Efficient Storage of Big-Data for GPS Applications

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Technology

IN

Computational Science

by

Pavan Kumar Akulakrishna

Supercomputer Education and Research Centre

Indian Institute of Science

BANGALORE – 560 012

JUNE 2014

Dedicated to my Mom, Dad and Grand Parents

Acknowledgements

Gratitude is a beautiful human feeling, but it finds fulfilment only in expression. And when you express it, you enrich the life of not just the person whom you are expressing it to, but your own life more so. So I would take this opportunity to express gratitude to some people, who have made a difference in my life and in this work as a result.

Firstly, I would thank the Almighty for keeping me amongst kind and beautiful people. I would also thank my parents who have trusted me and always have supported me in various stages of my career.

I take this opportunity to express my profound gratitude and deep regards to my guides **Prof. S.K.Nandy** and **Dr. J.Lakshmi** for their exemplary guidance, monitoring and constant encouragement throughout the course of this Masters project. The gracious, help and guidance given by them time to time shall carry me a long way in the journey of life on which I am about to embark.

I am always inspired by humbleness and patience my guide Prof.S.K.Nandy has and I would cherish this inspiration for lifetime. A special thanks to Lakshmi Mam, for motivation, friendliness and support which is overwhelming and cannot be expressed in words.

I also take this opportunity to express a deep sense of gratitude to all the faculty of SERC, who have guided me through the course work, for their cordial support, valuable information and guidance, which helped me in completing my masters through various stages.

I am obliged to staff members (Ms.Mallika, Mr.Shekar) of SERC, for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my masters programme.

I would also like to specially thank my grandparents (A.Gopal and A.Shankuntala Bai) for their blessings and well wishes on me.

I thank my siblings (Sai, Sushma, Maanasa), lab-mates(Aakriti, Nitisha, Geetha Venkatesh), batch-mates (Pavan Badarla, Jagvir, Aniruddha, Jaganath, Jayashimha, Aditya, Aashish) for their constant encouragement without which this project would not be possible. I thank my undergraduate friends (Suman, Sidhartha) in Bangalore who also encouraged and supported me. I would also thank Nehru Bal Sangh, Bangalore, which also played an important role in my life.

I thank institution's management, hostel office, mess management, for creating a great place to stay in the institute. Last but not the least, I would also thank Indian Institute of Science (famously Tata Institute) founder, J.N.Tata, for establishing such an institute which has great ambience and minds.

Abstract

Increase in smart mobiles has resulted in popularity of navigation applications like navigator and maps. These applications can potentially use large amount of data for a variety of applications that are associated with route information. The nature and format of the data collected depends on the sensing method used, of which GPS is currently the most popular. As of now most of these applications use static data with current location. However, interesting applications are emerging with time dependent updates enabled on the available Geo-spatial data. One of them is real time traffic monitoring which performs analysis of spatial and time dependent measurement of data. The goal of these systems is to provide information such as average speeds, volumes and densities on a given segment of a roadway. There is large demand in industry and by transportation agencies to have access to high resolution state of traffic on highways and arterial roads globally. GPS applications need real-time responsiveness and are location-sensitive. GPS data is time-variant, dynamic and large. Current methods of centralized or distributed storage with static data impose constraints on addressing the real-time requirement of such applications.

In this project we explore the need for real-timeliness of location based applications and evolve a methodology of storage mechanism for the GPS application's data so as to meet the deadlines posed by the current needs of the day-to-day life in regard to the live-traffic updates. So far, the data is distributed based on zones and also it has limited redundancy leading to non-availability in case of failures. In our approach, data is partitioned into cells giving priority to Geo-spatial location since GPS applications are Geo-location-sensitive. The key feature of this method is to adopt distribution of geo-spatial data in such a way that majority of location based queries can be answered within a cell. The geography of an area like a district, state,

country or for that matter the whole world is divided into data cells. The size of the data cells is decided based on the previously observed location specific queries on the area. The cell size is so selected that a majority of the queries are addressed within the cell itself. This enables computation to happen closer to data location. As a result, data communication overheads are eliminated. Apart from being location sensitive, we also build some data redundancy while distributing the data across cells. We use a nine-cell approach wherein each cell stores data of eight of its neighbours along with its own data. This redundancy is used not only to enable failover mechanisms but also to target performance. Cells that have an overload of queries, can easily pass-off some of their workload to their near neighbours and ensure timeliness in response. Further, since data distribution to cells is done based on the query density on the area, we achieve effective load balancing of data and ensure better utilization of resources. Experimental results show that our approach improves query response times, yields better throughput and reduces average query waiting time apart from enabling real-time updates on data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 GPS Applications

GPS applications like finding current location, finding a route from current location to some destination, finding nearest police station or hospital or restaurants etc., are becoming trendy and useful to the society in day-to-day life [8]. Also various classes of people use GPS-applications for different reasons. Scientists use GPS to track some dangerous species either in water or some other locations where human reach is not prevalent. Similarly, farmers use them for dimension tracking of their fields and drivers use it to get directions to various locations. From Google Maps to consumer Global Positioning System (GPS) devices, society has benefited immensely from routing services and technology. Also, there is a need for these applications to be more responsive i.e., the applications should provide real-time information. However, the issue is that these applications need to normally handle large amount of data from many sources. Hence better computing and storage mechanisms need to be explored to enable such applications. The issues concerning the real-time GPS applications are, when a driver uses the GPS to track the route to some destination, if at all there is congestion in some road network at that moment, then the application should provide information to driver well before so that he can avoid getting into the congestion and possibly take an alternative path.

1.2 Motivation

One study estimates that personal location data could save consumers worldwide more than $600 billion annually by 2020 [8]. Computers determine users' whereabouts by tracking their mobile devices, like cell-phones GPS etc., The study cites smart-phone location services including Foursquare, loopt etc., for locating friends, and ones for finding nearby stores and restaurants. The biggest single consumer benefit, the study says, is going to come from time saving and fuel savings from location-based services- via tapping into real-time traffic and weather data - that help drivers avoid congestion and suggest alternative routes. "New ways to Exploit Raw Data may bring surge of innovation", a study says [8]. Such above insights from various studies motivate Spatial Big Datas (SBDs) to become an area of interest to many researchers con-temporarily.

1.3 GPS Applications in Context of Storage

There is a specific big-data solution, for storage and distribution of data for a given problem [2]. In this thesis we look specifically at the GPS based data and its usage to integrate real-time data updates regarding Geo-spatial networks from the perspective of data distribution and storage such that applications can be built for real-time responsiveness.

