Efficient Storage of Big-Data for Real-Time GPS Applications

Pavan Kumar Akulakrishna
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore, India
serc.pavan@ssl.serc.iisc.in

J.Lakshmi
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore, India
jlakshmi@serc.iisc.in

S.K.Nandy
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore, India
nandy@serc.iisc.in

***Abstract*—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 far, the data is distributed based on zones and it also 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. 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. We also build some data redundancy, which is used not only to enable failover mechanisms but also to target performance. This is done by nine-cell approach wherein each cell stores data of eight of its neighbours along with its own data. 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, effective load balancing of data ensures 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.

I. Introduction

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]. Various classes of people use GPS-applications for different reasons. Scientists use GPS to track some endangered species either in water or some other locations where human reach is not possible. 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 realtime 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.

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 contemporarily.

There is a specific big-data solution, for storage and distribution of data for a given problem [2]. In this paper we specifically look 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 realtime is of the form route from A to B with special parameters. Hence, distributed storage gives chance to parallelism and so speedup 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 paper, 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 failure 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.

The rest of the paper is detailed as follows: Section 2 describes the related work to highlight the efforts published in literature in this area; Section 3 details the *NineCellGrid* methodology proposed for storing the data and describes how computations in routing query can be generated using this mechanism; Section 4 explains the experimental setup and observations achieved so far; Section 5 concludes the work followed by Section 6 that elucidates the scope for future work.

II. 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].

A. 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 geodatabases are a relatively new format. Geodatabases are databases stored in Microsoft Access (for the "personal" geodatabase), as a special collection of files (for the "file-based" geodatabase), or higher-end applications (e.g. SQL Server, Oracle, Informix). A geodatabase 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.

B. 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:

- Geodatabase replication: Geodatabase replication allows you to distribute data across two or more geodatabases such that edits can be made independently and synchronized periodically. It has built-in safeguards against data loss, data redundancy, and system instability. Geodatabase replication requires at least one versioned ArcSDE geodatabase [12].
- DBMS replication: ArcSDE geodatabases are built on top of DBMSs that include technology for replicating at the database level. Geodatabases, like other applications built on top of these DBMSs, can be used with this technology. Using DBMS replication with geodatabases requires knowledge of how geodatabase 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 geodatabase replication [12].
- Data copying and loading tools: Another technique for distributing data involves simply copying data from one geodatabase 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 geodatabase in the office each night. This technique can also be used where the data is nonversioned or where only personal or file geodatabases are involved. However, it has no built-in safeguards against data loss or data redundancy [12].

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.

III. 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.

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.

Here, the GPS data that is concerned to a location is decomposed in such a way that entire earth is mapped with a mesh/grid of cell dimension L×L, where L is some fixed value (explained in the following sub-section). Each cell is representing a unique L×L area on earth as in Figure - 1.

Fig. 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 - 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.

NameTable

A table called *NameTable* (Table - I) that is indexed with row key as the 'Latitude' and column key as the 'Longitude' which indexes 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)].

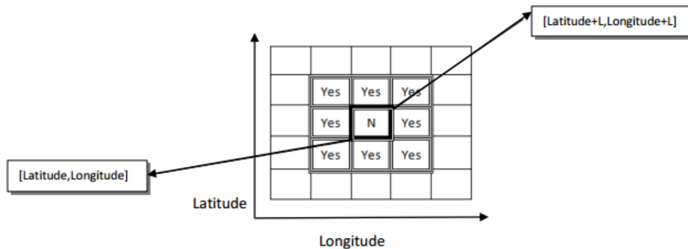
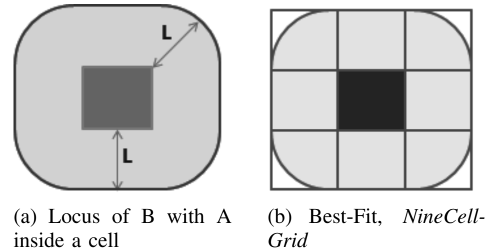


