Efficient Grid-GIS Framework for Spatial Data

A Thesis submitted
for the award of degree of

DOCTOR OF PHILOSOPHY

By:

Hari Singh

(950903034)

Under the Supervision of

Dr. Seema Bawa

Professor, CSED,

Thapar University, Patiala

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY, PATIALA - 147004, INDIA

October 2017

CERTIFICATE

I hereby certify that the work, which is being presented in the thesis, entitled **Efficient Grid-GIS Framework for Spatial Data**, in fulfilment of the requirement for the award of degree of **DOCTOR OF PHILOSOPHY** submitted in **Computer Science and Engineering Department** of **Thapar University, Patiala**, is an authentic record of my own work carried out under the supervision of **Prof. Seema Bawa**, and refers other research works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Hari Singh

Hari Singh
950903034

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Seema Bawa
31/10/2017

Supervisor
Prof. Seema Bawa
Computer Science & Engineering Department
Thapar University, Patiala, 147004, Punjab, INDIA

.....dedicated to all young generation researchers

Abstract

Geographic Information System (GIS) data is huge in volume and requires tremendous data storage capacity. Grid computing technology in the GIS domain provides cooperation and integration of services to implement a complex spatial function, and consequently provides a significant performance gain. The combination of GIS and Grid Computing, well known as Grid-GIS, has become a new research tendency.

The existing Grid-GIS architectures and frameworks, such as OGSI with WSDL and XML based Grid-GIS, OGSA with WSRF based Grid-GIS, and parallel processing oriented mobile agent based Grid-GIS, suffer from the less efficient data access, retrieval and complex procedures, for achieving fault tolerance, availability and scalability. Grid computing is characterized in dealing with a bag-of-tasks having few Inputs/Outputs (I/O)s. The analysis of voluminous spatial data, that is characterized as big data, requires few complex tasks, however the number of intermediate (I/O)s remain very high. The MapReduce computations are compatible for processing data-intensive spatial data that requires a large number of inputs and intermediate data. The MapReduce is also better than the mobile agent technology, as it provides built-in support for parallel processing operations and fault tolerance that abstracts the complexity of operations from the user. The integrated Grid and MapReduce for GIS data supplement each other by providing data analysis and computational environment together. Firstly, it provides high utilization of the resource pool. Secondly, the high data analytic feature of the MapReduce - Hadoop is complimented with the comprehensive accounting, resource utilization control, and policy management features of the grid. However, not much research work is found that integrates MapReduce and Grid-GIS. So, considering the benefits of the MapReduce, the proposed architecture and framework integrates the MapReduce in the Grid-GIS.

Three parallel spatial indexing algorithms based on MapReduce are also included as significant components of the proposed architecture and framework, these strengthen spatial data access and retrieval through indexing. H-bucket PMR Quadtree Spatial Index, parallel Hilbert TGS R-Tree Spatial Index and parallel Priority R-Tree Spatial Index are implemented. The H-bucket PMR Quadtree index is particularly designed for spatial data featuring lines. The other two indexes take Minimum Bounding Rectangles (MBRs) as approximation of spatial data. The parallel Priority R-Tree provides good performance in worst cases, such as, for non-uniformly distributed data (skewed data, data rectangles with high aspect ratio, clustered data, etc.).

Finally, an architecture and framework that integrates spatially indexed MapReduce and Grid-GIS "QUiPSHoT Grid-GIS" has been proposed. This proposed architecture and framework has been implemented and then tested by running it in an academic institution. Performance of the same has been measured for cost of bulk-loading spatial indexes, execution time, scalability and availability. The experimental results validate the competitive performance and usage of the proposed framework.

Acknowledgements

First, I would like to express my deep gratitude to my supervisor **Prof. Seema Bawa** for her invaluable advice and encouragement at every step of my Ph.D program. Without her unfailing support and belief in me, this thesis would not have been possible. Her contribution to this thesis goes well beyond her role as an academic supervisor and includes constant support on a personal level without which this journey may never have been completed. And for this, I am truly grateful. She is a great mentor for my life as well.

I also wish to extend my gratitude to the members of the PhD committee: Dr. Maninder Singh, Dr. A.K. Verma, Dr. V.P. Singh and Dr. Rajesh Khanna for their encouragement and insightful comments in relation to my research. I would like to many thanks all faculty and staff members of Thapar University, who have been kind enough to advise and help in their respective roles. I would also like to express my gratitude to Dr. B. R. Marwah, Ex. Professor at IIT Kanpur and Ex. Executive Director at N.C. College of Engineering, Israna, Panipat, Haryana, for his constant motivation and encouragement.

Finally, I would like to express my sincere and deep gratitude to my father Late Sh. Madan Singh and mother Smt. Pavetri Devi for their blessings. I would also like to express my sincere and deep gratitude towards my family member for their love, encouragement, care and support. Special thanks to my wife Namita for having faith in me and supporting me at every step. Without her support, I could not complete my Ph.D. program and finally a lot of love to my daughters, Nishika and Saanvi, to whom I could not give some of the time.

Hari Singh

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last several years, a huge amount of data has been generated all over the world as a result of existing automated computer applications. Advancement in computer networks and development of distributed applications like e-commerce and social networks have given a faster pace to the generation of data. Geographic Information System (GIS) data are huge in volume and requires tremendous data storage. The big volume of spatial data sharing and processing on traditional GIS software became inadequate. With increasing spatial data infrastructure, a large amount of distributed heterogeneous spatial data and GIS processing applications evolved. It is very difficult to process large amount of structured and unstructured data with traditional sequential programming methods. The large volume of data has led to the adaptation of parallel processing for efficient data processing over a set of collaborating computer machines. The thesis work focuses on spatial data and, processing and analysis of spatial data through emerging computing technologies. Grid computing and MapReduce are integrated for developing an efficient architecture and framework "QUiPSHoT Grid-GIS" for processing GIS data.

The organization of the thesis is as follows: Chapter 1 introduces to core components of the thesis work, Grid Computing, GIS and MapReduce. Chapter 2 provides a detailed literature survey of the work done so far on spatial occupancy approaches, Grid-GIS frameworks, integration of the grid and MapReduce, and MapReduce-based GIS processing approaches. Chapter 3 presents the proposed architecture and framework of an efficient Grid-GIS "QUiPSHoT Grid-GIS". Chapter 4 presents the design and implementation of different components of the framework. Chapter 5 presents experimental tests and demonstrations of the proposed framework for index building and spatial queries, for parameters, cost of bulk-loading spatial indexes, execution efficiency, scalability and availability. Chapter 6 describes conclusions and future scope of the thesis work.