In most of these applications, the data is either located at a centralized place or is distributed over multiple nodes. The application-level performance of fully distributed and centralized data dissemination approaches in the context of traffic advisory systems are challenging. Currently, there are many distributed storage mechanisms that are specific to applications. In the case of Geo-spatial applications most of the data distribution strategies are based on the one time computation or static nature of the query response. Extending such algorithms to support time-sensitive decision making is a challenge. Not only the limited time response but also accurate response is what is intended. Hence, an efficient mechanism that is not only intuitive

but also effective should be implemented to reach the Geo-spatial application deadlines. Generally, a query in real-time is of the form route from A to B with special parameters. Hence, distributed storage gives chance to parallelism and so speed up is achieved. In order to achieve the real-time update we also need to have efficient mechanisms to store, manage and distribute data. In this thesis, we would address the problem of how to get a real-time responsive solution for the users' queries with context to storage mechanism. The storage mechanism is critical in the computation of route from one location to another.The basic idea is to organize data in a distributed way and always identify and associate the computation on the data to where it is located. This allows for exploiting the concurrent nature of the computation and thus contributes to reduction in time for responding to the query.

The key contributions in this work include:

- Designing of a storage mechanism that exploits redundancy, to reduce the communication overheads and fail-over of nodes.
- The computation is brought closer to the data to reduce the latencies and communication overheads.
- The data load is balanced over the nodes with some relaxation to ensure better utilization of resources and make sure that load balancing is not a frequent operation.

1.4 Thesis Organization

The aim of this thesis is to study the various existing storage mechanisms for GPS applications and device a storage mechanism based on heuristics which enhances performance of GPS applications, which have large data. This thesis proposes a storage mechanism named as *NineCellGrid*, which is a spatial overlap of data with a decomposable unit as a cell. This cell dimensions are computationally found to target performance. Also, in this thesis the load-balancing algorithms are proposed to effectively utilize the resources. Finally in this thesis, a comparative study on performance of *NineCellGrid* with existing storage mechanisms is done

to evaluate the proposed *NineCellGrid* method. The rest of the thesis is organized as follows:

Chapter 2 describes the related work to highlight the efforts published in literature in this area. It depicts the various storage paradigms in GPS applications, such as centralized and fully distributed data disseminations. This chapter also highlights the drawbacks and necessities for these GPS applications.

Chapter 3 details the *NineCellGrid* methodology proposed for storing the data and describes how computations in routing query can be generated using this mechanism. In this chapter, the heuristics behind the storage and computational analysis to do so are explained. A case study on the evaluation of various scenarios is also explained in this chapter. This chapter also explains the load balancing of the data with some relaxation.

Chapter 4 explains the experimental setup and observations achieved so far. In this chapter, the illustration of data-load balancing, computationally evaluating the optimal value of a heuristic (L), and finally comparison of different storage mechanisms, like centralized pattern, fully distributed pattern and *NineCellGrid* method, is done in terms of minimizing turnaround times and maximizing throughput, are studied.

Chapter 5 presents the conclusions of the thesis. It provides major contributions of this work and major gains obtained by the strategies in storage mechanisms discussed. It also draws conclusions from previous chapter in regard to the performance.

Chapter 6 elucidates the scope for future work. It gives some future directions where this work could be extended and some untouched aspects in this proposed storage mechanism.

1.5 Summary

In this chapter, the types of GPS applications and their necessity to the society are introduced. This chapter also provides the motivation towards this problem of 'Efficient storage of big-data for real-time GPS applications'. And then formulates the GPS applications in context of storage and distribution of data. This chapter also highlights the key contributions in this work. And then briefs the outline of this thesis.

Chapter 2

Related Work

Data storage and dissemination is done in either a fully distributed manner or in a centralized manner [2]. In centralized approach, applications depend on the road-side infrastructure to connect to centralized location. And the data that reaches centralized location comes from the vehicles via the intermediate nodes called beacons [2]. Some organizations have offices at multiple levels. For example, a company can have offices at the national level, state level, and city level. Data distribution allows each office to locally manage the data applicable to its area and also share with the levels above and below [12].

2.1 Centralized Data Dissemination

In this type of data storage and dissemination, the data resides on the centralized system. In centralized approach, vehicles depend on the road-side infrastructure to connect to centralized location. And the data that reaches centralized location comes from the vehicles via the intermediate nodes called beacons [2].The data is queried from this system and computed on various nodes. With a centralized approach, vehicles rely on road-side infrastructure, either in a planned [9], [10] or in an opportunistic manner [11], to communicate with a central location. ESRI Geo-databases are a relatively new format. Geodatabases are databases stored in Microsoft Access (for the ”personal” geo-database), as a special collection of files (for the ”file-based” geo-database), or higher-end applications (e.g. SQL Server, Oracle, Informix).

A geo-database stores all features and related tables, as well as other files, within a single or distributed database format [12]. These databases deal with information that is accumulated periodically and mostly used for read-only purposes like searching for results on route queries. The answer to a given query may not change with time because the time dependent information is not collected or stored.

2.2 Distributed Data Dissemination

Several different data distribution techniques are available. Deciding which to use involves considering the requirements of your system as well as the benefits and limitations of each technique [12]. In some cases, more than one technique may be used to meet the system requirements. The following describes each technique:

2.2.1 Geo-database Replication

Geo-database replication: Geo-database replication allows you to distribute data across two or more Geo-databases such that edits can be made independently and synchronized periodically. It has built-in safeguards against data loss, data redundancy, and system instability. Geo-database replication requires at least one versioned ArcSDE Geo-database [12].

2.2.2 DBMS replication

DBMS replication: ArcSDE Geo-databases are built on top of DBMSs that include technology for replicating at the database level. Geo-databases, like other applications built on top of these DBMSs, can be used with this technology. Using DBMS replication with Geo-databases requires knowledge of how Geo-database data structures are implemented at the database level. ArcGIS does not provide out-of-the-box tools for implementing these systems like it does for Geo-database replication [12].

2.2.3 Data Copying and loading tools

Data copying and loading tools: Another technique for distributing data involves simply copying data from one Geo-database to another. This technique is useful for two systems with simple requirements. For example, a fieldworker updates a feature class and needs to copy that feature class to the ArcSDE Geo-database in the office each night. This technique can also be used where the data is non-versioned or where only personal or file Geo-databases are involved. However, it has no built-in safeguards against data loss or data redundancy [12].

2.3 Issues and Requirments

These above techniques are used in data replication to achieve data redundancy and fault-tolerance to some extent. Above storage mechanisms work for a scenario wherein the data is limited, structured, and not large. As the scenario of the current GPS applications demand large data so as to answer the queries in realtime. Hence, a scalable, highly concurrent, low latent and fault tolerant solution is necessary to be addressed.

2.4 Summary

In this chapter, the related work is presented to highlight the efforts published in literature in this area. It depicts the various storage paradigms in GPS applications, such as centralized and fully distributed data disseminations. In centralized storage pattern the there is a dedicated central server that handles the static GPS data. And in fully distributed dissemination there are three types of patterns namely, 'Geo-database Replication', 'DBMS replication' and 'Data Copying and loading tools'. This chapter also highlights the drawbacks and necessities for these GPS applications in regard to real-timeliness and data being large.

Chapter 3

NineCellGrid Method

In our approach we address the requirement of storing time-based updates on the geo-spatial data. We choose the distributed dissemination method and use the *NineCellGrid* method for storing data. In this method, data is distributed not based on the available storage nodes, but based on the region of area on which the computation is intended.