Fig. 2. Cell N, with identity [Latitude, Longitude] in NameTable

Dimension of cell (L)

Each 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 9 nodes and to locate those nodes in the *NameTable* is just the (-L, 0, +L) combinations to the keys, hence 9 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*

Fig. 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(a). In Figure - 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(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 less say (for instance), less than 0.2, which means queries with Euclidean distance less than L occur with a probability of 0.8. In order to achieve this criterion, L is chosen as

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

Where, P can be any percentage that gives better performance. Interestingly, if the percentage is considered high, that

		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 I. **NameTable** Here f and g are mappings from miles to Longitude and Latitude units

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% (an initial guess) in steps of $3-5\%$ it is found that around $\sim$80% suits better in this ("New York" dataset from NHTS) dataset taken (shown in experiments & results section). However, the value of P is different for different datasets. These experiments are done taking 5% as step size since very close percentages outperform each other. Hence, a higher resolution step does not help much. Finally, L is defined globally on substituting P% found in Equation I).

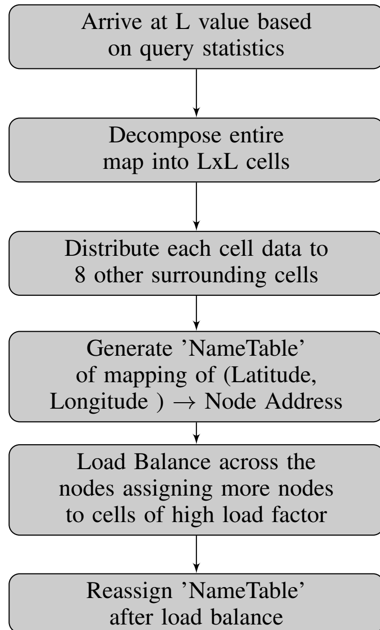


Fig. 4. *NineCellGrid Method*

Limiting Vertices and Case study of a 'A-B Route Query'

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

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

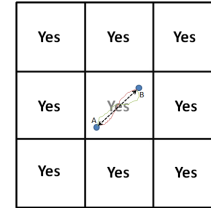


Fig. 5. Case 1: where A and B lie in same cell

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.

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

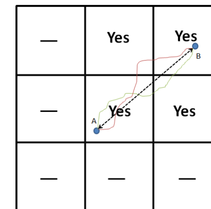


Fig. 6. Case 2: where A and B lie at 1 cell-distance apart

Here, the possible number of nodes that can compute the query are 4-6 nodes. Figure - 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.

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

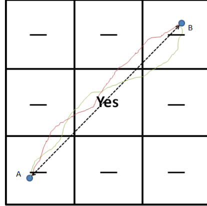


Fig. 7. Case 3: where A and B lie at 2 cell-distance apart

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

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

Load Balance

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 in order 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.

```
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 − δ )
4.          Find node P closest to E with load ≥ M − δ.
5.          Load(P) → Load(P) + Load(E);
6.          Load(E) → 0;
7.          NameTable[Latitude(E)][Longitude(E)] = Address(P).
8.      endif
9.  endfor
10.for each node E do:
11.     if( Load(E) ≠ 0 )
12.         Share the load among N nodes.
13.         (Load(E)/N) ∈ [M − δ, M + δ] & N ∈ (1,2,3,...)
14.         Update NameTable.
15.     endif
16. endfor
```

TABLE II. *Load Balancing*

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.

In this case load is not strictly balanced but given a relaxation δ *Algorithm: Load Balancing* (Table - II). Initially the load is estimated to each L×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.

IV. Experiments and Results

We use simulation to study the application-level performance of centralized, fully distributed and our Geo-spatially localized distributed storage (*NineCellGrid* approach) data dissemination models. Here, simulations are done using MPI. Dataset 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 - 8 shows the actual coordinates of the dataset used.

