry the rest of the process as reduce task.

Having coordinates accurately converted to points, next we could make trajectories of each taxi according to its unique ID. The conventional sequential way of executing query would be:

```
let taxiTrajectory = Points feed sortby[TaxiID asc, I asc]
groupby[TaxiID; Traj: group feed approximate[I, P] ]
consume;
```

Where first data is sorted according to TaxiID and the Instant in ascending order so that no discrepancy should occur and then groupby operator will collect all the points of same TaxiID in another attribute Traj according to approximate operator which will compute linear approximation between sampling points and Instants. The same query in parallel as hadoopMap and hadoopReduce could be displayed as:

```
        let    taxi_id_dlf_PHMap    =    PointsHMap    feed    spread[;TaxiID,
CLUSTER_SIZE, TRUE;];

        let taxitrajectoryHMap1 = taxi_id_dlf_PHMap
        hadoopMap[DLF, TRUE
        ; . sortby[TaxiID asc, I asc]
        groupby[TaxiID; Traj: group feed approximate[I, P] ]
        ]
        collect[] consume;


        let taxitrajectoryHRed = taxi_id_dlf_PHMap
        hadoopReduce[TaxiID, DLF, PS_SCALE
        ; . sortby[TaxiID asc, I asc]
        groupby[TaxiID; Traj: group feed approximate[I, P] ]
        ]
        collect[] consume;
```

These are some of the simple queries to begin with the analysis of the data. As we go further to analyze the data, complexity of the queries increase.

Chapter 3

RESULTS AND DISCUSSION

3.1 Computation Efficiency

The first task according to our objectives is to find the efficiency of our proposed system. For this purpose we have to process a query on single node system as well as multi node system comparing efficiency with the sequential query that is of simple Secondo without processing it in parallel using Hadoop operators.

To query the T-drive, the data was divided into four parts (each consisting of 1, 5, 10 and 15 million points respectively). Each part is run with multiple nodes to observe the optimal performance. Parallel Secondo was configured on a single node as well as on multiple nodes. Spatio-temporal queries were selected based on filtering, clustering, search and creating trajectories from points. These queries are basic steps often required when analyzing mobility data. All spatio-temporal queries were run with 1, 5, 10 and 15 million points.

3.1.1 Single Node Parallel Secondo

Deploying Parallel Secondo on the single computer with one Master node and one Slave node allow us to compute the efficiency. For example, is it worth to spend extra money for different hardware and still could get the best out of single computer by utilizing the maximum computation power that the system can get.

As our data contained about 15 million points of trajectory data, we divided this data in 4 parts by limiting the number of points in order to get an idea of when will this system be better efficient and for how much data. In 4 parts we process data with 1, 5, 10 and 15 million points separately. Figure 3.1 illustrates the minimum amount of time required to process creating trajectories query in Parallel Secondo distributed environment. The x-axis shows number of data points while y-axis plots the time duration. The graph demonstrates the performance of a single node configured with Parallel Secondo. It is argued that parallel queries are efficient only for big data and for small data conventional sequential queries should be used.

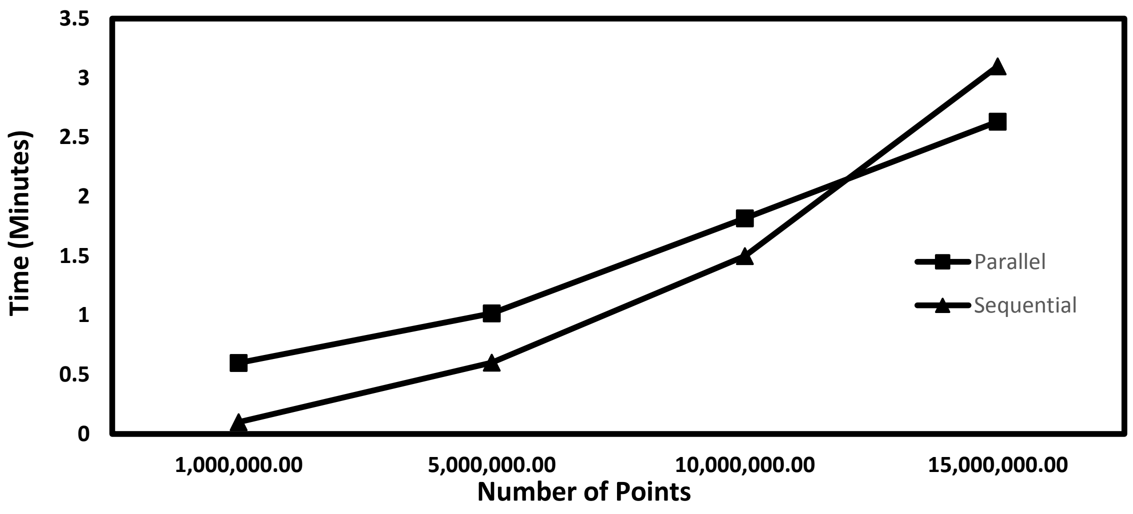


Figure 3.1: Graph showing computation efficiency of single node system.

It can be seen from Figure 3.1, that for less number of points the system is not useful as the data first need to be distributed and then it have to be again collected from the dataserver that makes the processing time more than that of the sequential query. However as we started reaching the maximum number of points (15 Million) that we had, parallel system efficiency started improving and the processing time decreased that shows us that this system is useful even on single node if we have Big Data because this system lets system use its full capacity of computation.

However the difference is still not much significant yet on 15 million points. If we draw the trend line of the graph using polynomial order 2 which fits the trend of the graph shown in above Figure, we can see a clear improvement in the efficiency of the system with having much bigger data to compute. This shows that this system is cost efficient too. It could enable user to utilize maximum computation power to get the best results in less time for processing Big Data.

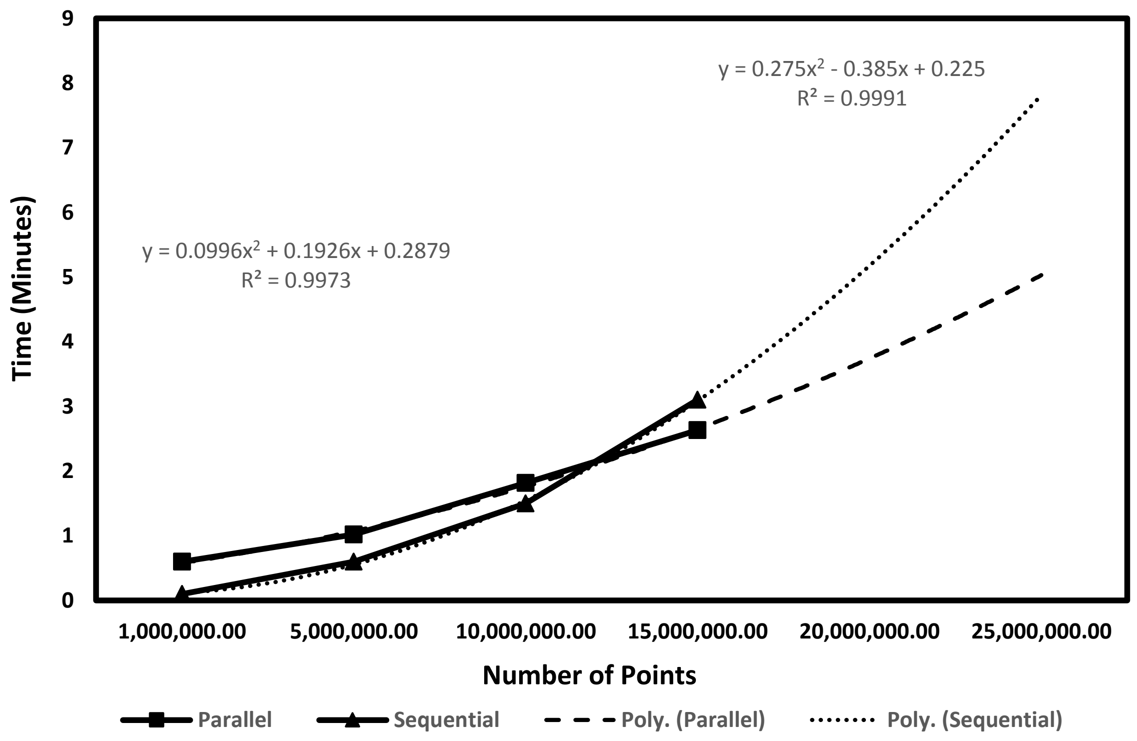


Figure 3.2: Graph showing computation efficiency of single node system and its trendlines.

3.1.2 Multi Node Parallel Secondo

The same procedure was implemented on a cluster of parallel node secondo system by applying same query but with varying cluster size to get the efficiency graph of the system and the ability of parallel secondo to withstand big data handling technique.

Here again data was processed in four parts with increasing points of 1, 5, 10 and 15 million. The same query was processed by activating different number of nodes at a time in order to find pattern in computation sequence and judge the optimal number of nodes required to process how much data. Graphs show different results achieved.

The graphs shown in Figure 3.3, 3.4, 3.5, 3.6 and 3.7 represent the variation in efficiency according to the size of the data. It is not ideal approach to increase the number of nodes indefinitely because it will add more time to distribute and collect the data from nodes then to process. Therefore it will decrease the efficiency as demonstrated in the graphs.

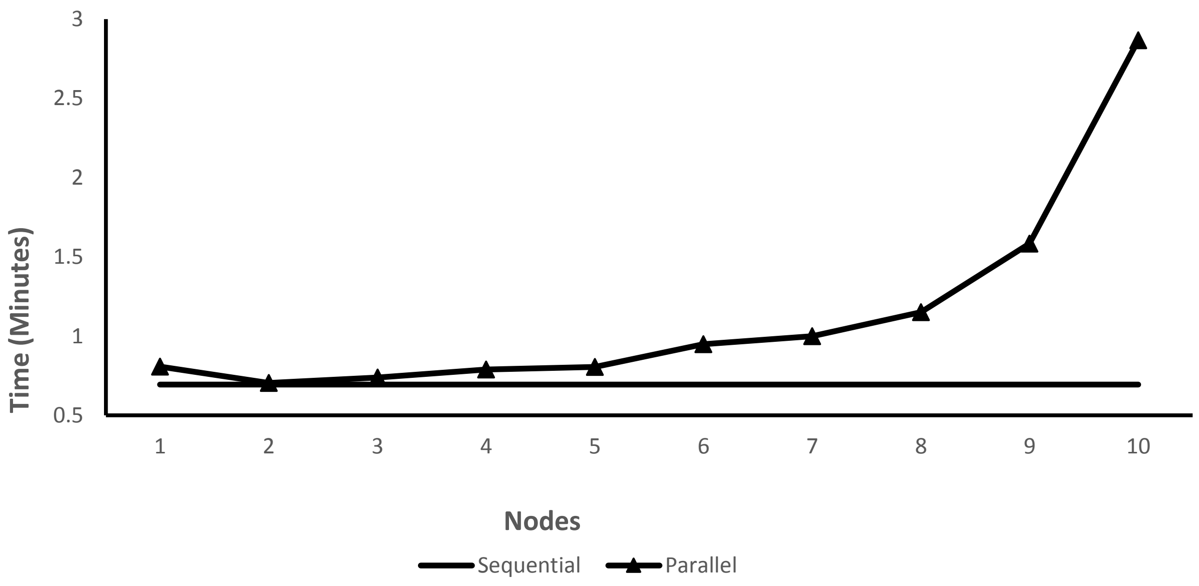


Figure 3.3: Computation Efficiency of Multiple Node System (1 Million Points).