3.1 Design Approach and Strategies

- Computation-closeness to the storage node is ensured, using Geo-spatially localized distributed data storage pattern (*NineCellGrid Storage Methodology*)
- Data-redundancy is used to reduce communications, improve performance to respond for real-timeliness and also build in fault tolerance.

3.2 Definition of Cell

The GPS data that is concerned to a location is decomposed in such a way that a region on earth is mapped with a mesh/grid of cell dimension L×L, where L is some fixed value (explained in the following sub-section 3.4). Each cell is representing a unique L×L area on earth as in Figure - 3.1. Each cell not only represents the data of the region associated with it but also represents the computation of that region. Hence, the region is both data node and compute

node for that region. So, this way the computation is done at the region where the data is located, that is 'Computation-closeness' to the storage node is ensured (as mentioned above).

Figure 3.1: Grid on Geographic Map [1]

In this approach we propose that each cell can hold the data of 8 other cells that are present around it. That is, here, apart from its own cell's data it stores an extra 8 cells' data that surrounds it as shown in Figure - 3.2. So, the query that is to be answered by a cell can now be answered by all the 9-cells that contain that data.

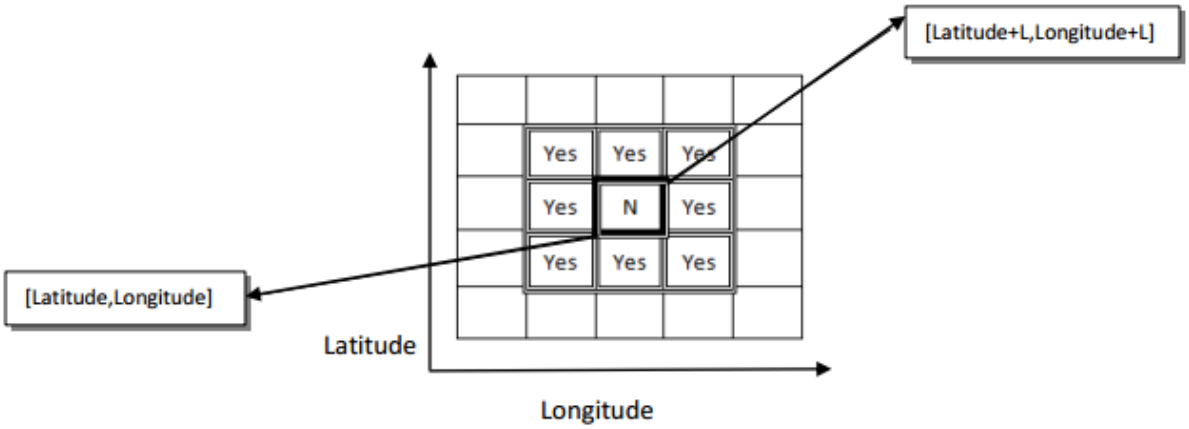


Figure 3.2: Cell N, with identity [Latitude, Longitude] in NameTable

3.3 NameTable

A table called *NameTable* (Table - 3.1) that is indexed with row key as the 'Latitude' and column key as the 'Longitude' which gives the address of the node which is responsible for that cell or region in reality of dimensions L×L. The actual region that is covered by the node is a square (not necessarily) of four vertices mentioned as [(Latitude, Longitude),(Latitude + L, Longitude),(Latitude, Longitude + L),(Latitude + L, Longitude + L)]. This table gets updated as and when the region is reassigned to a different node. The load at each node gets redistributed inorder to balance the load. The *NameTable* is automatically updated with these changes in the load balancing stage. This table information is more important and sensitive since, without which the data cannot be located and correlated. Hence, this table is stored more securely. That is, it can be stored with high redundancy or in secure systems. Moreover, *NameTable* spans least space on disk since, it is number of nodes times the size of *NodeAddress*. For instance consider the number of nodes are 10,000 and IP addresses are stored then 10 K × 4Bytes, which is 40Kb. Hence, this *NameTable* is small in size but highly important and sensitive information.

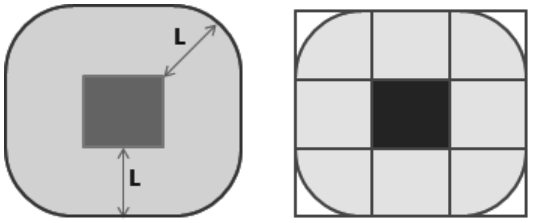
		Longitude				
		-74.499998	-74.499998 + f(L)	-74.499998 + 2*f(L)	...	-73.500016
Latitude	40.300009	<*NodeAddress*>	...	..	...	
	40.300009 + g(L)	<*NodeAddress*>	...	..	...	
	40.300009 + 2*g(L)	<*NodeAddress*>	...	..	...	
	...	<*NodeAddress*>	...	..	...	
	41.299997	<*NodeAddress*>	...	..	...	

Table 3.1: **NameTable** Here f and g are mappings from miles to Longitude and Latitude units

3.4 Dimension of cell (L)

The cell is of length L, where L is the standard units of Longitude and Latitude. Then, L sized cell is mapped to a node. Hence, each cell/region's data is present in exactly nine nodes and

to locate those nodes in the *NameTable* is just the (-L, 0, +L) combinations to the keys, hence nine possibilities. The reason for choosing nine cell's data within one cell is to make sure that any 'A-to-B route' query of Euclidean distance $\leq$ 'L' can be addressed by a single cell. And this L is chosen in probabilistic manner considering the previously observed queries.



(a) Locus of B with A inside a cell

(b) Best-Fit, *NineCell-Grid*

Figure 3.3: *NineCellGrid*

Consider, all the possible locations of source A, in query "A-to-B", within a cell it would be filled square or cell. Then, the locus of point B such that B is not more than L Euclidean distance away from A is a rounded rectangle (rounded square) with corners as a quarter circle of radius L as shown in Figure - 3.3(a). In Figure - 3.3(a) the filled square represents the possible locations of A in a cell. And the filled outer rounded rectangle (rounded square) represents the locus of B such that A is in that cell. To accommodate all the data within a single cell, the best fit would be this Nine Cell Grid shown in Figure - 3.3(b). That is, all the surrounding 8 cells data to be located in this central cell so that it has the complete coverage of B in any case without having to communicate with neighbouring cells/nodes.

The data of the cell is stored in distributed manner across the 9 nodes using the HDFS (Hadoop Distributed FileSystem) [15]. It is also noted that specific type of data which supports reduce operations are stored in HDFS format only, else they follow the mechanism of complete data copies i.e., entire file is stored as copies in all the 9 cells. That is, unlike HDFS, the data is stored completely in all the 9 nodes without breaking into chunks. This is because not all the data can be processed using the reduce operations.