1.1 Grid Computing and Architecture

The evolution of parallel and distributed computing started as a result of under-utilized processing power of personal computers (PCs). Earlier, PCs were the result of advance-

ment in hardware, such as processor, memory, etc. This advancement made PCs cheaper and affordable as compared to mainframes. Mainframe computers were quite powerful and popular for batch-processing tasks. The drawback of mainframes was that users needed to wait to get the output of their submitted jobs. However, the processing power of mainframe was utilized completely. The PCs provided comfort to users, as they need not to wait. But PCs were sitting idle after accomplishing a job and hence, their processing power was not utilized completely. Many distributed computing technologies have evolved to effectively harness the processing powers of PCS. Peer-to-Peer (P2P), Cluster Computing, Common Object Request Broker Architecture (CORBA) and Distributed Computing Environment (DCE) are examples of such early distributed computing technologies. A detailed comparative study among these is provided in [42].

Wireless network provides instant sharing of information through wireless electronic devices where communication infrastructure is not possible. However, packet drops are more in such kind of decentralized adhoc network, as all routing activity is handled by nodes [43]. The sensor nodes in wireless sensor network collect data and send this data to sink for post data analysis. The efficiency of wireless sensor networks improves through minimizing energy consumption incurred during the routing process [44]. Internet provided communication and exchange of information. But, it could not go beyond to provide a coordinated and collaborative computing using distributed computing data resources. The recent distributed computing technologies, such as grid computing, could make such things possible and enhanced utilization of processing power of computing resources. However, other issues started coming-up, such as failures, security, quality of service, etc. Grid computing addressed all such issues through a grid middleware. The grid middleware is a system software that provides interaction between the operating system and applications. UNICORE, Alchemi, Globus Toolkit, Legion, Gridbus, Condor are the examples of grid middleware. A detailed description and comparison among these middleware is provided in [45, 46, 47].

The grid computing is about coordinated resource sharing and problem solving in a dynamic multi-institutional virtual organizations (VOs) through a Grid Architecture that consists of a set of protocols, services, APIs and SDKs. A VO is formed when a number of distributed individuals/organizations share their resources, under mutually agreed sharing rules, to provide a solution for a particular task. Distributed individuals/organizations can be dispersed geographically and comes in a VO with partially overlapping objectives. These individuals/organizations are quite competent in one particular area/service providing. Grid utilizes the best competent services in the VO and provides to its users. The sharing goes beyond the file exchange and the coordinated sharing of other resources

such as softwares, computers, etc. is also possible under a controlled environment. The grid computing technologies complement the existing distributed computing technologies due to its scope over the dynamic multi-institutional virtual organizations [1].

The grid architecture organizes components into layers [1], as shown in Figure 1.1. The components within each layer share common characteristics but can build on the capabilities and behaviors provided by any lower layer. Collective layer implements a wide range of global services and application-specific behaviors through protocols defined in the Resource and Connectivity layers. Resource and Connectivity protocols facilitate sharing of a wide range of heterogeneous resources defined in Fabric layer.

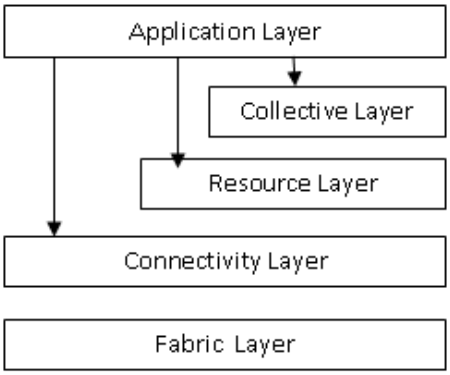

Figure 1.1: The Grid Layered Architecture [1]

The grid computing technology fully utilizes distributed resources and information in a virtual organization [48]. An "hourglass model" layered architecture was used by early computational grids [49]. In a feasible architecture of Grid Computing, the Open Grid Services Infrastructure (OGSI) is integrated with Web Services Description Language (WSDL) and EXtensible Markup Language (XML) schema to store state-full attributes in traditional web services [50]. However, due to the weaknesses of Open Grid Services Infrastructure (OGSI) [51], another technology for dealing with the state-full information, Web Service Resource Framework (WSRF), evolved [52]. It supports integrated data and services, and share dynamic resources such as, high-performance devices, high-capacity storage and heterogeneous systems. It separated the logic of state-full information of grid services from web services [53], and considered it as a resource. Moreover, the WSRF provides a series of specifications and interfaces [54] for describing the relation between services, managing resources, and grid resources in a grid computing environment [55].

The service-oriented architecture (SOA) has emerged and become popular with the advancement in Web services and grid computing. The Open Grid Service Architecture

(OGSA) was developed, as the SOA, in grid computing [56, 57, 58]. Based on the OGSA, an architecture of the grid, Globus Toolkit (GT 4), was developed by the Globus corporation [59]. The GT 4 [45] grid middleware is widely used for deploying computational grids that brought developers conveniences in Grid environment [60].

1.2 Geographic Information System (GIS)

GIS is a computer system capable of assembling, storing, manipulating, analyzing, and displaying geographically referenced information. The various researchers have defined GIS depending on its usage in different areas and disciplines [61, 62, 63, 64].

In the early days of GIS technology, usage of GIS was very limited due to expensive GIS services. However, its accessibility and adoption has been rising with the development of low cost software and hardware. GIS consists of four main components: software, hardware, data and people [61]. GIS software is an important component of a GIS system that is used to create, edit and analysis of spatial and attribute data. GIS hardware comprises components for data generation devices, such as digitizer, Global Positioning System (GPS) data logger, keyboard, scanner, etc.; storage devices such as hard disk, CD/DVD, pen drive, memory card, etc.; processing device, such as computers, servers, laptops connected in networks; and output devices for data analysis, such as printers, plotter, and monitor.

The geographic data, comprised of location and other characteristics, are obtained through primary or secondary sources. The geographic data collected by users is known as primary source. The geographic data purchased from commercial organization is known as secondary data. There are three dimensions of primary and secondary data: spatial, temporal and thematic or attribute data. The spatial dimension describes location information. The temporal dimension describes time of data collection. The thematic or attribute data describe characteristics of feature to which the data refer. It is also known as non-spatial data.

Primarily two spatial data formats are used for representing GIS data: vector data and raster data. A third data type is associated with the two spatial data types that provides attribute information in tabular format. The vector data type uses points, lines and polygons to represent geographical data boundaries, such as roads, lakes, railway lines, rivers, boundaries of parks, etc. [65]. This representation uses X,Y coordinates to plot data, but can not represent variability features, such as soil type [62, 66].

(i) **Point:** A point feature is used to represent a particular location and is represented

with a single coordinate in X-Y plane (x,y).

(ii) **Line:** A line feature is used to represent spatial object that has a particular length. It is composed of a series of points that locate the spatial object.

(iii) **Polygon:** A polygon feature is used to represent an enclosed area, such as the perimeter of spatial geometry. It is composed of a series of line objects.

The raster data represents digital imagery and is represented with two parameters, a grid of pixels and intensity and color of the pixels. The location of a pixel is represented with a particular row and column. The amount of pixels present in a given area determines the resolution of the image. A clearly visible image or high resolution image is composed of a large number of pixels in an area. While a poorly visible image or low resolution image is composed of a small number of pixels in an area [65].