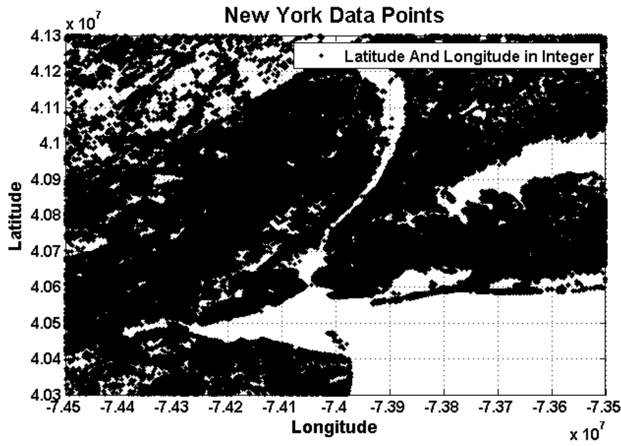


Fig. 8. New York City dataset actual coordinates [14]

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 - 9, 10 and 11 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 - 11) 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 - 11 would be either between 150 to 300 as the M is around 225 or zero.

Figures - 12, 13 and 14 show the histograms of load distribution of the nodes before replication, after replication and finally after the load-balancing respectively. On comparing Figures - 12 and Figure - 13 it is observed that the maximum value of the load in Figure - 12 is one-ninth that of the Figure - 13 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 - 14 the scenario after load balancing shown with a relaxation of 75 i.e., $\delta = 75$. After balancing in Figure - 14 the nodes get into either the region of $[M - \delta, M + \delta]$ or to 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 $< M - \delta$.