A general query that is processed in these GPS-applications is to route from location A to location B. Given a query to route from location A to location B, our L value is chosen such that any query made by the traveller using this application has the A-B Euclidean distance

not more than L. Or the probability of a query that exceeds the L Euclidean distance is very less say (for instance), less than 0.2, which means queries with Euclidean distance less than L occur with a probability of 0.8. Inorder to achieve this criterion, L is chosen as

L = min $\{$ *X : Euclidean distance of P% of queries are* $\leq$ *Euclidean distance of X* $\}$

Where, P can be any percentage that gives better performance. It is seen that on considering about $\sim$ 80% of the previously observed queries on the dataset used, it shows better performance (shown in results section). Interestingly, if the percentage is considered high, that is something close to 100% will not help. The reason being, if percentage is high then it tends towards centralized storage pattern. On the other hand, if the percentage considered is very low then it tends to a scenario of high communication for most of the queries. Hence, there exists an optimal percentage to be considered to arrive at L. Hence, on checking the performance of percentages from 70% in steps of $3-5\%$ it is found that around $\sim$80% suits better in the dataset taken (shown in experiments & results section). These experiments are done taking 5% as step since very close percentages outperform each other. Hence, a higher resolution step does not help much. Finally, L is defined globally.

3.5 Limiting Vertices

The edges of the graph that are cutting the grid have corresponding vertices, these are called the limiting vertices. These limiting vertices store the additional information of the edge information to another node, and node address. Whenever, there is a need for it to go towards that edge traversal the information on this vertex conveys so. And the data from that node to this node is communicated to get the increased coverage of map to compute the query. In actual the limiting vertex rather stores the latitude and longitude coordinates of the cell to which its corresponding limiting vertex resides. And then, the 'Latitude' and 'Longitude' are used as keys to reference *NameTable* to get the corresponding address of the node. 'Latitude' and 'Longitude' are stored since the node addresses for that region keeps changing as and when the redistribution is done.

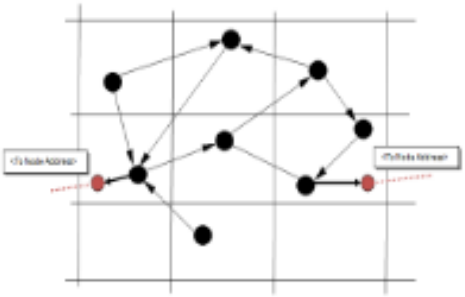


Figure 3.4: Limiting Vertices

3.6 Case Study of 'A-to-B route' Query

Different scenarios of an A-B query are illustrated as follows:

3.6.1 Case 1: Within a Cell

Case 1: Both A and B lie in the same cell of L×L (occurs with 0.8 probability assumed, hence L is chosen)

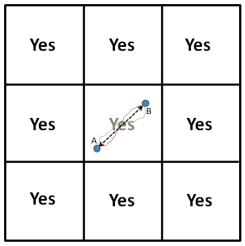


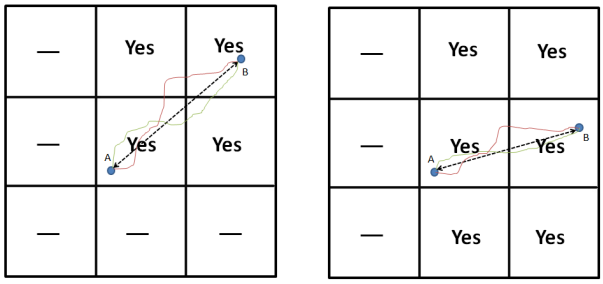
Figure 3.5: Case 1: where A and B lie in same cell

Now that both A and B being in the same Cell, this query can be answered by 9 Nodes. Also, if there is some congestion on certain node that node is not given the task and rest nodes are assigned this job of this query from A to B. This is the task of the job-trackers to assign the job to the relevant node with less task load. Hence, flexibility on assignment of jobs is also a boon to this type of distributed storage. In this case, the central node can be given this task/job. Since it covers symmetrically much region. However, if the central node already has high load

then this task is assigned to a different node among the nine nodes. The next preference is given to that cell whose sum of Euclidean distances from center of the cell to source A and destination B is least. Job-scheduling is done this way to ensure better task load balance and to yield better throughput.

3.6.2 Case 2: 1-Cell Apart

Case 2: A and B lie at 1 Cell distance apart



(a) 1-Cell Diagonal (b) 1-Cell Non-diagonal

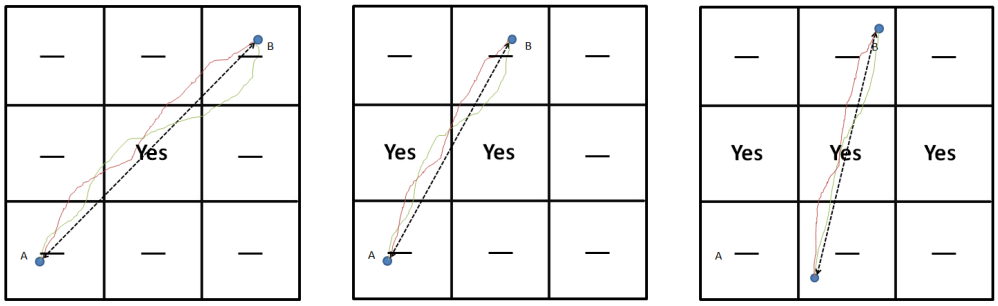
Figure 3.6: Case 2: When A and B are 1-cell apart

Here, the possible number of nodes that can compute the query are 4-6 nodes. Figure - 3.6 shows the scenario of 4 nodes. Ofcourse, this has a relatively lower flexibility than the previous case. But the query can be simultaneously processed by any of the 4-6 nodes.In this case the priority is given to the node with least sum of Euclidean distances from center of this node to both source A and destination B. However, based on the task load the node, which has next least sum of Euclidean distances from center of the node to A and B, is considered and so on.

3.6.3 Case 3: 2-Cell Apart

Case 3: A and B lie at 2 Cell distance apart

In this case, the possible number of nodes that can compute the query are 1-3 nodes. Figure



(a) 2-Cell Diagonal (b) 2-Cell Semi-diagonal (c) 2-Cell Non-diagonal

Figure 3.7: Case 3: When A and B are 2-cell apart

- 3.7 shows the scenario of 1 node. It has lesser flexibility than the previous cases. But, this case happens with the probability of less than 0.2 since our L is chosen such a way. Similarly, in this case the priority is given to the node with least sum of Euclidean distances from center of this node to both source A and destination B. However, based on the task load the node, which has next least sum of Euclidean distances from center of the node to A and B, is considered and so on.

3.6.4 Case 4: Greater than 2-Cell Apart

Case 4: A and B lie at a distance of more than 2 cell distance apart

In this case, the communication is done with neighbouring nodes to get the updated data and then process the query. Or this node may also send the job state to the neighbour node whose data is needed and the process will run on that neighbouring node.