The vector and raster data models can be compared on some parameters which are as follow:

(i) The data structures used to represent raster objects are simple, easy to understand and use. The vector data objects use compact data structures that are difficult to understand and use.

(ii) The spatial location and geometry represented with vector objects are more precise and accurate than raster data objects, while spatial inaccuracies are very common in raster data models.

(iii) The raster format can easily represent spatial variability in objects, such as variation in color in an image, and does not require any special or expensive computer technology. The vector format can not represent spatial variability and is generally used to represent graphics, such as hand drawn maps, that requires special computer technology for representing spatial objects accurately.

(iv) The visibility of a raster image is less pleasing because of heavy and blocky appearance of boundaries, while graphics with vector data type is very sharp, clear and pleasing.

The processing and data-storage power of recent distributed computing technologies have been utilized in a number of domains. A distributed network technology has been used, for studying the behavioral aspects of individuals in social networks, in a secure private sensitive information [67, 68, 69]. The voluminous GIS data were difficult to process on a stand-alone computer. So, advancement in the field of distributed computing was also applied to GIS. A distributed GIS is characterized with distributed computational and data resources. The early approaches used for data access in distributed GIS brought

raw data to computation for processing. The voluminous spatial data put much traffic on the network due to large amount of data transfer. A problem with distributed GIS was of heterogeneity of spatial data located at different places. Due to it, a large amount of spatial data was not useful for other GIS applications and hence was not interoperable. Various GIS specifications for resolving interoperability issue evolved, such as the Geographic Markup Language (GML) [70]. Ontology, GML-based spatial data exchange model, and Scalable Vector Graphics (SVG) based information visualization model are used for heterogeneous GIS data integration and inter-operation. The Open Geospatial Consortium (OGC) web services [71] enabled seamless integration of widely distributed heterogeneous geo-processing and location services for heterogeneous spatial data. It provided Web Map Service (WMS) and the Web Coverage Services (WCS) specifications. The WMS specification permits interoperability through keeping multiple maps on multiple distributed servers and sharing interactively for a particular task. Similarly, the WCS specification also permits interoperability through sharing spatial data distributed on various coverage servers [72]. The Internet based GIS and Web-GIS provided interoperability between different GIS platforms and systems, and delivered GIS data and services. However, problems were faced due to loosely coupled client-server mode and stateless implementation of Web-GIS to the integration of spatially distributed GIS services. The collaborative tasks and complex GIS services, implemented through a chain of simple GIS services were unable to be accomplished due to the stateless characteristic of Web-GIS. It resulted in lower efficiency as the result of one task could not be utilized for processing another service without another web request. The weakness and problems of data distribution of Web-GIS is solved with advanced technologies, such as grid computing and web services [73].

The grid computing technology provides an improved distributed GIS, as some of the problems faced in implementing distributed GIS were resolved by the grid technology, such as a built-in support for a good dynamic resource management and security features over the grid of heterogeneous systems. The grid computing also provides a high scalability for GIS data processing and analysis with increased volume of the GIS data on distributed Grid-GIS sites [74].

The development of grid middleware, for integrating Grid technology and GIS, provides solution to the distributed and heterogeneous spatial databases, and expand range of GIS services by integrating services available in isolation. The grid computing technology in the GIS domain provides cooperation and integration of services to implement a complex spatial function. The Grid-GIS technology is applied in various fields. The spatial data mining on Grid-GIS (Grid SDM) is an application of spatial data mining in

grid environment [64]. It uses four layers: Application client, Data mining middleware, Spatial data mining server and Data server. The Application client provides an interface for distributed spatial data access. The Data mining middleware decomposes a task onto different grid nodes. The Spatial data mining server accesses, processes and discovers knowledge from spatial data distributed on grid nodes through OGC compliant standard unified spatial data services. The other applications of Grid-GIS technology is the Spatial Decision Support Emergency Response Services Architecture (SDSERSA) that integrates Spatial Decision Support System (SDSS) on the Grid-GIS [74], Grid-GIS in digital mine [75], Digital Coal Mine Safety Grid GIS (DCMSGG) based on OGSA with the technology of dotNET and Web Services [76], and etc.

1.3 MapReduce-Hadoop (A part of the proposed framework)

The MapReduce programming model offers a new distributed environment for parallel processing. It is a loosely coupled architecture where computing nodes can be added or removed on the fly to a cluster and provide scalability. In a cluster, one node acts as master node and others work as slave nodes. The cluster makes a distributed file system that harnesses the storage capacity of all nodes in a cluster. The simplicity, ease of use and ability to easily handle large datasets, ease of programming by using the Map and Reduce functions, the fault tolerant nature of the MapReduce model and the property of abstracting parallelization from users are reasons that have motivated researchers to use the model. It provides a good performance through scaling out to computing clusters.

It was predicted that parallel processing would provide new ways of thinking about the existing concept of programming language, operating system, storage system, etc. for large distributed systems [77]. However, parallel processing is complicated and frameworks that provide parallel processing use abstraction to simplify things. The Hadoop a Java implementation of MapReduce, is one such framework that works on key-value storage concept. It has mainly two components MapReduce and Hadoop Distributed File System (HDFS) [78]. The MapReduce part of the Hadoop abstracts all details of parallel processing from users and they get a very simplified framework for programming. The MapReduce has become very popular for parallel processing of arbitrary data. It works on a divide-and-conquer strategy and breaks a computation into sub-computations over a set of computers in a cluster that operate in parallel. Each smaller computation is

handled separately and the result of the computation is returned at a central point.

A detailed survey of approaches that supports MapReduce for distributed data management and processing, such as MapReduce implementations, High level language support for MapReduce, MapReduce implementation on database operators, Database Management System (DBMS) implementation on MapReduce, MapReduce extensions for data-intensive applications, is presented [79]. The powerful and simple architecture of the Hadoop provides a good performance for large amount of data processing when configuration parameters are calibrated properly [80, 81, 82]. However, the performance of MapReduce is affected by Input/Output (I/O) [83], data locality issues [84], scheduling strategies [85, 86], and indexing over input dataset [87, 88, 89, 90, 91, 92]. Several factors contribute towards improving MapReduce, such as data access, avoidance of redundant processing, early termination, iterative processing, query optimization, fair work allocation, interactive real-time processing, and processing n-way operations [93].

1.3.1 The Hadoop Architecture and Work Flow

A number of researchers have presented work flow of the MapReduce - Hadoop [94, 95, 96, 97, 98, 99]. The Hadoop is configured for configuration parameters, such as data block replicas, number of mappers and reducers, block size, etc. prior to submitting a job. The Hadoop works in map and reduce phase. Users submit a job through a MapReduce script consisting of map and reduce function(s), and libraries, along with input and output. A MapReduce task broadly flows in the following twelve steps, as shown in Figure 1.2.

(i) Client starts HDFS.

(ii) Client starts MapReduce.

(iii) Client loads file onto HDFS; Splitting the file up into blocks and pipeline replication of blocks happens automatically.

(iv) Client submits a MapReduce job to Job Tracker

(v) The Job Tracker takes the detail about blocks of the HDFS containing data chunks for input data from Namenode.