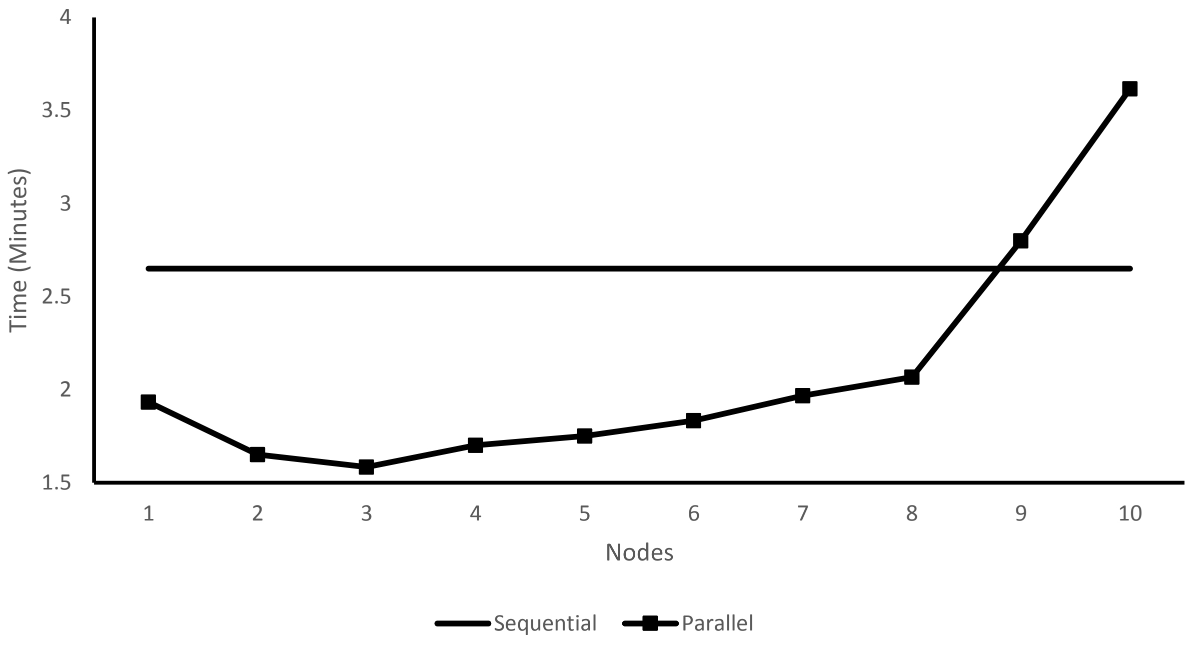


Figure 3.4: Computation Efficiency of Multiple Node System (5 Million Points).

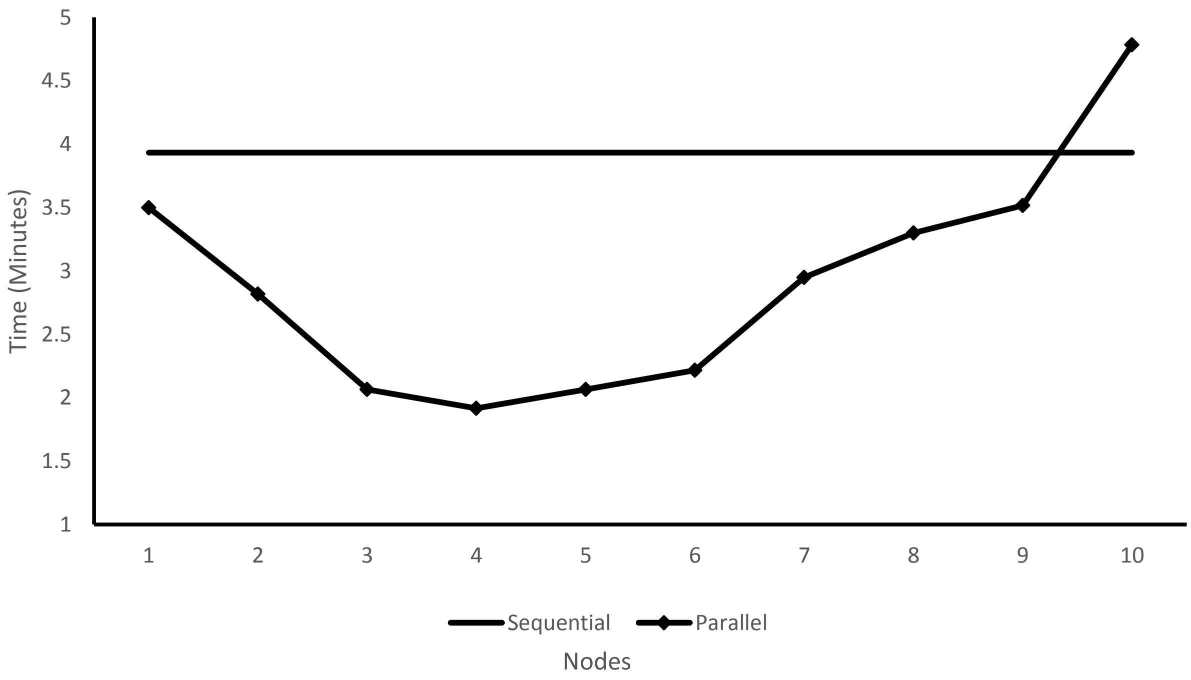


Figure3.5: Computation Efficiency of Multiple Node System (10 Million Points).

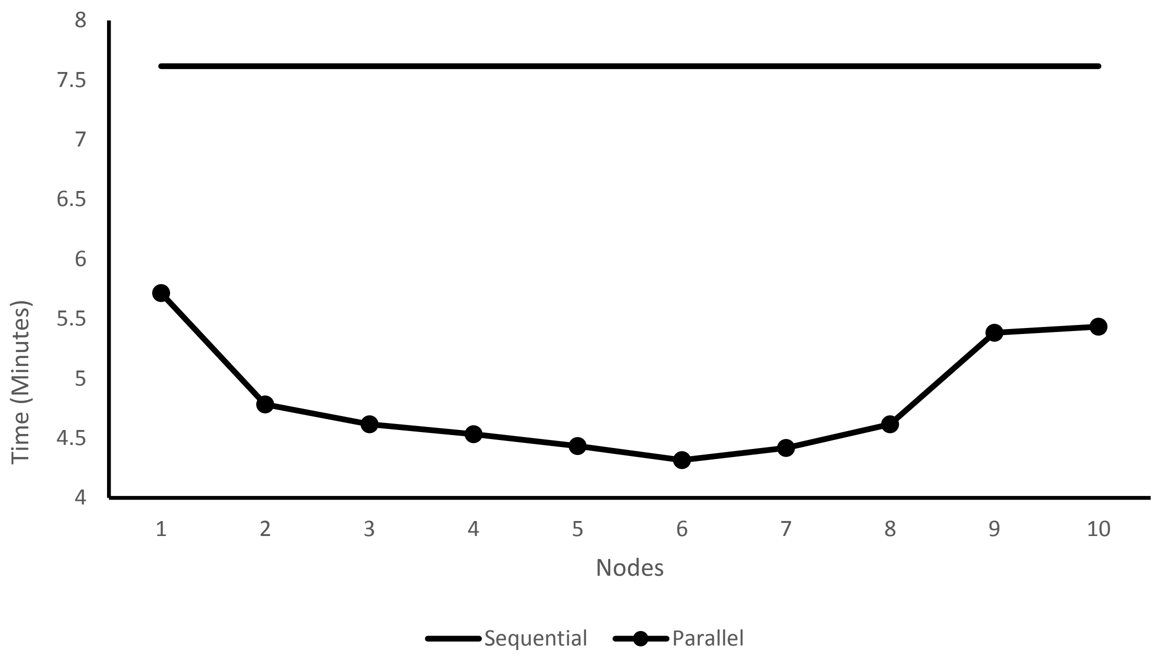


Figure3.6: Computation Efficiency of Multiple Node System (15 Million Points).

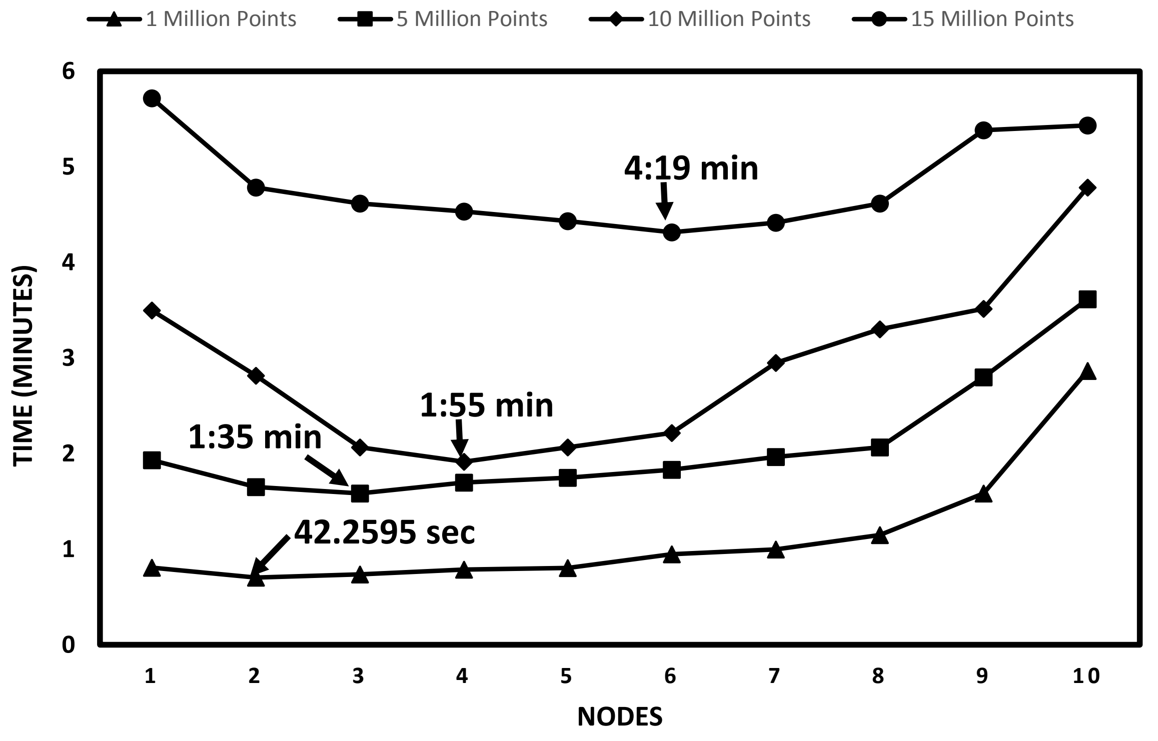


Figure 3.7: Graph showing computation efficiency of multiple node system.

According to our analysis, 1 million point data is not big enough to be processed in parallel environment hence sequential query is still a better approach for datasets containing points less than 1 million. However if for any reason one wishes to process query in parallel paradigm then 2 nodes will give the maximum efficiency close to sequential efficiency. The ideal number of nodes changes with the increase in size. As we increase the data to 5 million, the ideal performance of parallel secondo reaches to 3 node system, 10 million points show 4 node system as its best approach and for full data of 15 million points 6 nodes would be sufficient to attain maximum advantage from the system in analyzing the data. With the increase of data size optimal number of nodes will increase and therefore will be the computation power.

This shows that our proposed system is good enough for big data however for small data sets lack the ability to produce improved results. Fortunately Parallel Secondo has the compatibility of processing both sequential and parallel queries that makes it versatile in nature to handle any amount of data big or small efficiently.

Additionally there are other factors which can cause slow distribution of jobs. For instance, one factor is the cost of join. When the datasets need to be joined and stored on different nodes, the data need to be transmitted between the nodes to perform the join. Network bandwidth can also make distributed computing slower. Orakzai et al. (2015) describe partial parallelization, partial cluster utilization, network cost, disk IO, and sorting costs among others which can reduce the performance.