Inorder to get complete data of the region, a node has to communicate with atmost 11.11% of the total nodes only. Hence, reducing latency to a greater extent. That is, for instance consider a 9×9 region with central node in this grid wants the entire regions data. Then inorder to get the entire region it needs to communicate with 8 other nodes only. Among them four are two cells apart in east, west, north and south directions. Other four are two cell distances apart in diagonal directions. That is, in north-east, north-west, south-east and south-west directions. Among 81 cells (9×9 has 81 cells) only 9 cells communicate with each other to get entire

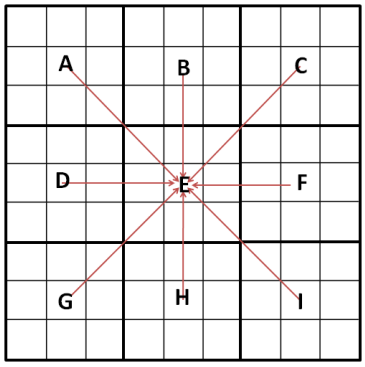


Figure 3.8: Case 4: When A and B are more than 2-cell apart

data of the region. Hence, $1/9^{th}$ of the nodes communicate which is 11.11% of total nodes involving in communication.

3.7 Load Balancing

After arriving at the value of L, there is the problem of load-imbalance across the nodes, due to the natural state of earth. That is, due to water bodies there can be unnecessary assignment of the part of region less intensive of links/roads, updates, etc.,. Hence, that node is assigned relatively more L×L cells. The load balancing is done to distribute the data across different nodes to handle query density on the region. While assigning another cell to it inorder to load-balance, it may so happen that it is assigned with the a cell that is going to overlap with the cells already present with this cell. Then, only those cells' data is stored which do not fall in common.

Whenever the load-factor is less, the node is expanded (i.e., larger region is covered by assigning more L×L cells) and when it is high, it is contracted. However, this expansion and contraction occurs very rarely. That is, whenever there is fast growing city or rapid growth in road network then it may lead to expansion and loss of road network due to some natural calamity or some artificial destruction lead to contraction. But, these are not that frequent hence, the shifting of data from one node to another is rare.

Algorithm: ***Load Balancing***
Inputs: [*NameTable*; Relaxation δ]
Outputs: [Load balanced distribution; Updated *NameTable*]
1. Initialize load based on number of edges, vertices, etc.,
2. **for** each node *E* **do**:
3. **if**(Load(E) $< M - \delta$)
4. Find node P closest to E with load $\geq M - \delta$.
5. Load(P) $\rightarrow$ Load(P) + Load(E);
6. Load(E) $\rightarrow$ 0;
7. NameTable[Latitude(E)][Longitude(E)] = *Address(P)*.
8. **endif**
9. **endfor**
10.**for** each node *E* **do**:
11. **if**(Load(E) $\neq 0$)
12. Share the load among N nodes.
13. $(Load(E)/N) \in [M - \delta, M + \delta]$ & N $\in$ (1,2,3,...)
14. Update NameTable.
15. **endif**
16. **endfor**

Table 3.2: *Load Balancing*

In this case load is not strictly balanced but given a relaxation δ. Initially the load is estimated to each L$\times$L cell considering number of junctions/vertices, roadlinks/edges, landmarks, etc.,. Mean of estimated load across cells, called as M, is calculated. The load is balanced with relaxation δ i.e., a cell is said to be balanced if its load lies in $[M - \delta, M + \delta]$. Relaxation depends on logistics like resources availability, rate of data increase etc.,. Now, each cell with load less than M-δ is set to zero after the load is transferred to a node that is closest to it and also having load value more than M-δ. After setting these cells to zero, the cells with either load equal to zero or load value greater than M-δ. Those cells with load value greater than M-δ are shared by N nodes such that after dividing the load value with N the resulting value lies in $[M - \delta, M + \delta]$. Hence, the utilization of the resources is improved this way. The actual placement of the physical nodes is done after load balancing. That is, all the nodes handling the same region will be placed together so as to reduce latency and overheads due to communication.

3.8 *NineCellGrid* Methodology

In this section the overall methodology is summarized starting from how the L value is obtained (Figure - 3.9).

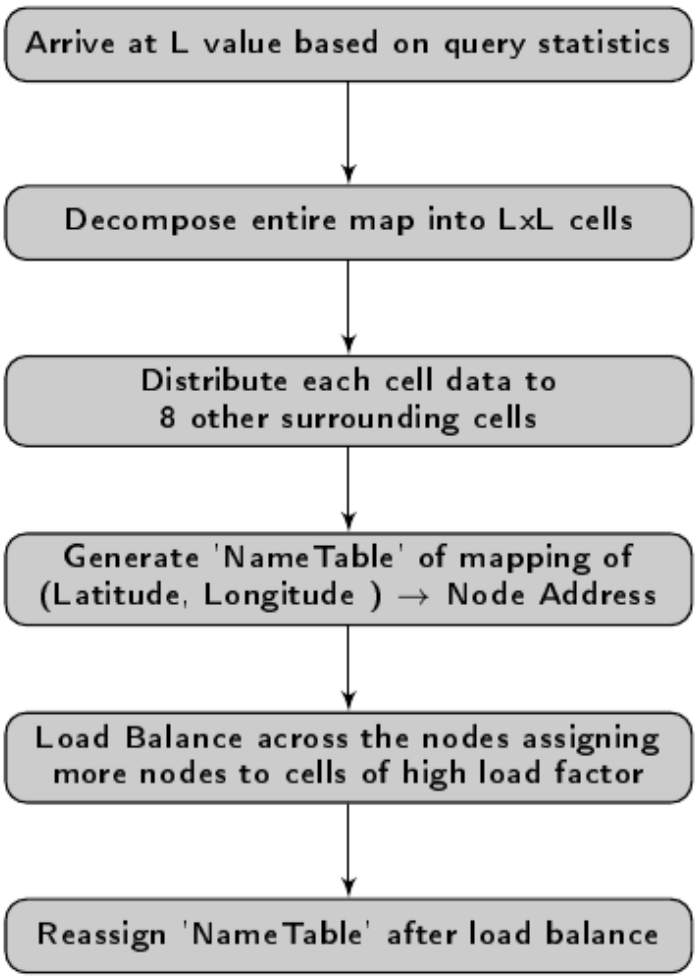


Figure 3.9: *NineCellGrid Methodology*

Initially the L value is decided using the previously observed queries list. Initially percentage 'P' is guessed as 70% (i.e., the corresponding L is assigned accordingly) and then in steps of 3-5% it is increased or decreased to arrive at optimal percentage or optimal L value. After arriving at L value the region decomposed into $L \times L$ cells and the copies of eight surrounding

cells are made. Then the *NameTable* is generated to map (Latitude, Longitude) $\rightarrow$ 'NodeAddress'. Now, load balancing is done to effectively distribute the data load across the cells as per the Algorithm in 3.7.

3.9 Summary

In this chapter, the storage mechanism based on a heuristic (L) is proposed to target performance. This chapter starts with description of design and strategy details. And then definition of a cell and dimension of cell, L is illustrated. And then, description of *NameTable* associated with the cell is detailed. A case study on the evaluation of various scenarios is also explained in this chapter. This chapter also explains the load balancing of the data with some relaxation.