(vi) The Job Tracker supplies java code to run map computation on local data to Task Trackers containing input file chunks.

(vii) The Task Tracker starts Map computation

(viii) The Task tracker periodically sends Heartbeats and Job status to the Job Tracker.

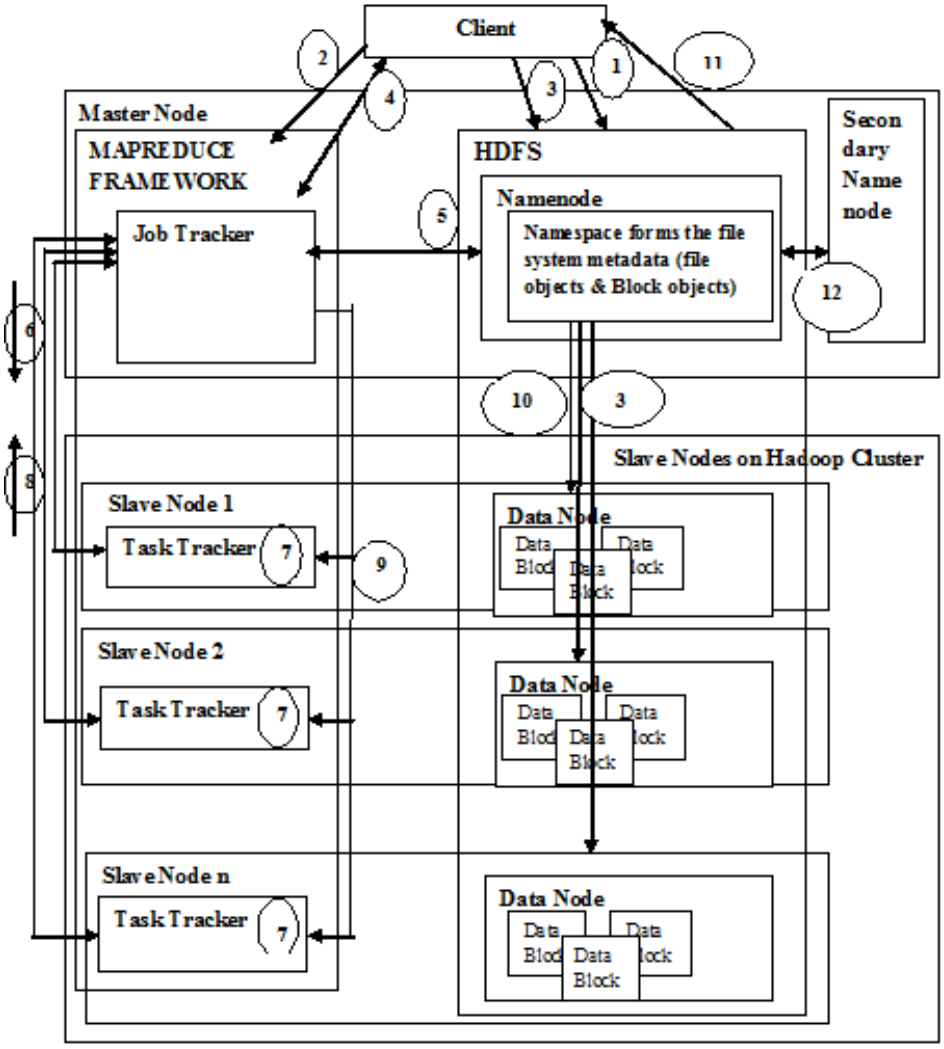

Figure 1.2: A MapReduce task flow in Hadoop architecture

(ix) The Task Trackers finishes map tasks. The Job Tracker starts Reduce task on datanode(s) and instructs the reduce task to collect intermediate results from all the map tasks.

(x) The Reduce task begins a final computation phase. The Reduce task writes the results to an output file and finally the aggregated results are written to HDFS; splitting the file up into blocks and pipeline replication of the blocks happens automatically.

(xi) Client can read the result file from HDFS with APIs.

(xii) The Namenode keeps a backup of namespace and updates at secondary namenode periodically since beginning.

The namenode maintains a list of datanodes in the cluster and the blocks belonging to each data node. When the datanode registers first time, it immediately sends a block report revealing block locations. Each datanode sends a block report to namenode which contains the block replicas in its possession periodically. The block report contains identity number of the blocks, length of the blocks, and time of generation of each block replica. The periodic heartbeats are sent by each datanode to namenode to inform the disk utilization statistic, number of data transfers performed for namenodes space and load balancing decisions, and to keep its status alive and not dead. For dead datanodes, namenode facilitates its block creation, through replication, on idle datanodes.

1.3.2 Performance Metrics for the Hadoop Cluster

This section discusses issues, related to the Hadoop configuration calibration metrics and large scale data processing, that affect the performance of the Hadoop Cluster.

1.3.2.1 The Hadoop Configuration Calibration Metrics

The performance of a MapReduce job is dependent on parameters set-up during configuring Hadoop for running MapReduce jobs. A high performance is obtained for a specific value set of these calibration metrics. Following are the metrics that affect the performance of a MapReduce job:

(i) **Input and output types:** The four factors that affect performance of a MapReduce [83] are: a) Direct I/O vs Streamed I/O: The direct I/O reads data from a disk directly, whereas the streamed I/O reads data from storage by an inter-process communication scheme, such as TCP/IP or JDBC. The streamed I/O is more preferable when reading data from a remote node, otherwise the direct I/O takes less time for data reading from a local node. By default, the Hadoop uses a streamed I/O for all local or remote data read. b) By default, the Hadoop does not provide any indexing on input data. A suitable indexing filters out records to avoid unnecessary computation and speeds-up query performance. c) Immutable decoding versus Mutable decoding for parsing: The raw data is converted into a set of records before the map function processes the data. The records are presented in a key-value form and the fields in each value part are decoded to data types. The raw data, decoded as immutable records, are read-only and field for the value is set once and cannot be changed after the record is created while field for mutable records is reset. Whenever the map function parses the next input record, a new

immutable record is created while for mutable decoding fills the fields of a mutable record with the next record. The number of immutable records becomes equal to the number of input parsed records, and consumes more memory and CPU overhead while only a single mutable record is created. By default, the Hadoop employs immutable decoding while parsing. d) Processing with the fingerprint of the key during sort-merge phase: For aggregation tasks the cost of key comparison is reduced by comparing keys only for the preprocessed output on the fingerprints of the original keys. Such a fingerprint of actual key strategy on the sort and merge considerably reduces the Central Processing Unit (CPU) execution time.

(ii) **Primary memory size:** As the namespace uses primary memory for building metadata (data about the data) over an HDFS file system. This metadata generally consists of information about file chunks contained in data blocks on different slave nodes over an HDFS. So its size is determined by the size of the input dataset. A shortage of Random Access Memory (RAM) on Namenode crashes a job, so generally the RAM on a Namenode is kept more than the RAM on slave nodes. It is approximately close to one-third of the total memory of the Hadoop cluster.