3.2 Pattern Analysis

The next objective to find certain patterns related to taxi trajectories or any other trajectories which could be beneficial for the society in navigation or traffic management. The GUI of parallel Secondo is relatively less user friendly as compared to proprietary software like ESRI ArcGIS, ERDAS Imagine or even some of open source software like QGIS which provide user friendly GUI and pre-developed functions in the form of buttons or tools with proper help available. As Secondo is still under development phase hence it requires a little more user friendly environment.

Some of the patterns we have identified in this research are as follows:

3.2.1 Detailed Road Mapping

We face different traffic condition at different times of the days, moreover situation changes entirely on the weekends due to the built up infrastructure and their inhabitant characteristics like if there is a school or college nearby road then there is a possibility of increase traffic situation at morning starting time and school leaving time. The same occurs if there is a number of offices nearby road that has specific leaving time. History records of trajectories could be a great tool to map the road conditions throughout the day that will makes navigation systems an advantage to suggest routes to users according to the time of the day.

Hence we choose a random road segment for our analysis that contains following landmarks adjacent to roads location which could be the influential factors for road traffic conditions.

```
Yulong 2nd St = 40.120284, 116.669145 to 40.120309, 116.674296
Contains:
```

```
Niulanshan Winery Franchise Store
Golden Phoenix
Beijing Hongleyuan Information Consultation Service Center
China Construction Bank
Postal Savings Bank of China
```

To represent the road in Parallel secondo we have to use the coordinates to *makeline()* operator which will make only one straight segment by utilizing two points. Whereas if road is not straight and composed of more than two points than *collect_line()* operator will be helpful to create line in Parallel Secondo. To cover the whole width of road buffer will be created around the line at fix distance.

```
let StartPoint = circle(makepoint(116.47111501161794, 39.952968221), 0.0005, 2)
let StopPoint = circle(makepoint(116.47620047991018, 39.954556088), 0.0005, 2)
let Road = bufferLine(makeline(center(StartPoint), center(StopPoint)), 0.0008)
```

Now to analyze this roads temporal situation of taxi trajectories we have to use multiple queries in Parallel mode as well as sequential mode for better efficiency. As parallel queries are efficient for big data hence once data is filtered out then it is better to implement sequential queries.

This ability of processing parallel as well as sequential queries on a single platform without reconfiguring makes the system flexible and easy to use.

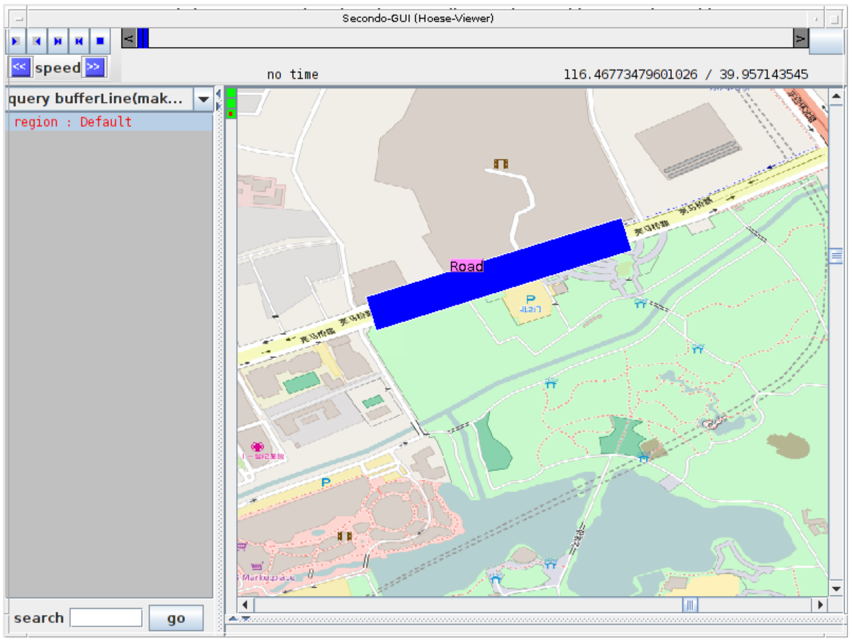


Figure 3.8: Yulong 2nd St Road representation in Secondo.

Therefore our first step should be to spread the trajectory data to all the nodes in order to start parallel query.

```
        let taxiroutes = taxiTrajectory feed spread[;TaxiID,
CLUSTER_SIZE,TRUE;];
```

This will create DLF files in all the nodes in the cluster. Now we are able to perform *hadoopMap* operator in order to filter out all the trajectories that passed the road under analysis.

```
        let Road_taxi = taxiroutes hadoopMap[DLF, TRUE
        ; . filter[(.Traj passes StartPoint) and (.Traj passes StopPoint)]
        extend[StartTime: inst(initial(.Traj at StartPoint)),
        StopTime: inst(initial(.Traj at StopPoint)),
        Roadm: .Traj at Road]]
        collect[] consume
```

Operator *initial()* will create extract the initial intime value when trajectory entered the region and *inst()* will convert that value to instant. For precise analysis we filtered the trajectories which passes both the points (starting and ending) so that only those trajectories could be included in the analysis which covered the whole road segment.

```
      let RoadPeriod = Road_taxi feed
      extend[TotalTime: get_duration(createPeriods(.StartTime,
.StopTime, TRUE, TRUE)), Length: length(gk(.Roadm))] consume
```

Another way to remove all those taxi trajectories which just crossed the road and furthermore passed through both start and stop points of road at any other time using removeshort() operator which will remove any taxi period that is short in duration.

```
      let Road_Filter = RoadPeriod feed extend[Roadmfilter: .Road3m
atperiods(removeShort(deftime(.Roadm), [const duration value (0
20000)]))] consume
```

Here all the trajectories that took under 20 seconds to cross the road will be removed and only trajectories which took more than 20 seconds will remain as the road segment is 785 meters long. Next as we wish to perform hourly analysis on this road hence we will limit our results to the fixed duration accordingly.

```
      let Roadld = Road_Filter feedproject[TaxiID, FilterTime,
Roadmfilter] extend[RoadmT: .Roadmfilter atperiods(createPeriods(
[const instant value "2008-02-04-20:00:00.000"], [const instant value
"2008-02-04-21:00:00.000"], TRUE, TRUE))] consume;
```

Here we are limiting the results only for time duration 8 pm to 9 pm on 4th February 2008. The next task includes calculating the length of the trajectory, average speed and time duration it took to cross the road and again filters will be apply in order to remove such trajectories which took too long time due to stays, crossing the road more than one time in the same hour or any other reasons. For some more precision distance filter is furthermore added in order to separate all the trajectories that stayed in the road for specific time that didn't filtered them out on time basis however have not actually cross the complete road.

```
      let Roadduration = Roadld feed extend[Roadl: length(gk(.RoadmT)),
Roadd: get_duration(deftime(.RoadmT)),Roads: avg_speed(.RoadmT,
"WGS1984")] consume;

      let Roadfilter = Roadduration feed filter[.Roadd > [const
duration value (0 10)]] filter[.Roadd < [const duration value (0
10000000)]] filter[.Roadl > distance(StartPoint, StopPoint)] consume;
```

We get three types of results in the form of duration the taxis took to cover the whole segment, average speed and the number of taxis passed that particular road in particular time.

```
      query Roadfilter feed avg[Roads];
      query create_duration(Roadfilter feed extend [Duration:
duration2real(.Roadd)] avg[Duration]);
      query Roadfilter count;
```

Hourly analysis were conducted to find any pattern in T-Drive. Daily graphs of all three results the time, speed and count of the taxi are displayed following. In the duration graph we see variations throughout the day. We could see the ups and downs at particular times. These graphs show very low correlation with one another and no prominent patterns are discernible. This is due to the small number of taxis passing the road as this data is of just one week. If we have data of months or year we could average out the daily routine of the traffic. Similarly speed graphs moreover show very little patterns whereas count graphs show one prominent pattern that count of taxi passing through the road increases after 0900 hours and gets its peak at evening around 1400 to 1600 hours.

3.2.2 Routing-Analysis

It is not always about finding the shortest route or the fastest one. For users to save fuel and make their trip economical it is essential to find a route that gives minimum breakpoints and let the vehicles to maintain constant speed as much as it can to improve the mileage of the vehicle. Taxi drivers usually knows the best routes of the city. However sometimes even they misjudge the cost of the destination they want to reach. Hence by studying different taxi trajectories of a certain time one can estimate the best route to reach the destination without wasting much fuel. These trajectories could furthermore be useful to estimate time in future according to the past records. Navigation systems like Google Maps could only estimate the time using road network without traffic influence. However this way we could be able to estimate time to reach destination keeping in view the amount of traffic on that particular day at that particular time using past experiences of the drivers. Figure 3.9, 3.10 and 3.11 displays the graphs of average duration, count and speed of taxis on the road throughout the week.

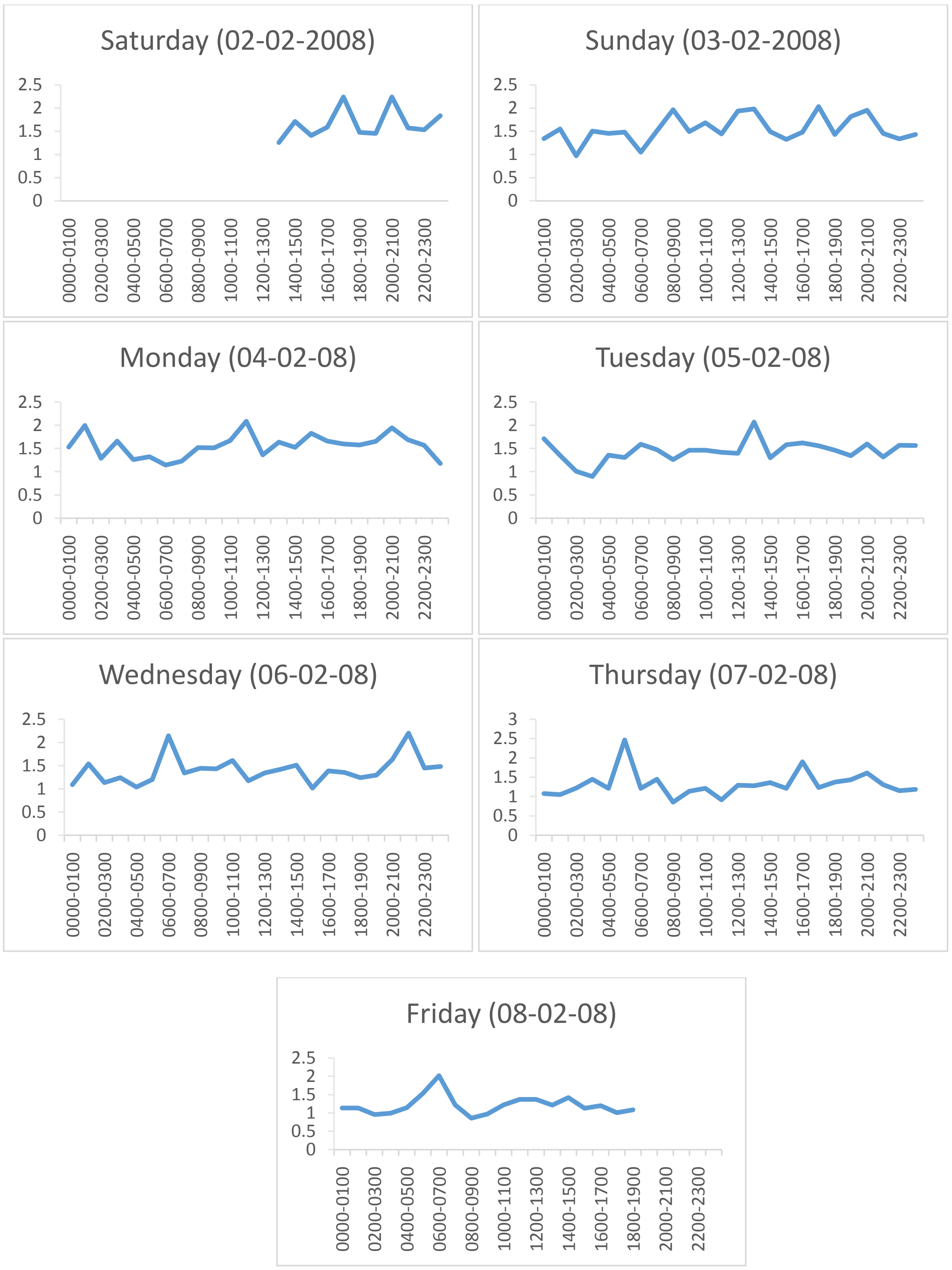


Figure3.9: Daily graphs of time duration Horizontal axes (time of day) Vertical axes (time in minutes).