Chapter 4

Experiments and Results

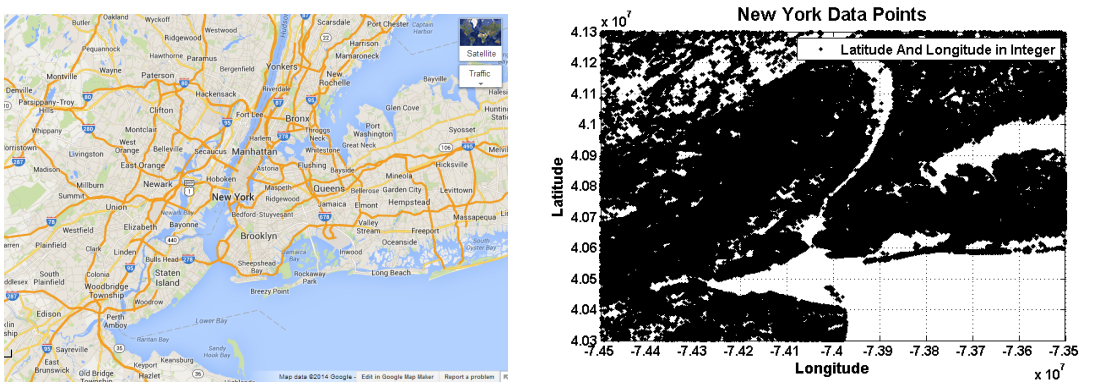
In this chapter, various experiments and illustrations are shown. The results comparing the throughputs and turnaround times of *NineCellGrid*, fully distributed patterns and centralized patterns, load balancing pattern using algorithm in previous chapter, and also the finding the optimal L value in steps of 5% starting from 70% are illustrated and explained.

4.1 Experimental Setup

We used MPI (Message Passing Interface) for simulation and Dijkstra's shortest path algorithm for route computation to study the application-level performance of centralized, fully distributed and our Geo-spatially localized distributed storage (*NineCellGrid* approach) data dissemination models. The implementation of *NineCellGrid* using MPI is done by assigning a process, one cell's data (after the neighbouring eight cells' data being copied and load balanced using the Algorithm - 1) and computation. Each process represents a cell in *NineCellGrid* context. Dataset 4.1 used for experiments is 'New York City' with co-ordinates [(40.3, 41.3),(73.5,74.5)] (Latitude,Longitude) [14]. In this dataset number of vertices/junctions are 264,346 and number of links/edges are 733,846. Figure - 4.1(b) shows the actual coordinates of the dataset used.

Dataset used	New York City
Latitude Range	(40.3, 41.3)
Longitude Range	(73.5, 74.5)
Vertices/Junctions	264,346
Arcs/Links	733,846
Source	DIMACS [14]

Table 4.1: Dataset: 'New York City'



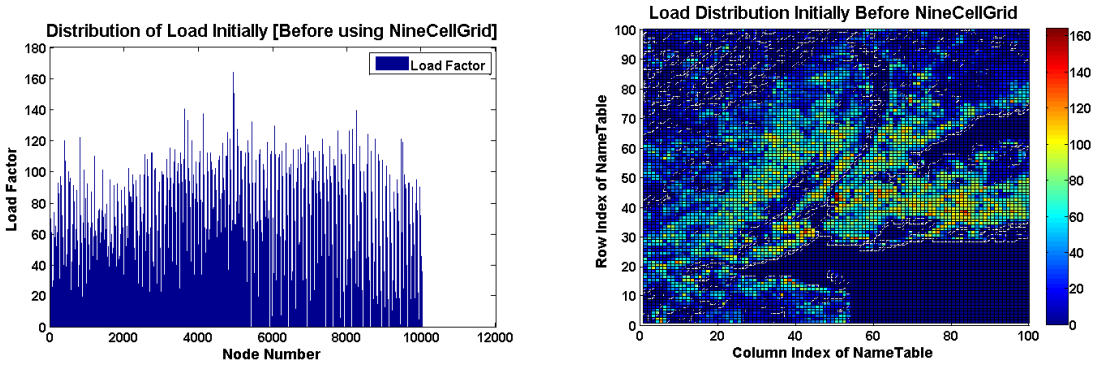
(a) Actual Geographic Map [1] (b) New York City dataset actual coordinates [14]

Figure 4.1: Dataset New York City

4.2 Load Balancing Illustration

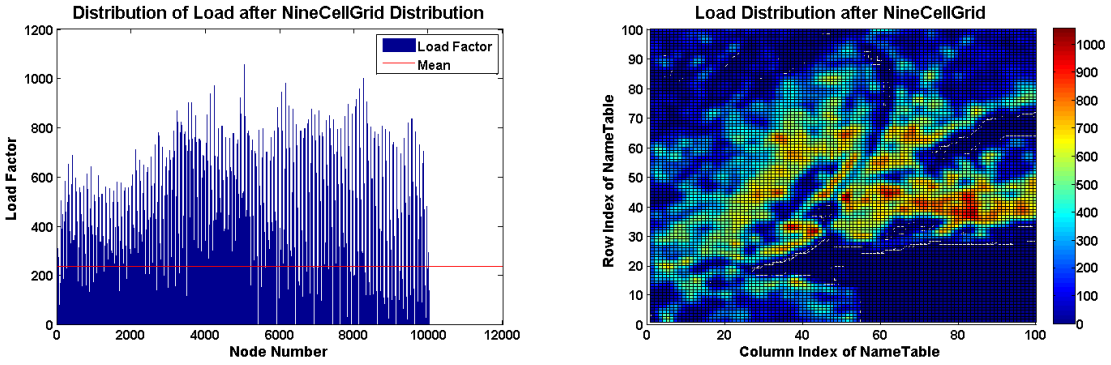
The load estimation of a node is done considering the number of junctions/vertices, links/edges in the region assigned to it. The value of the node's load is shown as the sum of all these values. Figures - 4.2, 4.3 and 4.4 show the load distribution of the nodes before replication, after replication and finally after the load-balancing respectively. These figures show the load in color based on the value of load a node or cell has. In this dataset the region with waters have low value of estimated load. Hence, are shown with low color value in the scale. The cells with higher load have higher color value in the scale i.e., towards red. After the load balancing (Figure - 4.4) the regions with less than $M - \delta$ load values are assigned to nearest cell with load value greater than $M - \delta$, hence, the regions with lower load values are vanished (i.e., are shown as zero, low scale value dark blue). And then the load is shared among N nodes such that 'load per node' is balanced. Hence, there is relatively lower load range, i.e., there is

no much varied color values. Here, the relaxation used is $\delta = 75$. So, the load values in the Figure - 4.4 would be either between 150 to 300 as the M is around 225 or zero.