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

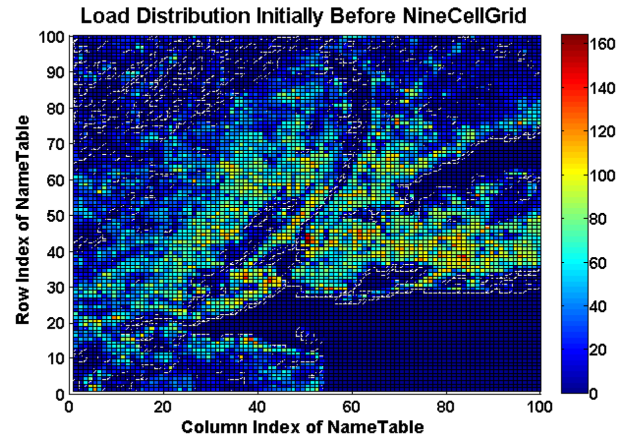


Fig. 9. Load distribution before replication

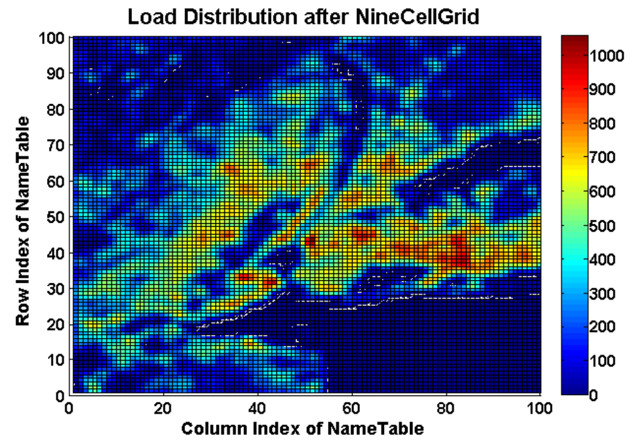


Fig. 10. Load distribution after replication

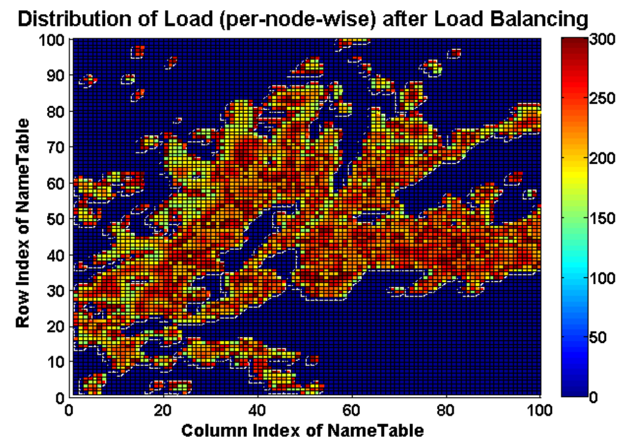


Fig. 11. Load distribution after load-balancing

City is calculated to be 6.8 miles (after selecting $P \sim 80\%$ for this dataset) from NHTS (National Household Travel Survey) [13]. This estimation of L is done by first calculating the value

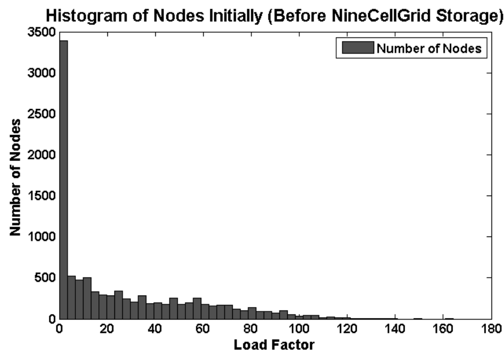


Fig. 12. Histogram of loads before replication in *NineCellGrid*

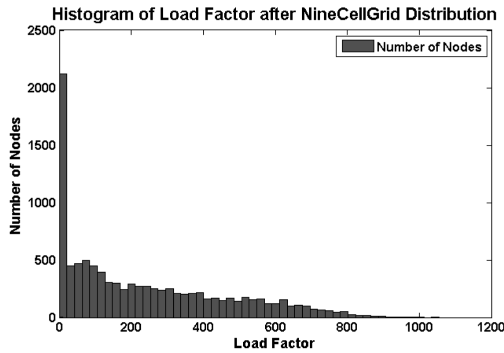


Fig. 13. Histogram of loads before load-balancing

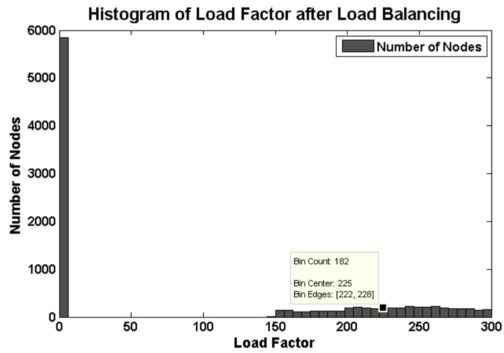


Fig. 14. Histogram of loads after load-balancing

of P (as P% in Equation I) and substituting in Equation I). The selection of percentage (P), which determines the L value, is done based on the analysis as shown in the Figure - 15. Initially taking 70% as the initial guess and then increasing in steps of 5% so as to get to the optimal percentage (P% in Equation I)). The graph shows 80% as optimal for this scenario. This P value of 80 is substituted in Equation I) to get L (which is 6.8 miles for this dataset). 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 P 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.

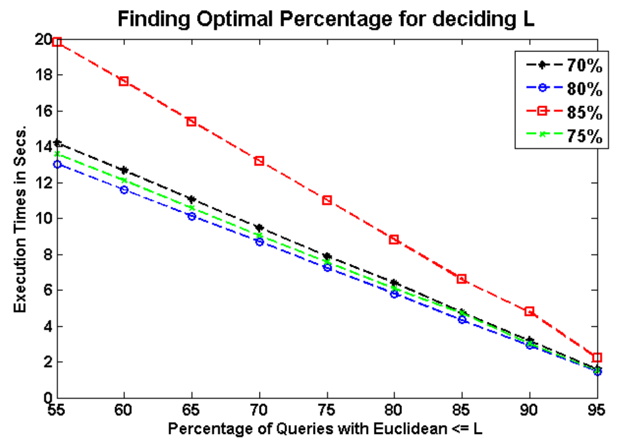


Fig. 15. Execution times vs. Percentages of queries with Euclidean distance of AB query less than L