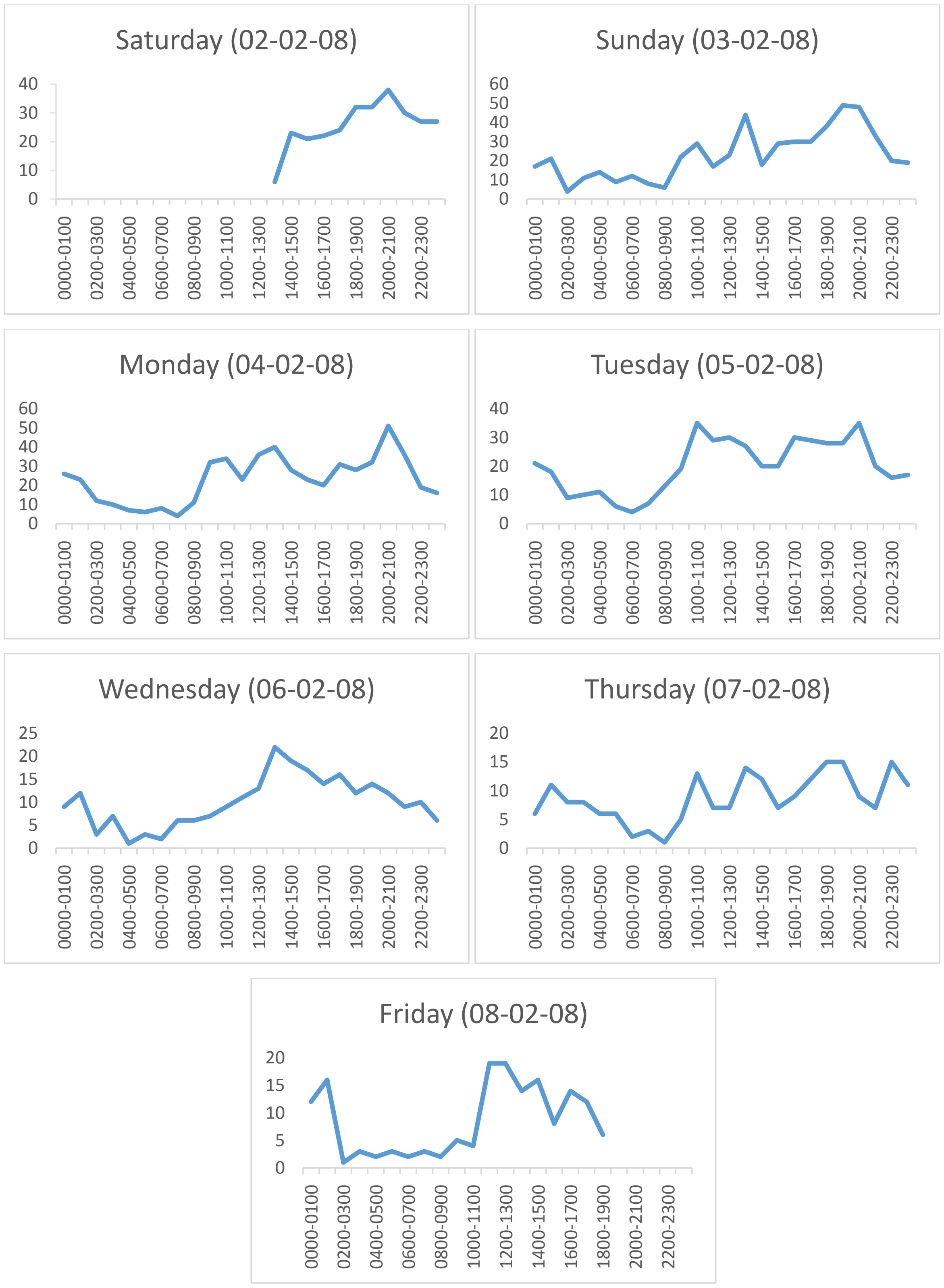


Figure 3.10: Daily Graphs of Taxi Count Horizontal axes (Time of day) Vertical axes (Time in minutes).

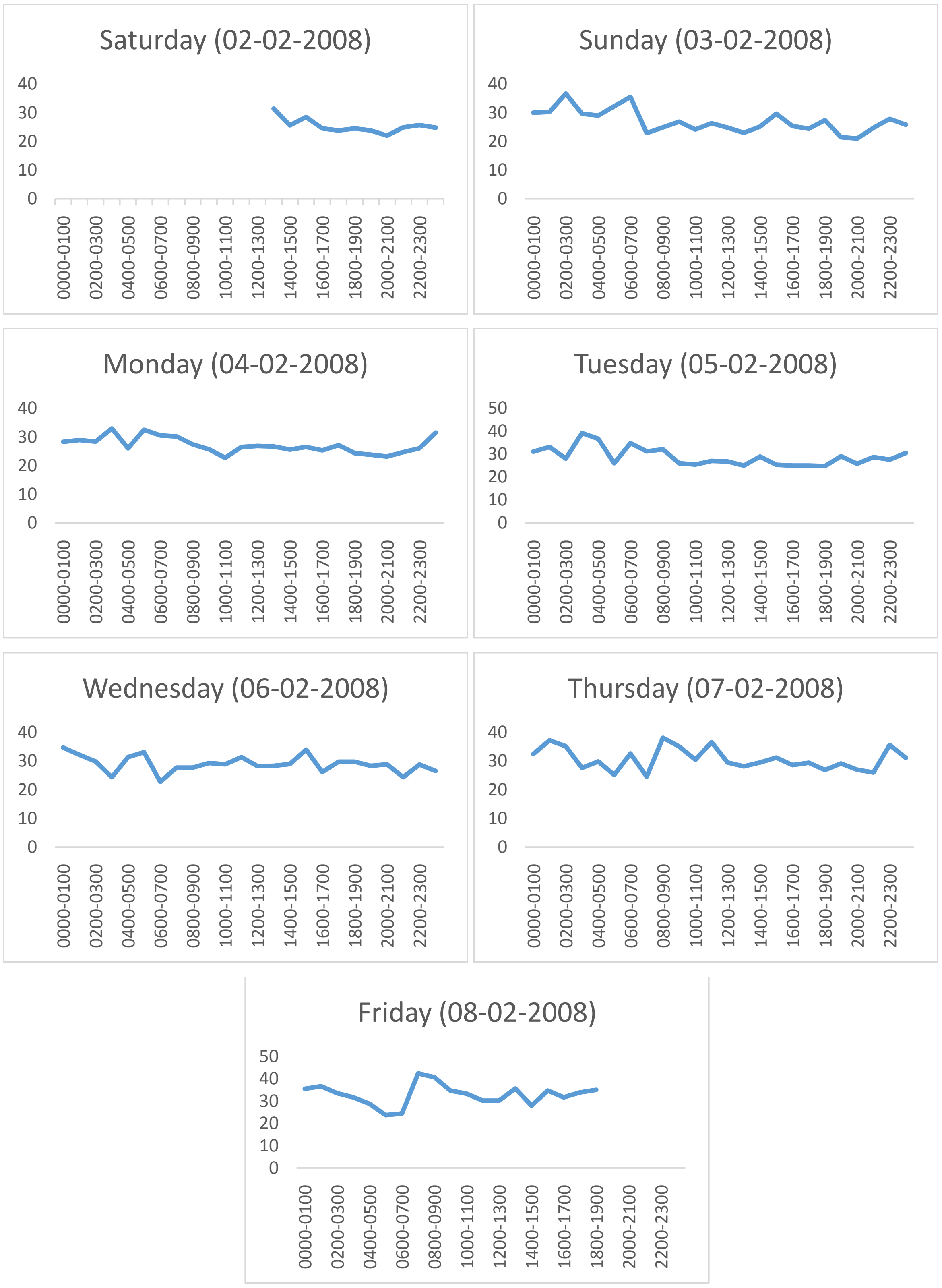


Figure3.11: Daily Graphs of Average Speed of Taxi, Horizontal (Time of day) Vertical (Speed in km/h).

Beijing Capital International Airport = 116.584321, 40.077808

Marriot City Wall Hotel = 116.434096, 39.905151

According to google map:

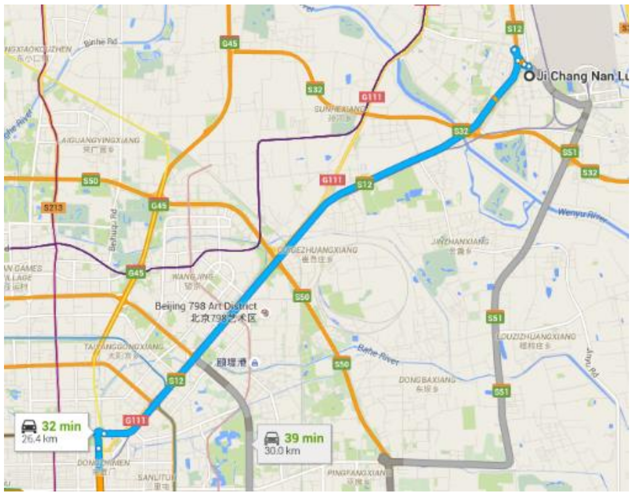
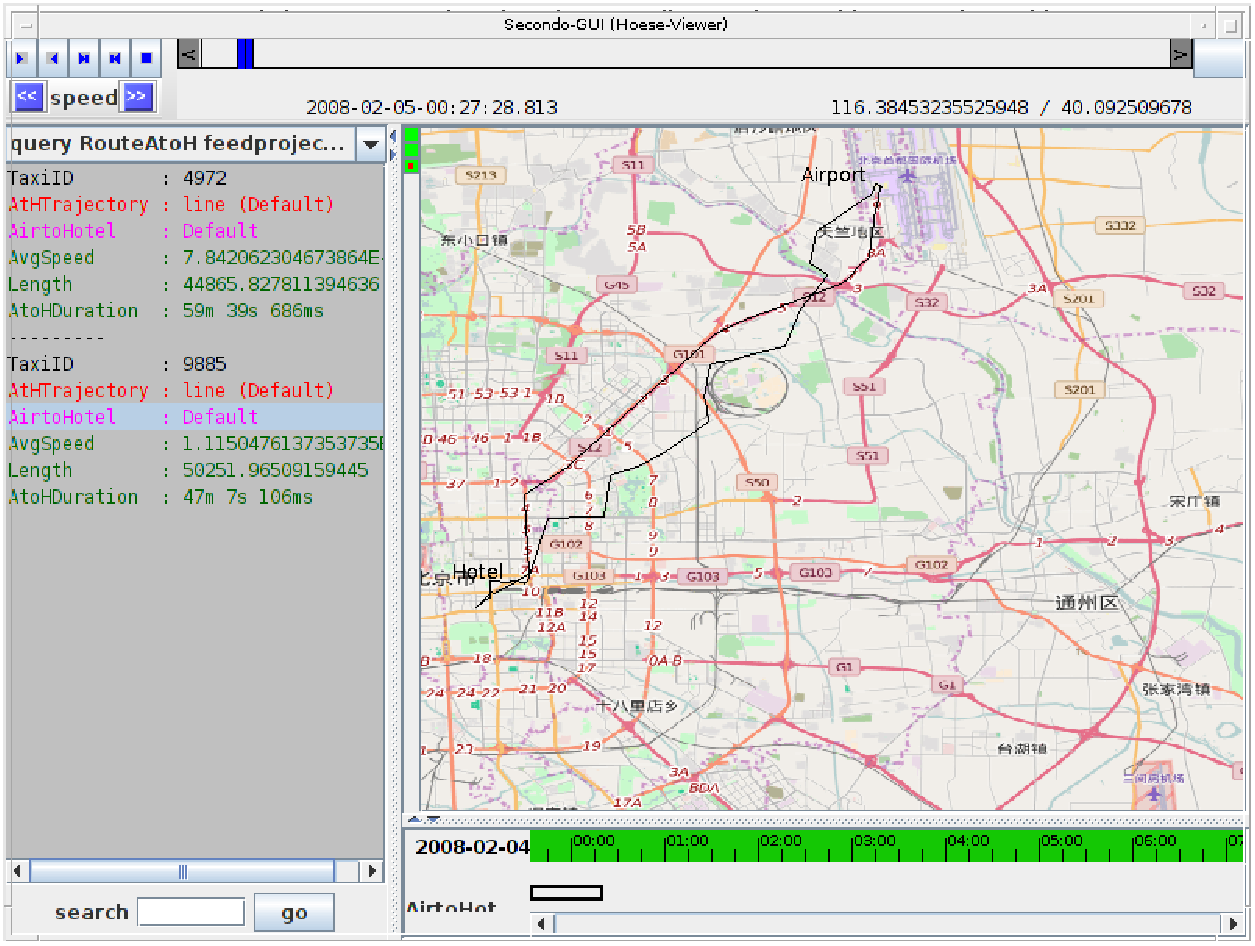