(a) Bar distribution (b) Geographic View

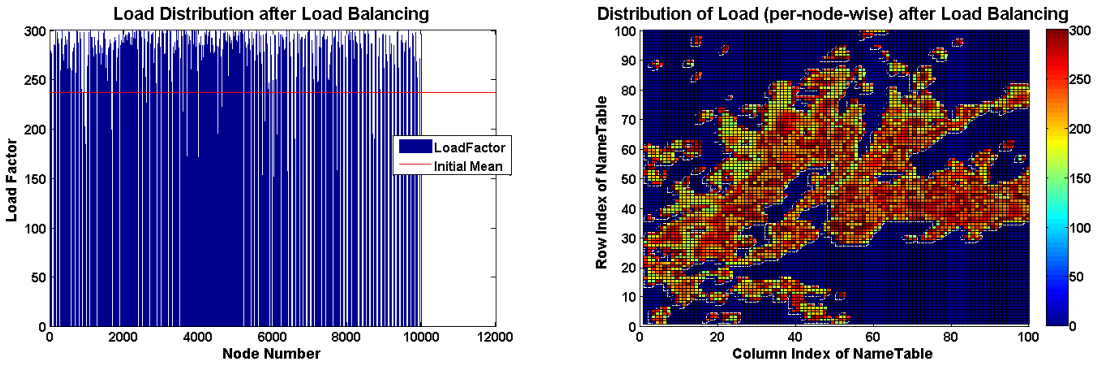
Figure 4.2: Load distribution before replication



(a) Bar distribution (b) Geographic View

Figure 4.3: Load distribution after replication

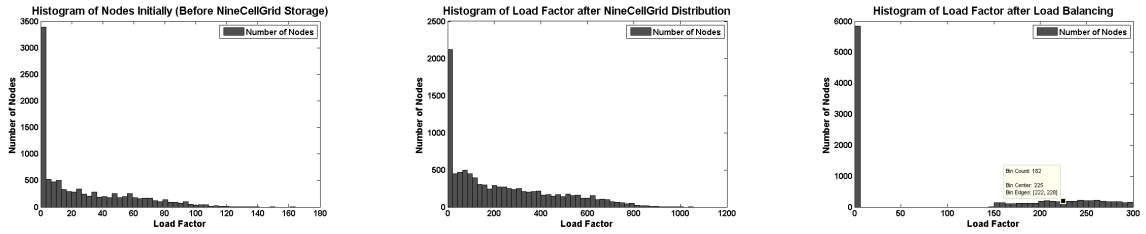
Figures - 4.5(a), 4.5(b) and 4.5(c) show the histograms of load distribution of the nodes before replication, after replication and finally after the load-balancing respectively. On comparing Figures - 4.5(a) and Figure - 4.5(b) it is observed that the maximum value of the load in Figure - 4.5(a) is one-ninth that of the Figure - 4.5(b) this is due to the fact that replication is done almost nine times. That is, each cell's data is stored in 8 other surrounding cells. Finally in Figure - 4.5(c) the scenario after load balancing shown with a relaxation of 75 i.e., $\delta = 75$. After balancing in Figure - 4.5(c) the nodes get into either the region of $[M - \delta, M + \delta]$ or to



(a) Bar distribution (b) Geographic View

Figure 4.4: Load distribution after load-balancing

the zero load value bin. That is, the load values are either between 150 and 300, with mean $M = 225$, or zero. All those zero load valued cells represent the cells with initial load value $< M - \delta$.



(a) Before replication in *NineCell-Grid* (b) Before load-balancing (c) After load-balancing

Figure 4.5: Histograms of Load

4.3 Finding Optimal L Computationally

The selection of percentage, which determines the L value, is done based on the analysis as shown in the Figure - 4.6. Initially taking 70% as the initial guess it is increased in steps of 5% so as to get to the optimal percentage. The graph shows 80% as optimal for this scenario. The percentage may not be the same in a different region as it depends on the city, country, etc.,. But there exists an optimal percentage of queries less than L to evaluate L. As mentioned in the earlier section the more the percentage tends to centralized storage pattern and leads to

less speed up. And on the other hand if less percentage it leads to higher communication and hence degrades performance.

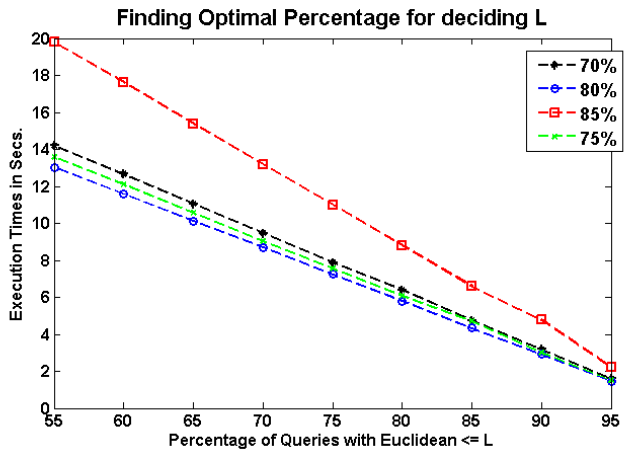


Figure 4.6: Execution times vs. Percentages of queries Euclidean(A-B) less than L

4.4 Performance Comparisons

The performances are analysed running a Dijkstra's greedy algorithm for shortest path computation [16]. Here, MPI is used to simulate the scenario. L value for the dataset New York City is taken as 6.8 miles (considering $\sim 80\%$) from NHTS (National Household Travel Survey) [13].

CGWR	"Cell Grid Without Replication"
DBMSAR	DBMS as Replication factor, RF = 5
Zonal	Fully distributed with RF = 5
Centralized	Centralized distribution.

Table 4.2: Notations used

4.4.1 Throughput Comparison

Figure - 4.7, shows the comparison of throughputs for various storage methodologies. In figure, the *NineCellGrid*, '*Zonal*' representing fully distributed, 'Centralized Storage' pattern,

'CGWR' representing *'NineCellGrid Without Replication'*, and finally 'DBMSAR' representing the DBMS as replication are compared in context of the throughputs they yield (replication value of 5 taken). Number of queries are 1000. Graph in Figure - 4.7 shows 'Percentage of queries with Euclidean distance less than L' vs. 'Throughput in Queries/Sec'. The throughput of *NineCellGrid* starts dominating other standard methods from 70% onwards. On comparing the throughput of a general scenario of above 80% expectancy, it is seen that *NineCellGrid* shoots a better throughput values than the others. The difference between the Fully distributed(zonal distribution of data) and *NineCellGrid* is about 470 (810-340). Hence, it can process about 300 more queries in a second.