(iii) **Block size:** The size of the input dataset and input split affects the number of Map tasks. Setting the block size of the input dataset can configure the splits. If size of data blocks is kept small then CPU bursts are short and upon task completion significant data transfer takes place to collect intermediate data for reducers. Increasing the data block size and consecutively reducing the number of tasks can reduce this data transfer time. However, large data blocks increase the execution time per block and hamper parallelism. The output of a Map task per data block affects performance of the Hadoop cluster, if the former is directly proportional to the latter. In such a situation, large block size causes map-side spills.

(iv) **The number of Map and Reduce tasks:** The Number of Map and Reduce tasks on slave nodes is determined by the strength of the processor. However, when the processor strength allows many map and reduce tasks, then the numbers must be chosen carefully. High Map/Reduce tasks means short CPU burst per Map/Reduce computations and low Map/Reduce tasks mean large CPU burst per Map/Reduce computations. The number of Map/Reduce also relates to the data chunk in each data block and hence to the CPU execution time. If map computation is simple then CPU bursts are short but the startup overhead becomes significant as compared to the computation. And upon task completion, intermediate data transfer takes place for data shuffling to partition the data among reducers. Larger CPU burst computations decrease parallelism feature and hence reduces the performance.

(v) **Data replicas:** Providing a fault tolerant system with data replication has been the big characteristics of the Hadoop cluster. However, it may degrade performance. So a limit must be kept over the number of replication of data blocks. There are several reasons for it. Firstly, creating replicas over a large cluster becomes an expensive operation. Secondly, when replicas are created and put on different data blocks, then data transfer in the form of replica takes place over the network and causes network resource consumption, and puts load over the network. Thirdly, the generated replicas are put on data blocks of the slave datanodes and hence, consume local disk capacity.

1.3.2.2 The Factors for MapReduce Programming Logic

The factors related to a structure of the MapReduce programming logic affect the performance of a MapReduce job. A list of such factors is as follows:

(i) **Partition function or Data locality:** MapReduce is capable of processing huge amount of data by partitioning input data according to a partitioning function. It distributes partitioned dataset over a cluster of computing nodes. Each computing node carries out similar processing on the local dataset and returns local results to the master computing node. Balancing load depends on the partitioning function. If the partitioning function is able to uniformly distribute data over computing nodes, then the cluster gets a uniform load balance of tasks. Otherwise, some computing nodes completes its part of processing early and sits idle, while the other computing nodes finish late and become bottlenecks for the whole task. When an input dataset is put on the Hadoops HDFS, it is splited into data chunks of the size of the data blocks on the datanodes with a default value of 64 MB. If data is split in such a way that the map computation in a datanode does not find the data locally, then more data transfers will occur and this will put an extra burden on the network and also decrease the efficiency of computation. The performance can be increased, if input data is partitioned in a particular way, so as to maximize the data locality. Then the movement of intermediate data generated as a result of Map computation over the network can be reduced and consecutively, performance can be enhanced.

The data-locality issue in heterogeneous environments is focused on improving the MapReduce performance in the Hadoop clusters [84]. The aim is to minimize data movement between slow and fast nodes by an initial data placement scheme that distribute and store data across multiple heterogeneous nodes based on their computing capacities and then applying data redistribution scheme due to appending of

new data to an existing input file, deletion of data blocks from an existing input file, and addition of new data computing nodes into an existing cluster. Data locality is used for reading and writing data to local disc and hence reducing the network overhead and bandwidth.

(ii) **Data filtering:** For executing some queries, some of the tuples from a dataset are required while for other query only particular field data from a tuple is required. A filter function applied over the input dataset, after a careful analysis of the processing required by a Map function over the dataset, can avoid unnecessary processing of data that is not relevant or not important for a particular task. By filtering out such data, for which processing is not required, can save CPU cycles for such unwanted resource consuming tasks and performance improvement can be achieved.

(iii) **Indexing:** The Hadoop by default manages and processes data in the form of key-value storage. To process a job, the map part of MapReduce scans entire data distributed on datanodes. It is therefore more suitable for processing sequential queries and unfit for random read queries. The support for the latter type of queries is provided through indexing over distributed data. It provides a selective access mechanism for data and avoids processing any undesired part of the data.

(iv) **Recomputation:** MapReduce produced outputs for a job and intermediate results of a longer processing job are stored on the disk. It provides a fault tolerance mechanism for MapReduce jobs. However, these results cannot be re-used by another MapReduce job. The results are recomputed to carry out processing of another job. If alike relational databases, it becomes possible to reuse the results, it would certainly improve the MapReduce job performances.

Chapter 2

Literature Review

2.1 Grid Computing Types and Characteristics

The Section describes the types of grids and the general grid characteristics in sub-sections 2.1.1 and 2.1.2, respectively.

2.1.1 Types of Grids

Virtualization and provisioning differentiate grid computing from other styles of computing, such as mainframe and client-server. The virtualization and provisioning applied over the infrastructure, application and information resources form the infrastructure grid, application grid and information grid, respectively [100].

(i) **Infrastructure Grid:** Infrastructure grid manages hardware and software resources. The hardware resources include networks, storage, memory and processors. The software resources include databases, application servers, storage management, operating systems and system management. These infrastructure resources are treated as a single pool and are allocated on demand. It saves money by eliminating under-utilized capacity and redundant capabilities. The distribution of computing and storage capacity among many different computers minimizes chances of single points of failure, and improves the availability of the system [48, 49].

(ii) **Application Grid:** Virtualization and provisioning in the Application Grid refers to using the application resources of the grid, such as application logic and process flow, as services. It encourages agile software development and better software reuse. The Service-Oriented Architecture (SOA) is compatible for building applications in grid computing. It provides a set of services for building applications with such characteristics. The services are independent and well-defined encapsulation of software functionality. The services are invoked in a distributed execution environment using heterogeneous platforms. The XML-based Web Service standards is very popular for implementing the SOA. It overcomes the drawbacks of earlier

distributed computing architecture in three aspects - standards, broad adoption, and loose coupling. The Web Services standards define Web Service Description Language (WSDL) and (Simple Object Access Protocol) SOAP for interfaces and message passing. The Web Services standards are broadly adopted as these are based on the underlying pervasive Internet standards, such as HTTP. The Web Services standards provide loose coupling between services as compared to the distributed architectures [50, 51, 52, 53, 54, 55].

(iii) **Information Grid:** Virtualization and provisioning in the Information Grid refers to using information resources of the grid, such as data and metadata, as services. The data can be structured, semi-structured, or unstructured. It can be created by any application and having storage in any location, such as databases, local file systems, or email servers. The information grid thus provides a way for better exploitation of information resources [56, 57, 58].

2.1.2 Grid Characteristics

The following general grid characteristics are expected to be present in any grid architecture and framework [63, 1, 48, 49].

(i) Abstraction/virtualization: The grid abstracts functions from the infrastructure, application and information in the form of services for computing and problem solving in a collaborative environment.