Figure 3.12: Google Map navigation system from Airport to Marriot Hotel Beijing.

Google map gives us three different routes to reach our destination moreover time required. However mostly this time does not incorporate the amount of traffic it face. These time representations could be improved using historic data of trajectories of vehicles that will share their actual experience on the road and they could determine the accurate time required to reach our destination as shown in the Figure 3.13.

By filtering all the taxis trajectories that starts their journey from airport and reach destination of hotel, we could see the actual time it took for drivers to complete their route. According to these driver the first route which Google estimated 32 minutes took almost 1 hour. However chances of errors here are more as we have example of only two taxis. However if we have data of taxi trajectories for more duration like month or year then we will have more examples of trajectories that took the same route on different time which will let us analyze the condition of the route at different period of time in a day and week. Hence more data is required to complete the analysis accurately.



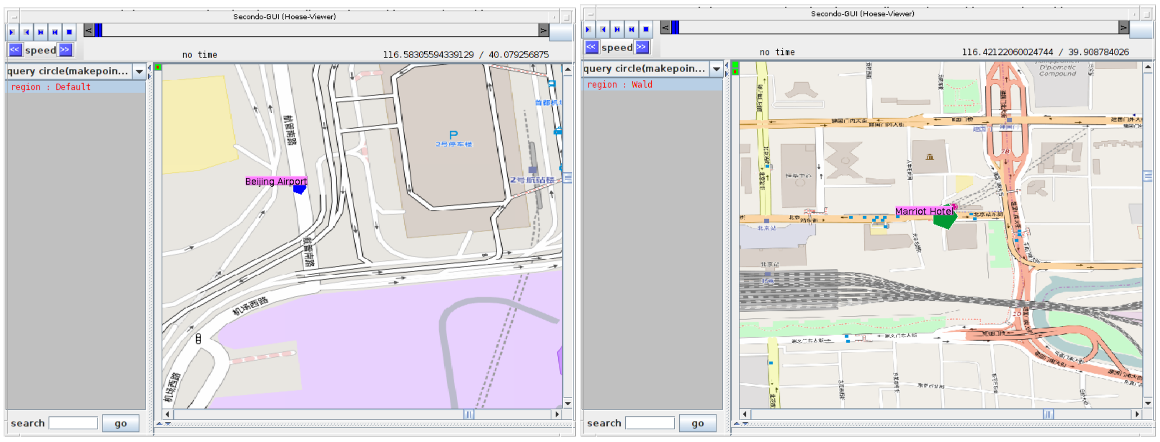
```
TaxiID          : 4972
AtHTrajectory   : line (Default)
AirtoHotel      : Default
AvgSpeed        : 7.842062304673864E
Length          : 44865.827811394636
AtoHDuration    : 59m 39s 686ms
---------
TaxiID          : 9885
AtHTrajectory   : line (Default)
AirtoHotel      : Default
AvgSpeed        : 1.1150476137353735E
Length          : 50251.96509159445
AtoHDuration    : 47m 7s 106ms
```

Figure 3.13: Directions suggested by Routing Analysis in Parallel Secondo GUI.

To achieve this analysis following queries were utilized. First step contains demarcation of airport and hotel location. By making point according to their latitude and longitude coordinates we will get point values of both airport and hotel. However as chances of taxi trajectories of passing right through that point is low hence a region is made as circle around both points that covers the road which connects these points.

```
let Airport = circle(makepoint(116.584321, 40.077808), 0.0005, 5)
```

```
        let Hotel = circle(makepoint(116.4273380799, 39.903717084),
0.0005, 5)
```



(a) (b)

Figure3.14: Representation in Parallel SECONDO (a) Airport (b) Marriot Hotel.

Here circle operator is responsible for making region by carrying out three parameters as point, radius distance and edges of the shape (value should be within 2-101). Next step contains filtering out all the trajectories that passes both these location/regions. For this purpose we would process parallel query as system has to compute all 10,356 taxi trajectories and will take time hence using hadoopMap operator to perform the filter operation. There is no need to spread the data again as we had already spread trajectory data to all the nodes in the previous analysis.

```
        let Route = taxiroutes hadoopMap[DLF, TRUE
        ; . filter[(.Traj passes Airport) and (.Traj passes Hotel)]
        extend[AirportTime: inst(initial(.Traj at Airport)), HotelTime:
inst(initial(.Traj at Hotel))]]
        collect[] consume
```

This will result all the trajectories that passed these two points. Furthermore we computed the time at which they passed these locations in order to analyze that whether these trajectories chose to go from airport to hotel or they were just passing by at different points.

```
      let RoutePeriod = Route feed extend[AtoHTime:
createPeriods(.AirportTime, .HotelTime, TRUE, TRUE)] consume
```

This will create the period in between both passing. Now using this period we will create the trajectory in form of line as well as moving point (mpoint) for representation purposes.

```
      let Routempoint = RoutePeriod feed extend[AirtoHotel: .Traj
atperiods(.AtoHTime), AtoHDuration: get_duration(.AtoHTime)] consume
```

Operator get_duration() will convert period into time duration and the atperiods() operator is useful for creating mpoint according to the given period. Finally some time filters should be placed in order to extract only those taxis that choose the direct path from airport to hotel.

```
      let RouteAtoH = Routempoint feed extend[AtHTrajectory:
trajectory(.AirtoHotel),
      AvgSpeed: avg_speed(.AirtoHotel), Length:
length(gk(.AirtoHotel))]
      filter[.AtoHDuration > [const duration value (0 1500000)]]
      filter[.AtoHDuration < [const duration value (0 4500000)]]
consume
```

As Google maps stated the time to reach hotel through any path greater than 30 minutes hence 25 minutes was chosen as lower limit for the trajectory keeping in view some exceptions and the upper limit be given as 75 minutes in case of some really worst traffic jam situations. Some extra information could furthermore be collected like the average speed of the vehicle and the distance travelled by the vehicle which are accomplished by using avg_speed() and length() operator.

Now for representation it would be wise to display only required attributes as this process could take time if entries are in large amount. The feedproject[] operator allow us to choose the attributes we want to display.

```
query RouteAtoH feedproject[TaxiID, AtHTrajectory, AirtoHotel,
AvgSpeed, Length, AtoHDuration] consume
```

There are chances that the taxi might have stopped at a particular stop in search of passengers. That will disrupt the analysis integrity and estimation of time would be inaccurate hence breakpoints could moreover be analyzed if the taxi choose to stay at a point for a longer time apart from signals.

```
query RouteAtoH feed extend[AtHBreakPoints:
breakpoints(.AirtoHotel, create_duration(0,10000))]] consume
```

Stay duration is greater than 10 seconds. In our condition none of the two taxi stops at any point that shows the integrity of the analysis.

3.2.3 Brakes Analysis

Analyzing the points where vehicles have to apply brake most of the time or gets stuck in the traffic could allow us to identify the reason of obstruction and how the users can avoid such situations of sticking in traffic jams.

For such analysis Parallel Secondo provides several operators to query the stay points and then clustering options using DBScan and maximum distance techniques. These analysis will be helpful to analyze the regions where most of the times vehicles have to stop due to traffic jam, signal or for parking purposes.

In parallel processing finding break points could be accomplished by using following lines of commands:

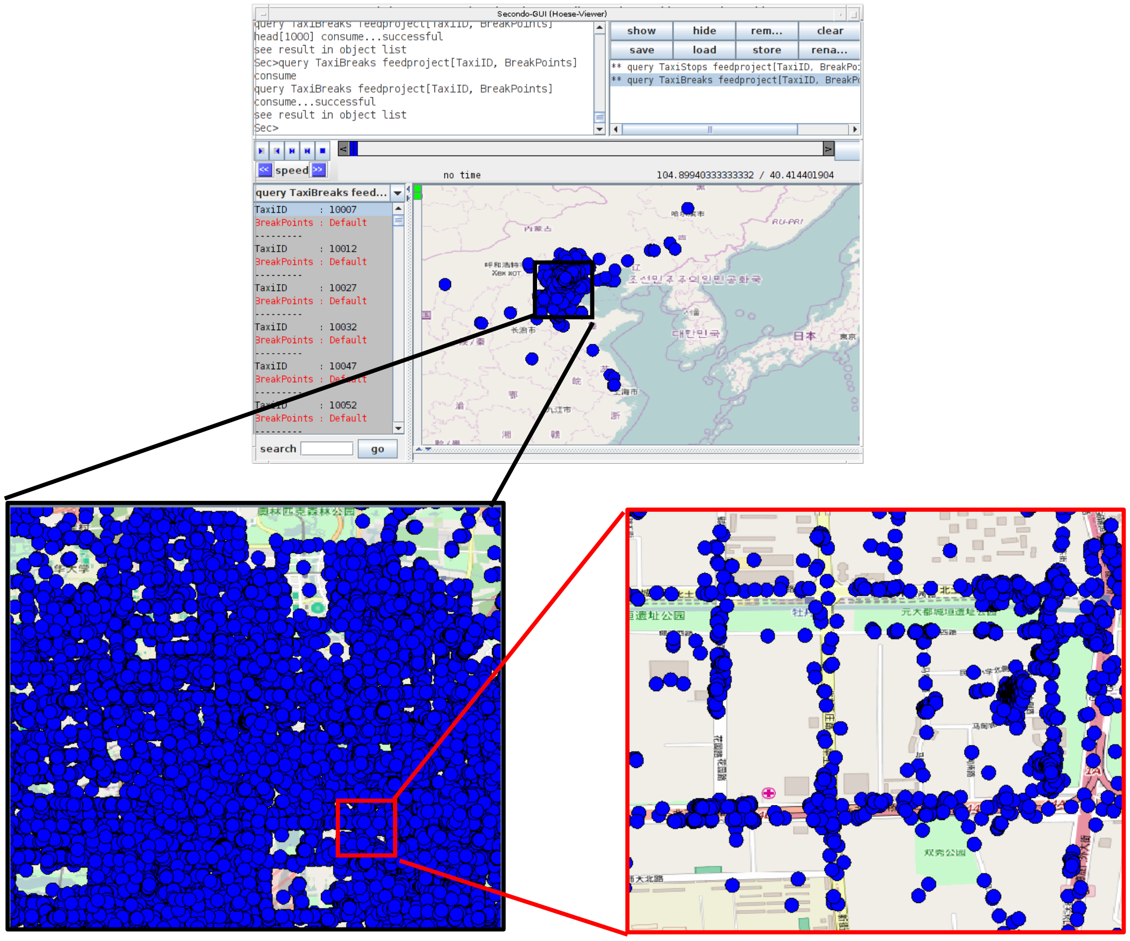
```
      let TaxiBreaks = taxiroutes hadoopMap[DLF, TRUE
      ; . extend[BreakPoints: breakpoints(.Traj,
create_duration(0,60000))] ]
      collect[] consume
```