Figure - 16, shows the comparison of throughputs for various storage methodologies. In figure - 16 the notations, *NineCellGrid* represents *NineCellGrid Method*, '*Zonal*' represents fully distributed pattern, 'Centralized' represents centralised storage pattern, 'CGWR' represents *'NineCellGrid Without Replication'* or *Cell-Grid-Without-Replication [CGWR]*, and finally 'DBMSAR' represents the DBMS as replication (replication value of 5 taken). Each of the methods are compared in context of the throughputs they yield. Number of queries are 1000. Graph in Figure - 16 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.

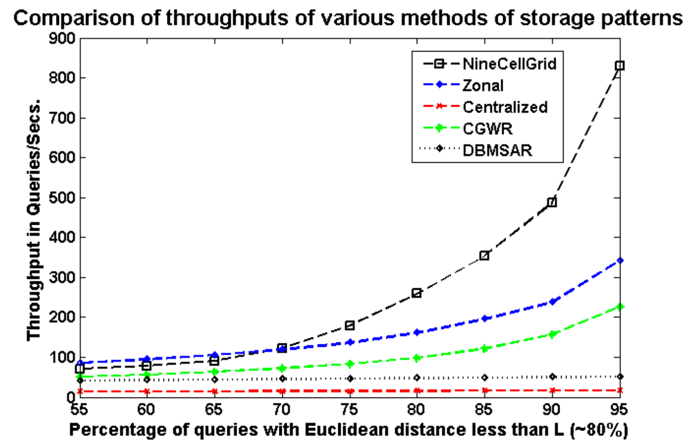


Fig. 16. Throughput Comparison of various storage methodologies

Figure - 17, 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 replication) has relatively higher average times than *NineCellGrid* and Fully-distributed since there is no replication at all and so leading to heavy communication.

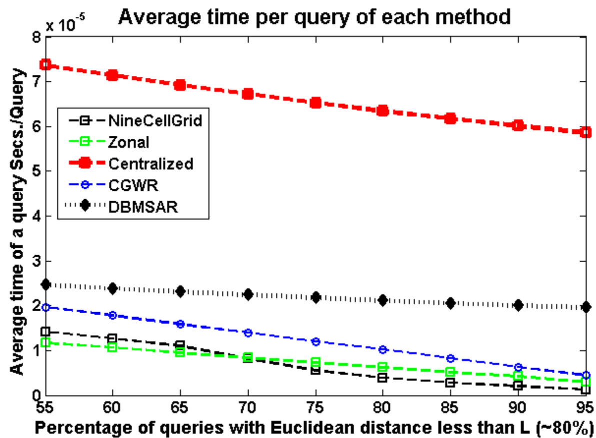


Fig. 17. Average Time Per Query Comparison of various storage methodologies

V. Conclusion

This paper 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. Hence, for a real-time application where responsiveness matters, this type of strategy is remarkable. In our method, we improve 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. Hence, the tradeoff of redundancy to communication overheads is done to overcome excess space utilization. In addition, we gain 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. *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. Data redundancy in the *NineCellGrid* enables easy load balancing among cells by migrating the computations, rather than redistribution of data to balance the load. There is load balancing done among the nodes so that, as the data increases there is no frequent redistribution of data to balance the load. In our approach, we adopt the relaxation to certain extent in order to overcome frequent load balancing. 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.

VI. 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.

References

[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 trafiňĄc 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] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Dehorah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robet E. Gruber. Google, Inc. Bigtable: A Distributed Storage System for Structured Data. OSDI, 2006.

[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.