(ii) Resource sharing: Sharing of data resources, storage resources, computing resources, information resources, knowledge resources in a distributed environment is possible with the grid middleware.

(iii) Flexibility/programmability: The programmability feature of the Grid provides workflow management and resource reconfigurability, and makes grid a flexible architecture. Grids can provide a flexible resource scheduling, resource allocations and configurations, in real time through grid middleware.

(iv) Determinism: The grid provides a mechanism to fulfill the requirements of users for running a particular task. It determines a matching between the requested resources and the provided resources.

(v) Decentralized management and control: The Grid architecture supports decentralization of management and control over resources. It allows multiple applications to coexist simultaneously.

(vi) Dynamic integration: The grid architecture allows coordinated use of resources distributed at multiple sites for computation through various integrated approaches. The integration brings virtual organizations in picture that collaborate to handle a specific task.

(vii) Scalability: The grid provides high scalability by expanding or shrinking resources for handling a specific task. The distributed resources are taken from both the local or remote sites.

(viii) High performance: Grids provide high performance due to abstraction, integration and sharing of resources.

(ix) Security: A grid provides a secure environment for computation as the sharing is highly controlled, with resource providers and consumers mutually agreeing on sharing resources and rules.

(x) Pervasiveness: Grids are broadly adopted as these are based on the underlying pervasive Internet standards, such as TCP/IP.

(xi) Customization: Grids provide flexibility for workflow, scheduling and resource reconfigurability, and therefore are capable of customizing to fulfill different requirements.

2.2 Spatial Occupancy Approaches in GIS

The spatial data representation is broadly based on spatial occupancy methods, as shown in Figure 2.1, that decomposes the region from which the data is drawn into sub-regions. The spatial occupancy methods can be categorized into two parts: overlapped or non-disjoint decomposition and disjoint decomposition. In the non-disjoint decomposition methods, an object is associated with only one bounding rectangle commonly known as Minimum Bounding Rectangle (MBR), such as R-tree [101]. The R-tree allows overlapping of objects to minimize coverage in intermediate nodes. It requires less storage and large search time as multiple nodes need to be searched. A variant of the R-tree known as the R*-tree uses the tree node splitting to better rearrange objects and, improves on the storage utilization and search time as compared to the R-tree. The major drawback of the non-disjoint decomposition based methods is that an object may be spatially contained in more than one bounding rectangle, yet it is only associated with one bounding rectangle. A spatial query may require searching many bounding rectangles in search of an object. In the worst-case, the whole database or all bounding rectangles would need

to be searched. An efficient solution to spatial queries, such as computing closest point to the boundary of a circle [102] and hyperplane [103] among a set S of n points, has been provided.

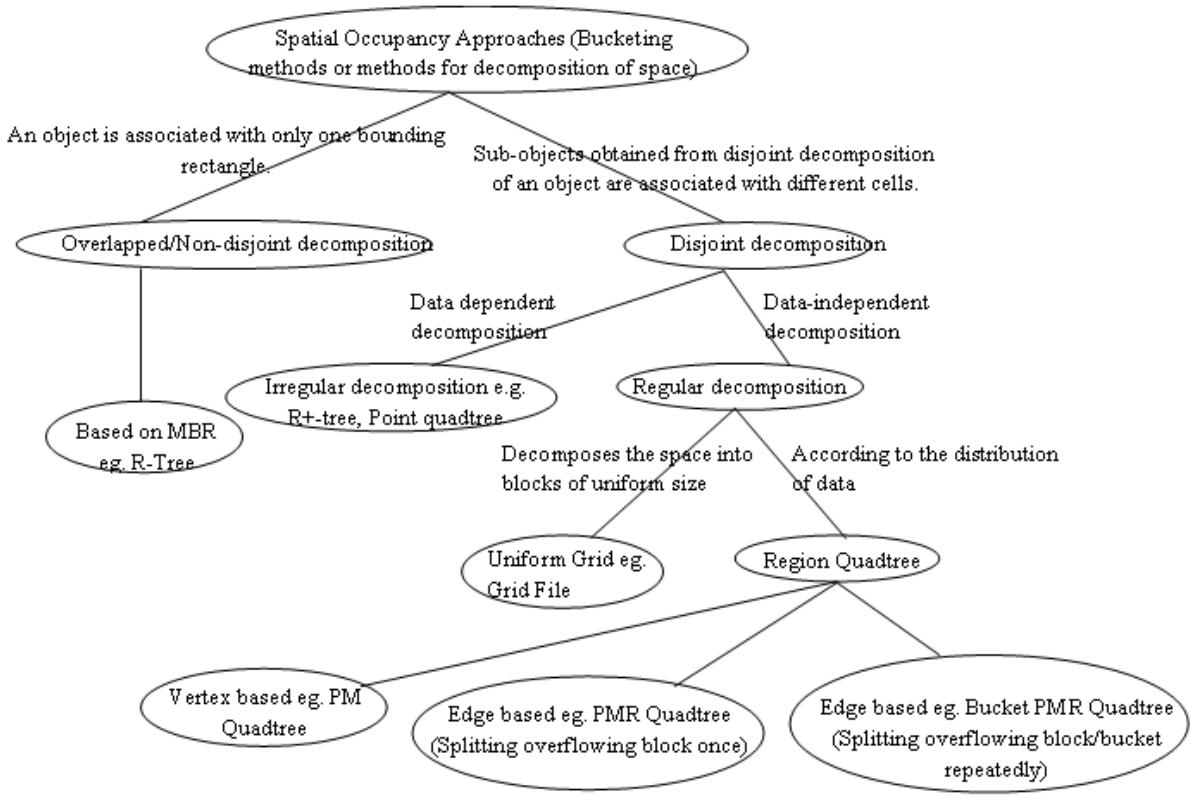

Figure 2.1: Spatial Occupancy Approaches

2.2.1 Disjoint Decomposition Approaches (Quadtree-Based)

All the overlapped or non-disjoint data structures suffer from a high load time of the structure, poor tree structure and poor search time, but they provide good storage utilization. On the other hand disjoint decomposition methods require more storage, but they reduce search time considerably. Disjoint decomposition causes sub-objects obtained from the main object to be associated with different cells. This causes a little problem when the area covered by an object is calculated because it forces area calculation for all cells, to which an object is associated. The R+-Tree [5], Quadtree [104] and Grid file [105] are examples of data structures that follow disjoint decomposition approach.

The R+-tree is a variant of R-tree that allows disjoint decomposition of space, by not allowing overlapping among intermediate nodes. It leads to a good improvement in search time, but requires more storage as compared to the R-tree and R*-tree. The drawback of

R+-tree is that the decomposition is data-dependent, which makes it difficult to perform some tasks that require composition of different operations and data sets. In contrast, the grid file index and quadtree data structures have a greater degree of data-independence. The grid file index [105] uniformly decomposes a grid space into blocks of uniform size with the use of a multidimensional hash algorithm. The spatial dataset is mapped onto a grid of uniform sized cells and accesses through an indexed grid file. A grid file consisted of a one dimensional array that provides a mapping from spatial location to grid directory. Each directory with-in an n-dimensional grid directory points to a specific dataset. Though this method has a simple structure, but it also has a high disk I/O time. Two disk accesses are required for obtaining the data. The first disk access obtains directory entry and the second retrieves the data. Besides a high disk I/O time, the grid file index is more suitable for representing uniformly distributed data. In general, the spatial data are not uniformly distributed, and so, a more flexible quadtree data structure that suits to non-uniformly distributed data has been considered in our work.