Here all the trajectories will be computed and all breaks in the trajectories where taxi stayed for more than a minute will be stored in TaxiBreak variable in form of points. Projecting all these points on Parallel Secondo GUI would cause immense amount of computation power and time, moreover there are chances that the system crashes for displaying such large amount of data hence for displaying purpose we selected break points of top 1000 trajectories in Figure 3.15 using *head*[] operator.

```
query TaxiBreaks feedproject[TaxiID, BreakPoints] head[1000] consume
```

However displaying break points will not be useful enough until we perform some clustering analysis which will determine the regions of interest. There are different types of clustering algorithms present in the Parallel Secondo environment which includes DBSCAN and maximum distance based algorithms. Examples of both algorithms and their results are shown in the Figure 3.16 and 3.17.

```
let taxi_breaks = TaxiBreaks feed spread[;TaxiID, CLUSTER_SIZE, TRUE;]



let BreaksClusterDB = taxi_breaks hadoopMap[DLF, TRUE

; . projecttransformstream[BreakPoints] collect_points[TRUE]

cluster_c[100, 0.005] namedtransformstream[BCluster] ]

collect[] consume
```



```
query TaxiBreaks feedproject[TaxiID, BreakPoints]
head[1000] consume...successful
see result in object list
Sec>query TaxiBreaks feedproject[TaxiID, BreakPoints]
consume
query TaxiBreaks feedproject[TaxiID, BreakPoints]
consume...successful
see result in object list
Sec>
```

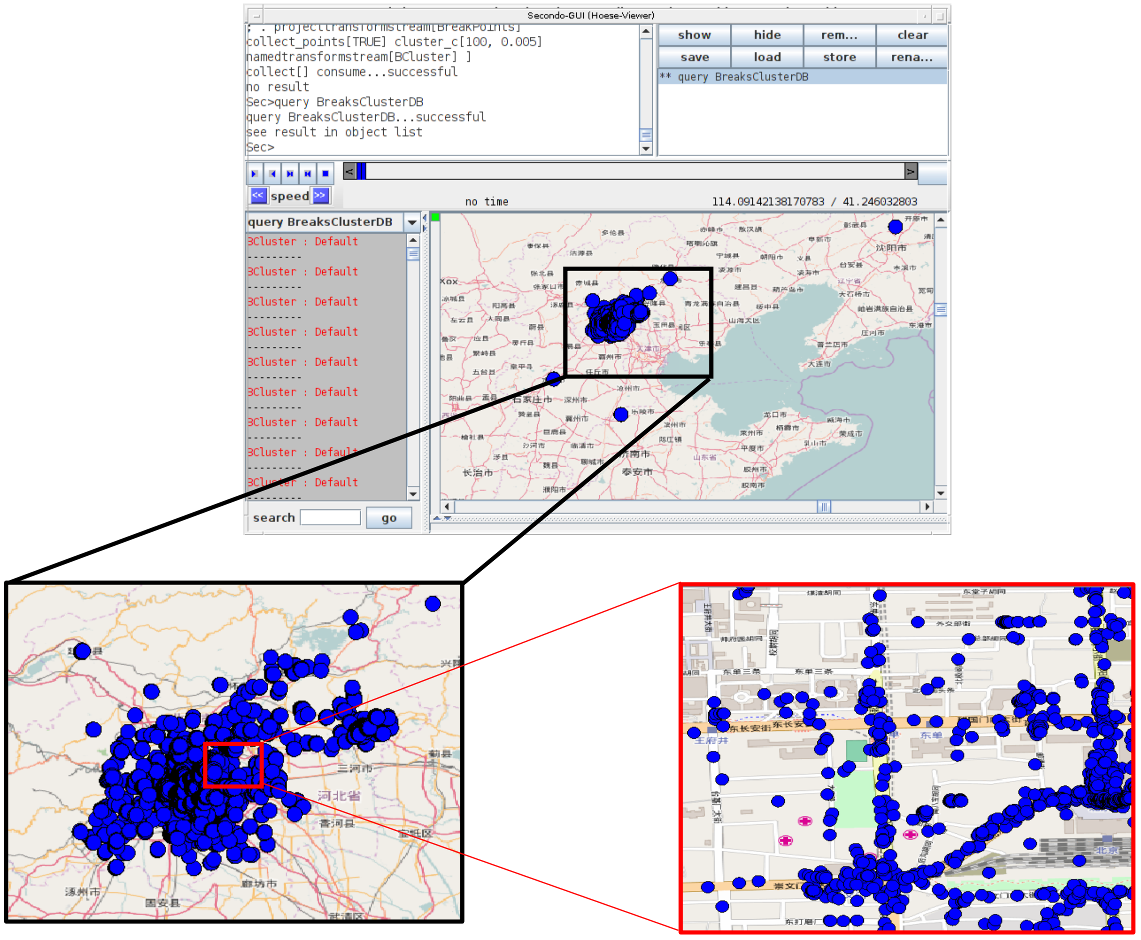
```
** query TaxiStops feedproject[TaxiID, BreakPo
** query TaxiBreaks feedproject[TaxiID, BreakP
```

```
no time                104.89940333333332 / 40.414401904
```

```
query TaxiBreaks feed...
TaxiID      : 10007
BreakPoints : Default
---------
TaxiID      : 10012
BreakPoints : Default
---------
TaxiID      : 10027
BreakPoints : Default
---------
TaxiID      : 10032
BreakPoints : Default
---------
TaxiID      : 10047
BreakPoints : Default
---------
TaxiID      : 10052
BreakPoints : Default
---------
```

Figure 3.15: Brake Points of taxi trajectories in Beijing.

Clustering the brake points using DBSCAN algorithm gives us advantage to study those regions where massive amount of occurrence took place. As the figure above displayed the city center crowded with break point clusters. A one pattern to be noticed is that the point density increases vigorously at the junctions and crossings which could identify the positions of traffic signals.



```
; . projecttransformstream[BreakPoints]
collect_points[TRUE] cluster_c[100, 0.005]
namedtransformstream[BCluster] ]
collect[] consume...successful
no result
Sec>query BreaksClusterDB
query BreaksClusterDB...successful
see result in object list
Sec>
```

```
** query BreaksClusterDB
```

```
no time          114.09142138170783 / 41.246032803
```

```
query BreaksClusterDB
BCluster : Default
---------
BCluster : Default
---------
BCluster : Default
---------
BCluster : Default
---------
BCluster : Default
---------
BCluster : Default
---------
BCluster : Default
---------
BCluster : Default
---------
BCluster : Default
---------
```

Figure 3.16: Clustered results of DBScan Algorithm.

Similarly we applied maximum distance based clustering and compared with DBSCAN clustering.

```
        let BreaksClusterMD = taxi_breaks hadoopMap[DLF, TRUE
        ; . projecttransformstream[BreakPoints] collect_points[TRUE]
cluster_d[0.005] namedtransformstream[BCluster] ]
        collect[] consume
```

Until now our data was not significantly big to make a difference. However some queries require more computation power than the others like the one mentioned above. Clustering requires powerful computation as it has to analyze all the points and their neighborhood. Hence processing this query in parallel environment and comparing with sequential mode gives us a significant difference. Until now we were having time efficiency of maximum 2 minutes on this data. However clustering query as DBScan algorithm gives us time efficiency of 53.7786 seconds on 10 nodes in comparison of sequential query with processing time of 5:06 minute. If we scale up this difference we will get the true application of big data and its advantages. Even the maximum distance based algorithm creates a more significant difference where parallel query took 1:41minute processing time and sequential query after 20 minutes gives no response. This shows the effectiveness of parallel environment.

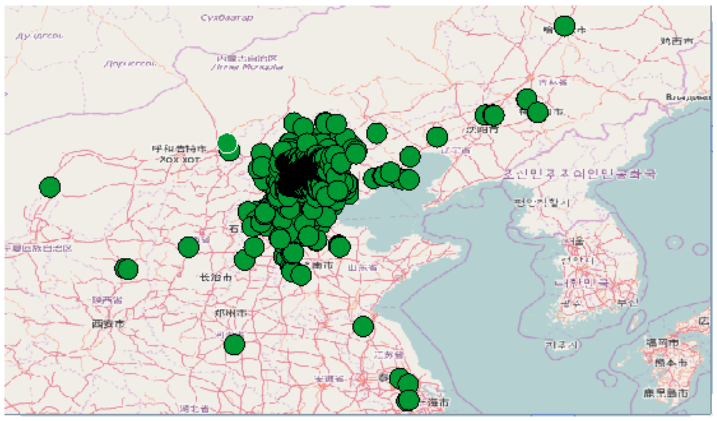
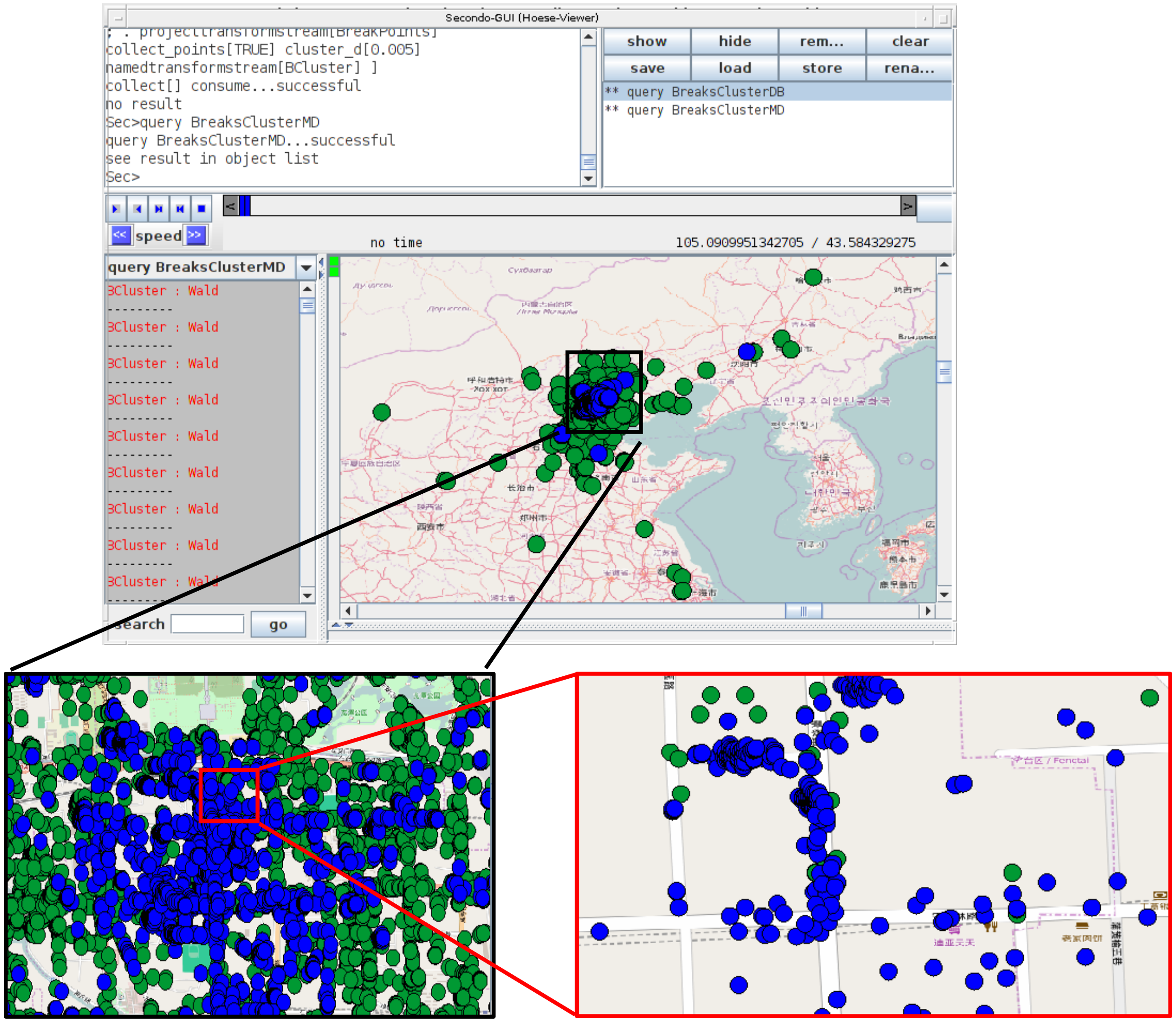


Figure 3.17: Clustered Results of Break Points using Maximum-Distance algorithm.