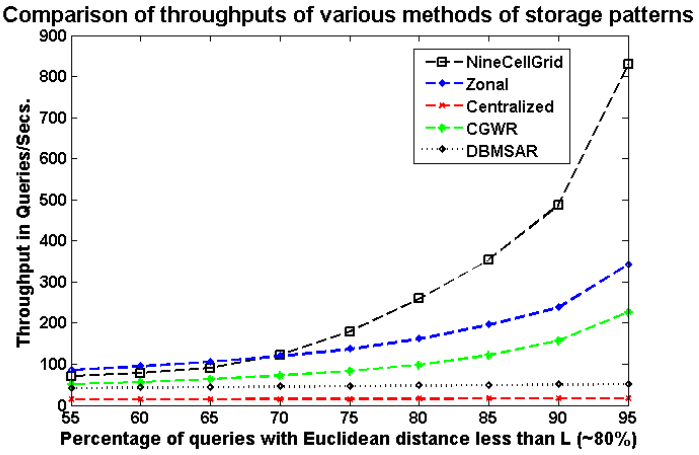


Figure 4.7: Throughput Comparison of various storage methodologies

4.4.2 Turnaround Time Comparison

Figure - 4.8, shows the 'Percentage of queries with Euclidean distance less than L' versus 'Average time per query taken in seconds'. It is observed that the *NineCellGrid* storage method has least 'Average time per query' after about 70% onwards. Fully distributed has better 'Average time per query' before 70% because of the fact that they communicate relatively less at that stage. Whereas, the other storage methodologies have higher average times. The centralized has highest since there is no data parallelism. 'CGWR' (*NineCellGrid* without

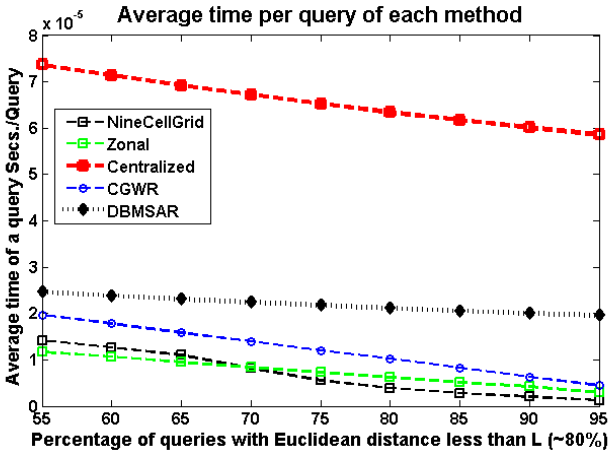


Figure 4.8: Average Time Per Query Comparison of various storage methodologies

replication) has relatively higher average times than *NineCellGrid* and Fully-distributed since there is no replication at all and so leading to heavy communication.

4.5 Summary

In this chapter, the illustration of data-load balancing as per the proposed algorithm is shown by considering a dataset of New York City which has water bodies, hence this dataset is experimented to study the data load migrations visually. And then evaluation of the optimal value of a heuristic (L) is done computationally. And finally comparative study of different storage mechanisms, centralized pattern, fully distributed pattern and *NineCellGrid* method, is done with respect to the metrics turnaround time and throughput.

Chapter 5

Conclusion

This thesis presents *NineCellGrid* approach to distributed data layout for GPS (Spatial Big Data) applications and quantitatively establishes that *NineCellGrid* data layout achieves better throughput and average turnaround times compared to fully distributed and centralized storage patterns. *NineCellGrid* is therefore well suited for a real-time applications where soft deadlines are easily met by exploiting data redundancy in a particular query among the nine neighboring cells. In our method, we improved the performance by exploiting redundancy in a particular pattern of data in each cell being replicated among its eight neighbors. Despite the high data redundancy, and the overheads of extra space used, the benefits of higher computation to communication ratio manifests as higher overall speedup for queries. In addition to that, there is a flexibility of lowering the replication factor, which also leads to balance the load at the same time. In addition, we gained not only redundancy in data but also fault-tolerance and high data parallelism. The *NineCellGrid* data layout incurs less communication and low latency compared to the standard methods, since computation is brought to where the data is located rather than having to move the data.

Data redundancy in the *NineCellGrid* enables easy task-load balancing among cells by migrating the computations, rather than redistribution of data to balance the load. There is data-load balancing done among the nodes so that, as the data increases there is no frequent redistribution of data to balance the load. Hence, this load balancing is a boon not only to

gain resource utilization but also to gain better throughput, which is observed from the results. When the percentage of incoming queries with Euclidean distance of source to destination is low, the throughput of *NineCellGrid* method is comparable with fully distributed and centralized methods. Whereas, when the percentage of incoming queries with Euclidean distance of source to destination is high, the throughput of *NineCellGrid* method is far better than fully distributed and centralized methods. This is due to the fact that communication is lowered remarkably. Hence, the overall performance of our method is higher than or comparable with fully distributed and centralized methods.

Chapter 6

Future Work

Our future work will focus on balancing the load at each node by manipulating the replication factors and shifting the load more frequently in dynamic manner instead of doing it at relatively higher time intervals. Job-scheduling algorithms that learn from previously observed queries can be devised to improve performance even more.

A hybrid version of 9-cell, 7-cell, 5-cell and 3-cell storage patterns can be devised inorder to reduce the load due to redundancy. Also, instead of using cell as a square, a regular hexagon can be devised, which leads to a lower redundancy overhead of seven instead of nine.

Bibliography

[1] Google Maps http://maps.google.com

[2] Distributed or Centralized Traffic Advisory Systems - The Applications Take. Otto, J.S. ; Dept. of Electr. Eng. and Comput. Sci., Northwestern Univ., Evanston, IL, USA; Bustamante, F.E.

[3] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.C. Herrera, A. Bayen, M. Annavaram, and Q. Jacobson, "Virtual trip lines for distributed privacy-preserving trafc monitoring," in Proc. of ACM/USENIX MobiSys, Breckenridge, CO, June 2008.

[4] Research on the Data Storage and Access Model in Distributed Computing Environment. Haiyan Wu Coll. of Comput. and Inf. Eng., Zhejiang GongShang Univ., Hangzhou

[5] Google White Papers

[6] H. Zhu, Y. Zhu, M. Li, and L. M. Ni, "HERO online real-time vehicle tracking in Shangai," in Proc. of IEEE INFOCOM, 2008.

[7] T. Logenthiran, Dipti Srinivasan Department of Electrical and Computer Engineering National University of Singapore, Intelligent Management of Distributed Storage Elements in a Smart Grid, 2011.

[8] Shashi Shekhar, Viswanath Gunturi, Michael R. Evans,KwangSoo Yang University of Minnesota, Spatial Big-Data Challenges Intersecting Mobility and Cloud Computing, 2012.

[9] V. Taliwal, D. Jiang, H. Mangold, C. Chen, and R. Sengupta, ”Empirical determination of channel characteristics for DSRC vehicle-to-vehicle communication,” in Proc. of ACM VANET, 2004.

[10] Z. Wang and M. Hassan, ”How much of dsrc is available for non-safety use?” in Proc. of ACM VANET, September 2008.

[11] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, ”CarTel a distributed mobile sensor computing system”, in Proc. of ACM SenSys, 2006.

[12] An ESRI Technical Paper June 2007.

[13] NHTS: National Household Travel Survey, 2009.

[14] DIMACS: http://www.dis.uniroma1.it/challenge9/download.shtml

[15] The Hadoop Distributed File System, Shvachko, K., Yahoo!, Sunnyvale, CA, USA, Hairong Kuang ; Radia, S. ; Chansler, R.

[16] Dijkstra Shortest Path Computation Algorithm, by Dijkstra.