A quadtree represents a class of data structures that is based on recursive decomposition of space. The different types of quadtrees exist based on three criteria: first, the type of data to be organized in a data structure such as point, area, curve and volume. A point quadtree represents a space partitioning on the basis of point data. The region quadtree builds a data structure to represent the interior region of spatial objects. The quadtree for curvilinear data specifies the boundaries of spatial objects. It is either linear or non-linear. A volume quadtree specifies a region data of higher dimension (dimension more than two). Second, the data decomposition process, such as a regular or irregular decomposition. And third, the resolution, i.e. value of splitting threshold or the depth of the tree [104]. An irregular decomposition approach recursively decomposes a space, but sub-spaces generated are not of equal sizes. A point quadtree is an example of irregularly decomposed quadtree data structure. In irregular or non-uniform decomposition, the degree of data dependency is very high and space partitioning becomes arbitrary. It causes difficulty for tasks that require composition of different datasets and operations. A regular decomposition approach recursively decomposes the space into four quadrants of equal sizes. A region quadtree is an example of the regularly decomposed quadtree data structure.

A region quadtree [104] recursively decomposes a region containing spatial objects (points, area, curve, etc.) into four sub-regions of equal size, until a threshold minimum value is reached to realize a regular decomposition approach. It encodes only areas hierarchically. A region quadtree is used to represent point data, such as MX quadtree and PR quadtree. A linear or non-linear curvilinear data are also realized with region quadtree,

such as Edge quadtree, Line quadtree, PM quadtree family, and Polygonal Map Random (PMR) quadtree. An Edge quadtree causes a regular decomposition of a region (Polygon Map) containing a linear feature until a square is obtained that contains a single curve to be approximated by a single straight line. A Line quadtree does a regular decomposition of the region (Polygon Map) until the sub-region has line segments passing through their interior. It requires rectilinear regions having boundaries on the borders of pixels. It encodes both area and boundary of individual region hierarchically. The Edge quadtree, Line Quadtree and MX Quadtree suffers from a drawback that these works on approximations of polygonal map and cannot represent any of the properties of polygonal map, such as the multiple line segments meeting at a point (generally more than 5).

The PM quadtree overcomes the drawbacks of the Edge quadtree, Line quadtree and MX quadtree. The PM quadtree represents a data structure for representing polygonal maps that are either vertex based or edge based [104]. It builds the data structure by recursively breaking up a collection of vertices and edges of the polygon map until a simple structure is obtained that is represented with some other data structure. The vertex-based member of quadtree family, the PM quadtree inserts a line segment into a region by repeatedly sub-dividing the region until the resulting region contains at most a single vertex. A region containing a vertex of a line segment, allows another line segment in the region, if it has the same vertex. If the vertex of the line segment is not the same or the common vertex of two line segment falls outside the region, then the region is divided into two sub-regions.

In a PMR quadtree [106], the edge-based member of a quadtree family, there does not arise a need to sub-divide a region, where two line segments or their vertices are close. The line segments are inserted into the PMR quadtree, starting with inserting first line segment into an empty block. A block is split into four quadrants of equal sizes when the object count in the block reaches a threshold value. The splitting threshold is the permissible number of line segments in a block. A new line segment is inserted into a block, if it intersects the block, and the block is checked for the splitting threshold. A block is split into sub-blocks only once, as it contains a few very close lines. It is important to keep the particular value of the splitting threshold, as too small value causes many subdivisions that leads to many empty sub-blocks and hence the increased storage requirement. A large value of the splitting threshold causes a decreased construction time and storage requirement, but at the same time it increases the time for performing operations on it. The order of the inserted line segments decides the shape of the PMR quadtree.

However, in the parallel construction of the PMR quadtree, the insertion ordering of line segments is not known as the lines are inserted simultaneously, in parallel, and therefore,

a slight modification to the PMR quadtree, known as the bucket PMR quadtree, is used. As the tree data structures generally accesses data from primary memory and pointers are used to access data stored in pages of secondary memory. However, it gives rise to page faults. The bucketing method overcomes this problem by collecting data objects into sets called buckets and providing access to buckets through appropriate address computation mechanism. In bucket PMR quadtree, the block or bucket is split repeatedly until a splitting threshold is reached, unlike the PMR quadtree which allows splitting a block only once. The splitting threshold of a bucket is also represented as the node capacity of the tree node. The node capacity of a data structure tree node is defined as the data holding capacity of the node in terms of size of the data in the tree. It decides the number of objects in a bounding rectangle. The objects are grouped according to their proximity for better results.

It was found through the above discussed theoretical analysis of the non-disjoint and disjoint categories of the data structures on the basis of the existing research, that the regular disjoint approach is better in terms of providing efficient search time and it provides a better index. However, the storage requirement of these indexes is more as compared to the non-disjoint approaches. The regular disjoint approaches based on quadtree differ on the basis of the type of input data used to build the index. For dealing with curvilinear data consisting of roads [107], a good quadtree based approach that is efficient in dealing with the curvilinear data is required and chosen, for its parallel implementation in the MapReduce.

2.2.2 Non-Disjoint Decomposition Approaches (R-Tree Based))

A detailed survey of the existing traditional spatial indexing approaches has been carried out. The different existing dynamic and static spatial indexes, from non-disjoint decomposition domain, for the serial programming environment are categorized according to the strategies these uses, as shown in Figure 2.2. The motive is to identify a good spatial index, having the potential for parallelization. The non-disjoint decomposition of space partitions the input space into MBRs according to some heuristics, such that spatial objects are allowed to overlap in different MBRs. The disjoint decomposition of space partition input space into disjoint cells, such that spatial objects do not overlap. However, only the R-tree and variant indexes are explored thoroughly in this research work, with regard to parameters, such as space utilization, insertion cost, and query performance for uniform and clustered data. The reason for considering R-tree and variants only is because most of the work for parallelizing the existing indexes in MapReduce is from this

domain.