Green points represent maximum distance based clustering whereas blue points are the DBSCAN clustering points. Count of DBSCAN clusters are way less than maximum distance based that is 702 to 28299 respectively. This is because DBScan algorithm put limits at clusters to have minimum number of specified points (in our case 100) so that clusters could be made only those places where density of the points is higher whereas maximum distance based cluster have no such limitations.



```
; . projecttransformstream[BreakPoints]
collect_points[TRUE] cluster_d[0.005]
namedtransformstream[BCluster] ]
collect[] consume...successful
no result
Sec>query BreaksClusterMD
query BreaksClusterMD...successful
see result in object list
Sec>
```

```
** query BreaksClusterDB
** query BreaksClusterMD
```

```
no time                105.0909951342705 / 43.584329275
```

```
query BreaksClusterMD
BCluster : Wald
---------
BCluster : Wald
---------
BCluster : Wald
---------
BCluster : Wald
---------
BCluster : Wald
---------
BCluster : Wald
---------
BCluster : Wald
---------
BCluster : Wald
---------
BCluster : Wald
---------
```

Figure 3.18: Comparison between Cluster density of DBScan and Maximum-Distance algorithm.

3.2.4 Stops Analysis

Analyzing the trajectories of history in a bulk could let us have important information about where most taxi drivers tend to find their passengers and which point will be close to the user who want to avail the services of taxi.

This analysis can be done by the same method as applied in the previous analysis of break points by increasing the duration value to one hour. It will compute all the points where the taxi has stayed at one place for more than one hour. This could be taxi parking area or any other parking slot. The results of such analysis is shown in Figure 19.

```
        let TaxiStops = taxiroutes hadoopMap[DLF, TRUE
        ; . extend[BreakPoints: breakpoints(.Traj,
create_duration(0,3600000))]]
        collect[] consume
```

Figure 3.19 shows the stop or parking points of different Taxi. Mostly these points are off road that represents parking areas inside buildings or off road. If we perform DBScan cluster analysis to check at which parking region more than 50 taxi parked in the span of 7 days. We get the results as shown in Figure 3.20:

Blue dots shown in Figure 3.20 represents the cluster of taxis inside a parking spot. If we increase the maximum distance parameter to 0.01. We will get some more results:

As we see now we get three different locations of parking spots. By decreasing the minimum number of points to 20 we even get more frequent locations where most of the taxi drivers prefer to park.

3.2.5 Speed Analysis

Analyzing the areas and time where most drivers exceeds the speed limit of the road, will help the traffic control units to take precautions at that particular time and place to avoid accidents.

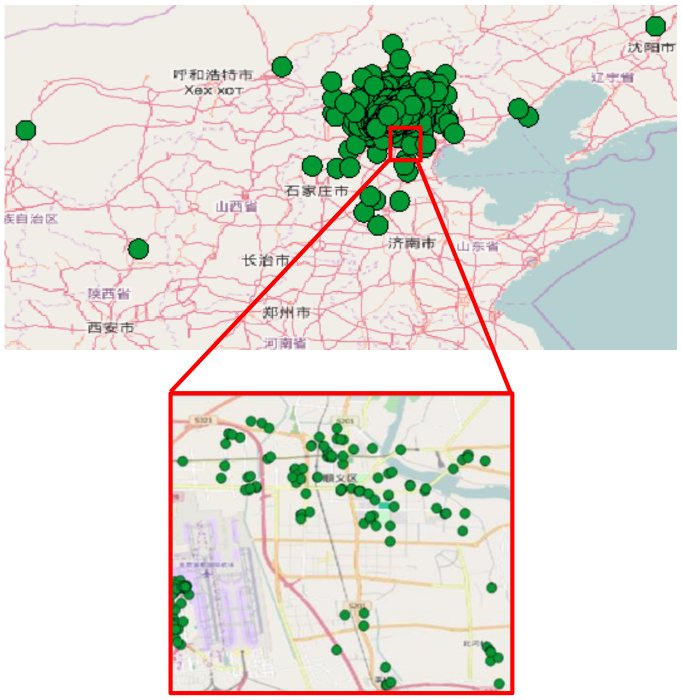
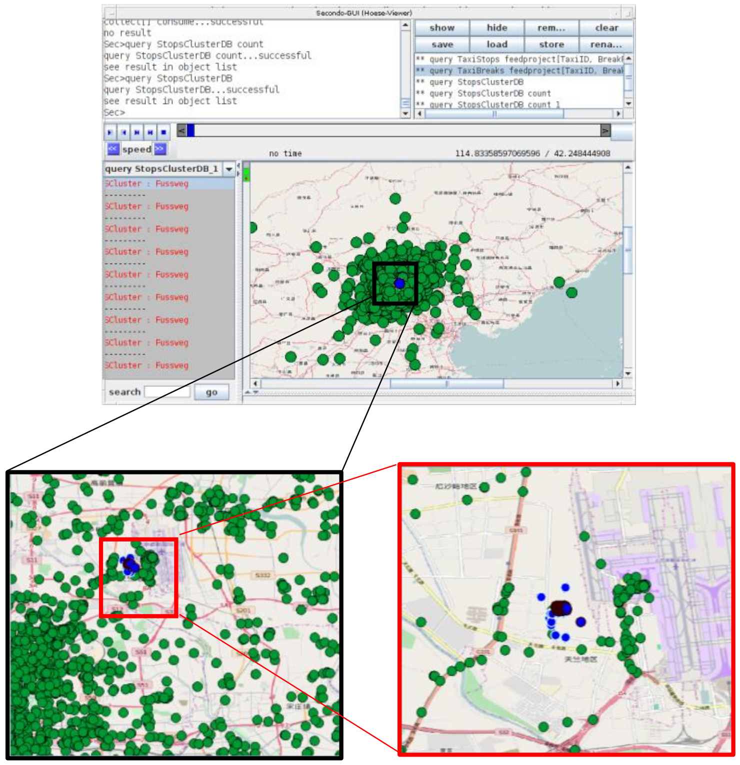


Figure 3.19: Stop Points of Taxi's in Beijing City.

For such analysis we use speed operator which determines the speed of the moving point and accordingly filters out the required results we need. Following set of commands are utilized in order to carry on the process.

```
let Speed120 = taxiroutes hadoopMap[DLF, TRUE
; . extend[Speedm: .Traj when[speed(.Traj,
create_geoid("WGS1984"))>33.333]] ]
      collect[] consume
```



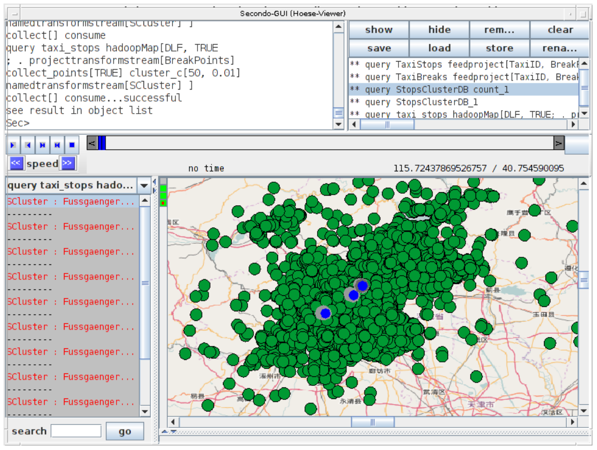
```
collect[] consume...successful
no result
Sec>query StopsClusterDB count
query StopsClusterDB count...successful
see result in object list
Sec>query StopsClusterDB
query StopsClusterDB...successful
see result in object list
Sec>
```

```
** query TaxiStops feedproject[TaxiID, Break
** query TaxiBreaks feedproject[TaxiID, Break
** query StopsClusterDB
** query StopsClusterDB count
** query StopsClusterDB count_1
```

```
no time                114.83358597069596 / 42.248444908
```

```
SCluster : Fussweg
---------
SCluster : Fussweg
---------
SCluster : Fussweg
---------
SCluster : Fussweg
---------
SCluster : Fussweg
---------
SCluster : Fussweg
---------
SCluster : Fussweg
---------
SCluster : Fussweg
---------
SCluster : Fussweg
```

Figure 3.20: DBScan Cluster of minimum 50 points.



```
namedtransformstream[SCluster] ]
collect[] consume
query taxi_stops hadoopMap[DLF, TRUE
; . projecttransformstream[BreakPoints]
collect_points[TRUE] cluster_c[50, 0.01]
namedtransformstream[SCluster] ]
collect[] consume...successful
see result in object list
Sec>
```

```
** query TaxiStops feedproject[TaxiID, Break
** query TaxiBreaks feedproject[TaxiID, Break
** query StopsClusterDB count_1
** query StopsClusterDB_1
** query taxi_stops hadoopMap[DLF, TRUE; . pr
```

```
no time                115.72437869526757 / 40.754590095
```

```
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
SCluster : Fussgaenger...
---------
```

Figure 3.21: DBScan cluster of minimum 20 points.

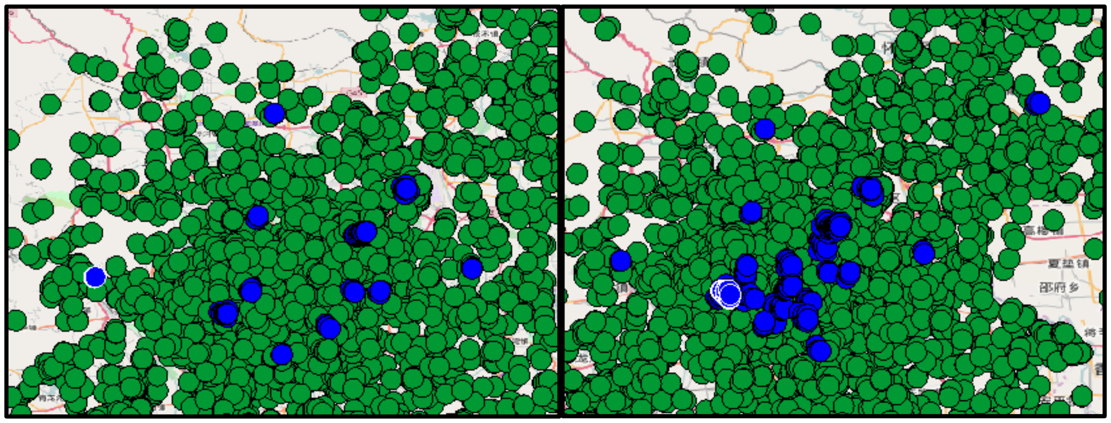


Figure 3.22: (a) Clusters with minimum point value 20 and maximum distance value 0.005.(b) Clusters with minimum point value 20 and maximum distance value 0.01.

After collecting those moving points that move more than 120 km/h (33.333 m/s) speed in another attribute we will again spread the new data on parallel nodes in order to carry out results faster.

```
let taxi_speed120 = Speed120 feed spread[;TaxiID, CLUSTER_SIZE, TRUE;]

let Speed120V = taxi_speed120 hadoopMap[DLF, TRUE
; . project[TaxiID, Speedm] extend[Speedv: vertices(.Speedm)] ]
collect[] consume


query Speed120V feed filter[maximum(speed(.Speedm,
create_geoid("WGS1984")))>33.333] extend[MaxSpeed:
maximum(speed(.Speedm, create_geoid("WGS1984")))] consume
```

The resultant query will give us 2867 taxi that violated the speed limit of 120 inside and outside the city out of 10,356. Figure 3.23 shows the regions where speed exceeded.

We could see that points reach the far end of the Beijing on expressway and motorways which are likely to go faster. However if we apply cluster algorithm using 20 as minimum number of points and 0.01 as distance parameter we get the results as shown in Figure 3.23:

The results in Figure 3.24 show that even in city center drivers tend to go faster at several occasions.

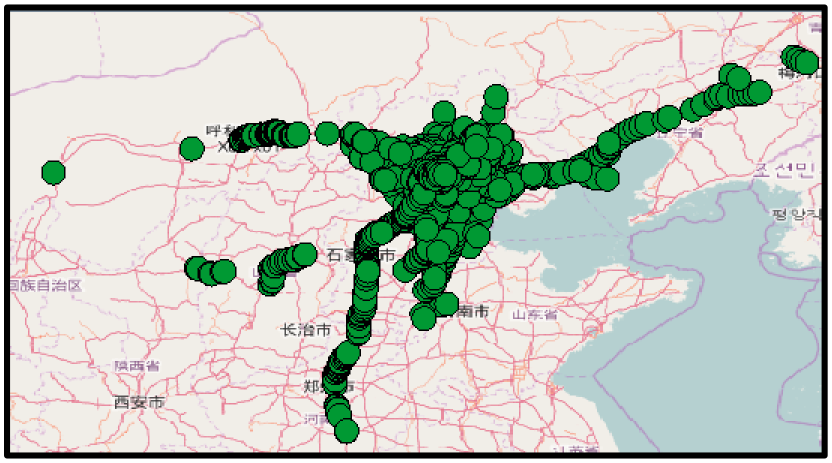


Figure 3.23: Points where taxi speed exceed 120 km/h.

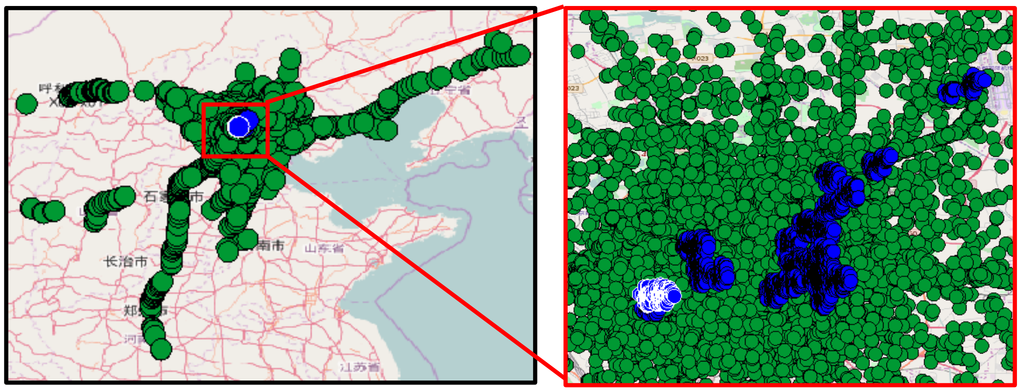


Figure 3.24: DBScan cluster of speeding points.

Chapter 4

CONCLUSION AND RECOMMENDATIONS

4.1 Conclusions

This study demonstrates a series of experiments which were carried out in a distributed spatio-temporal DBMS, Parallel Secondo, to test its efficiency. Parallel Secondo uses the Hadoop framework to share tasks across multiple processing nodes. T-drive datasets with varying numbers of data points were queried using standard spatio-temporal queries in sequential and parallel frameworks of Parallel Secondo. The results highlight that increasing the number of nodes in the distributed system will not always produce efficiency. This is due to the unavoidable overhead of distributing the processing to multiple nodes. The number of data points and the complexity of the spatio-temporal query are the principle factors which determine if parallel processing will be more efficient. The results of the study suggest optimal node numbers for different cases of queries and different volumes of data. This will serve as a guide for researchers wishing to improve the efficiency of spatiotemporal queries on large datasets.

Vehicles trajectories furthermore show the behavior of the driver and its needs. Analyzing such prospects leads us to an understanding world and brings easiness in human society. Knowledge extraction from historic data could be much more valuable than expectations if used correctly. Need of big data techniques for such analysis are imminent and requires much opportunities for researchers to focus on different innovative analysis using fast processing. These techniques should be refined in such a way that common user or researcher with low IT background could efficiently use such system in order to carry out analysis of its own field as data is everywhere and the need of studying data and extracting information from that data is increasing day by day with the evolution of technology and need of more luxury in human life.

In future, we plan to use Amdahl's law in order to check the distribution strategy of Parallel Secondo and the type of query being run. Amdahl's law is defined as "a law governing the speedup of using parallel processors on a problem, versus using only one serial processor." In the current study, the default data partitioning strategy of Parallel Secondo is not optimised for all kinds of queries. The users should calculate the probability of each type of query a benchmark model BerlinMOD has (e.g. range, point, join etc.) and based on that decide the best partitioning strategy which has the least overall querying cost.

4.2 Recommendations

We furthermore suggest some of the analysis that are possible if we could have live feed of the trajectories from different vehicles such as:

4.2.1 Crime Investigation

Increasing crime rates in every corner of the world need a special concern. Using trajectories of the vehicles we could establish a system to track down the path of the criminal's vehicle by tracking the nearby vehicles at that particular instant which could make possible to get information of the culprits heading in a particular direction.

4.2.2 Commercial Advertisement

Sending messages to the valuable customers about the new products and sale offering by different companies is becoming a famous tool for promotion and advertisement of the product. However to identify valuable customers would be a plus point rather than sending messages on mobiles to random users. One way to estimate these particular customers could be done by studying the behavior of different personalities by analyzing their vehicle trajectories like which vehicles have the same route to their owner's work where the outlet of the company is present. Where vehicle visits most of the time will determine the interests of that particular owner's taste.

4.2.3 Live traffic reports

Having live feed of the traffic could make the navigation system to next level. If a vehicle is stuck in a traffic jam identified by the trajectory condition of that vehicle could allow navigation system to divert all other users to some other route so that they could avoid the traffic jam.

REFERENCES

1. Chen, C. P., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, 275, 314-347.

2. Di Felice, P., Di Lonardo, L., & Petrocco, M. (2013). Querying a trajectories database about sex offenders. *International Journal of Database Management Systems*, *5*(1), 21–33.

3. Eldawy, A. (2014, June). SpatialHadoop: towards flexible and scalable spatial processing using mapreduce. In *Proceedings of the 2014 SIGMOD PhD symposium* (pp. 46-50). ACM.

4. Gao, S., Li, L., Li, W., Janowicz, K., & Zhang, Y. (2014). Constructing gazetteers from volunteered Big Geo-Data based on Hadoop. *Computers, Environment and Urban Systems*.

5. Giachetta, R. (2015). Computers & graphics special section on processing large geospatial data A framework for processing large scale geospatial and remote sensing data in MapReduce environment. *Computers and Graphics*, 1–10.

6. Golpayegani, N., & Halem, M. (2009, September). Cloud computing for satellite data processing on high end compute clusters. In *2009 IEEE International Conference on Cloud Computing* (pp. 88-92). IEEE.

7. Jiamin, L., & Guting, R. H. (2014). Parallel SECONDO: A practical system for large-scale processing of moving objects. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on* (pp. 1190–1193).

8. Lee, J.-G., & Kang, M. (2015). Geospatial Big Data: Challenges and opportunities. *Big Data Research, 2*(2), 74–81. http://doi.org/10.1016/j.bdr.2015.01.003

9. Lee, K., Ganti, R. K., Srivatsa, M., & Liu, L. (2014, November). Efficient spatial query processing for big data. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (pp. 469-472). ACM.

10. Li, J., Meng, L., Wang, F. Z., Zhang, W., & Cai, Y. (2014). A Map-Reduce-enabled SOLAP cube for large-scale remotely sensed data aggregation. *Computers & Geosciences*, *70*, 110–119.

11. Liu, W., Zheng, Y., Chawla, S., Yuan, J., & Xie, X. (2011). Discovering spatio-temporal causal interactions in traffic data streams. *Proceedings of the 17th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11), 1010–1018.

12. Lu, J. (2014). *Parallel SECONDO: processing moving objects data at large scale* (Doctoral dissertation, Hagen, Univ., Diss., 2014).

13. Mathew, W., Raposo, R., & Martins, B. (2012, September). Predicting future locations with hidden Markov models. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (pp. 911-918). ACM.

14. Orakzai, F. M. (2014). Trajectory data modeling and processing in HBase (Doctoral dissertation, Technische Universität Berlin).

15. Uddin, M. R., Ravishankar, C., & Tsotras, V. J. (2011, June). Finding regions of interest from trajectory data. In *2011 IEEE 12th International Conference on Mobile Data Management* (Vol. 1, pp. 39-48). IEEE.

16. Vitolo, C., Elkhatib, Y., Reusser, D., Macleod, C. J. A., & Buytaert, W. (2015). Web technologies for environmental big data. *Environmental Modelling & Software*, *63*, 185–198.

17. Yu, B., Sen, R., & Jeong, D. H. (2013). An integrated framework for managing sensor data uncertainty using cloud computing. *Information Systems*, *38*(8), 1252–1268.

18. Yu, Y., Zhao, J., Wang, X., Wang, Q., & Zhang, Y. (2015). Cludoop: An efficient distributed density-based clustering for big data using Hadoop. *International Journal of Distributed Sensor Networks*, *2015*, 1–13.

19. Yuan, J., Zheng, Y., Xie, X., & Sun, G. (2011). Driving with knowledge from the physical world. *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*, (5), 316–324.

20. Yuan, J., Zheng, Y., Xie, X., & Sun, G. (2013). T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE Transactions on Knowledge and Data Engineering*, *25*(1), 220–232.

21. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., & Huang, Y. (2010). T-drive: driving directions based on taxi trajectories. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. San Jose, California: ACM.

22. Yuan, J., Zheng, Y., Zhang, L., Xie, X., & Sun, G. (2011, September). Where to find my next passenger. In *Proceedings of the 13th international conference on Ubiquitous computing* (pp. 109-118). ACM.

23. Yuan, N. J., Zheng, Y., & Xie, X. (2012). Segmentation of urban areas using road networks. *Technical Report*.

24. Zhu, Y., Zheng, Y., Zhang, L., Santani, D., & Xie, X. (2011). Inferring taxi status using GPS trajectories. *ArXiv*. http://doi.org/MSR-TR-2011-144