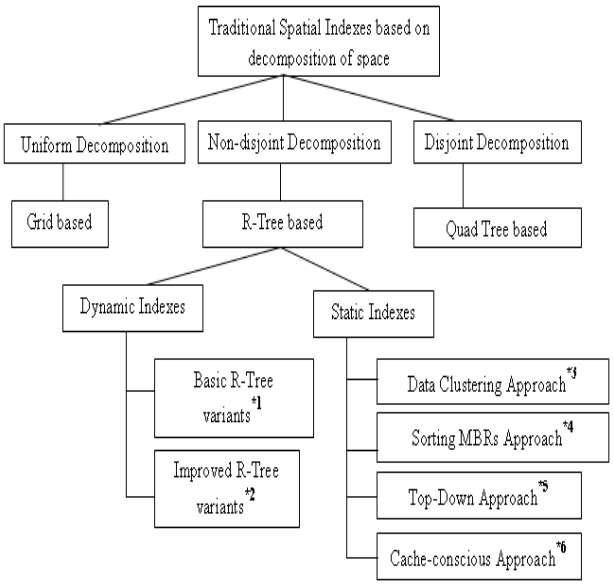

Figure 2.2: Traditional Spatial Indexing Approaches, where #1-[2, 3, 4, 5], #2-[6, 7, 8], #3-[9, 10], #4-[11, 12, 13, 14, 15, 16], #5-[17, 18], #6-[19, 20]

The basic R-Tree variants and the improved R-Tree variants are discussed under dynamic spatial indexes. Just-In-Time (JIT) indexes are examples of dynamic indexes in 2-Dimension that improve the performance of dynamic SQL queries. Theses indexes are created on demand, on the fly to satisfy the requirements of dynamic SQL queries [108].

(i) **Basic R-Tree Indexes:** The basic R-Tree variants include R-Tree [4, 2], R*-Tree [3] and R+-Tree[5]. The R-Tree is the most basic that considers margin, coverage and overlap, as efficiency evaluation metrics. The R*-tree tries to optimize the R-Tree structure with a split and forced reinsert algorithms that preserves proximity of smaller rectangles in a tree node. The R+-Tree also optimizes the R-Tree with its disjoint decomposition.

(ii) **Improved R-Tree Variants:** The improved R-tree variants work towards a better node-split by minimizing overlap among partitioned MBRs. Various spatial index structures, such as RR*-tree [6], WeR-Tree [7] and X-Tree [8] comes under this category.

Four different approaches for realizing static indexes are used: data clustering approach, sorting MBRs approach, top-down approach and cache-conscious approach.

(i) **Data Clustering Approach:** The clustering technique splits spatial objects among nodes on the basis of spatial proximity, according to some parameter to

minimize data access time. The k-means [9] and dimensional-sort curve [10] are examples of this category.

(ii) **Sorting MBRs Approach:** One class of R-Tree indexes is bulk-loaded by sorting MBRs either along one dimension or along both the dimensions in a two dimensional space. The tree leaves are filled-up first and then rest of the index is built step-by-step in a bottom-up manner [16, 12, 11, 14, 13, 15].

(iii) **Top-Down Approach:** One class of R-Tree indexes is bulk-loaded in a top-down manner. The R-tree index is constructed in two steps: firstly, a good partition of data is generated recursively, and secondly, the index is built from root to leaf. The Top-down Greedy Split (TGS) algorithm [105] and the Priority R-Tree [104] are the examples of this category.

(iv) **Cache-Conscious Approach:** The indexing approaches based on the use of secondary memory are used to represent the pointer oriented data structures. The accessing speed of such data structures is slow. The cache-conscious approach represents a fast accessing of data objects through the use of cache memory [19, 20].

The dynamic and static spatial indexes are analyzed for the following parameters. The Table 2.1 and Table 2.2 show the comparison among various dynamic indexes. The Table 2.3 and Table 2.4 show the comparison among static indexes.

(i) **Heuristics Applied:** Most of the indexed approaches towards spatial orientation tree structures work towards optimizing coverage, overlap, margin, and storage utilization. If access time is considered, the parameter for spatial query performance that minimizes overlap is more critical than minimizing coverage. The coverage includes a whole area that covers all related rectangles. The overlap includes an area that is common as a result of intersection of two or more nodes. The margin describes perimeter of a rectangle or combined lengths of the edges of a rectangle. The storage utilization is related to the height of a tree. The most efficient search requires minimal coverage, minimal overlap, minimal margin, and minimal storage utilization. Minimization of coverage in a rectangle-tree justifies a reduction in the amount of empty area covered with nodes. Minimization of overlap justifies a reduction in the set of search paths to tree leaves. Minimizing the margin supports an improvement in the structure with regard to ease of packaging of quadratic objects for the same amount of area. Good storage utilization leads to a reduction in the cost of processing spatial query, as the height of the tree becomes low. The techniques mentioned in this section for spatial index structures, mainly works on one or a combination of more factors for creating index structure.

(ii) **Incremental/Batch Oriented:** The batch oriented method is applied for static dataset, where the input dataset is known in advance before processing. The dataset is packed in an efficient way through applying some heuristic functions, to improve the efficiency of index building and query execution. The development of spatial indexes with incremental method follows a sequential processing of the dataset for bulk-loading an index. The insertion takes place slowly through confirming the position of newly inserted data item. The batch oriented method follows a fast approach. The bulk-loading of the index takes place in batch of data-items. The insertion occurs in batches through some heuristic, such as sorting along a single dimension or along multiple dimensions of MBRs. The batch oriented methodology lowers index construction time and query execution time considerably, and have the potential of parallelizing. The batch oriented methods follow a bottom up approach level by level, and consequently, achieves a high degree of parallelism.

(iii) **Linear vs D-Dimensional Data Distribution:** The linear method of data distribution works along a unique dimension for data insertion in an index. While D-dimensional or multidimensional method can work in multiple dimensions. It uses multi-dimensional split procedure than the linear split procedure for realizing an index. Though, linear packing methods are fast, but the D-dimensional approach better packs data. It takes into account positions and spatial extents of objects in all dimensions that are achieved by a linear method.

(iv) **The Input Data Distribution:** The nature of the data distribution in the input dataset affects performance of index construction and query execution. The data distribution can be uniform or skewed. Further, characteristic of the data, such as the point data, line data, polygon data, linear data and multi-dimensional data, affects data processing. The sizes of data, such as MBR, also affect data processing.

(v) **Number of Nodes to be Searched:** The performance of a search query depends on the number of nodes to be searched for extracting output. The number of nodes to be searched depends on the data distribution of indexed algorithm. If the algorithm allows overlapping of data objects among nodes, then number of searches are required. However, if the algorithm provides a disjoint decomposition of the dataset, then, less number of searches are required. The improved R-Tree variants provide a better index structure by minimizing overlapping among intermediate nodes.

(vi) **Applicability/Performance in Higher Dimensions:** It is not necessary that one algorithm that works well for a linear or low dimensional data, works fine for higher dimensional data. In high dimensional data, the problem is the overlap

among MBRs due to less freedom for splits in the directory. A wrong split axis leads to unbalanced partitions. The index structures generally work on a hierarchical and linear organization of the dataset. The reason is that hierarchical is most suited for low dimensional spaces where overlap between nodes is very low, and selective space need to be searched for queries. A linear organization is suited for high dimensional spaces where the overlap among nodes is high and the whole space need to be searched for queries.

(vii) **Benchmark:** The performance of the spatial indexing algorithms for various parameters discussed is tested against the state-of-the-art benchmarks. It strongly validates test results.

(viii) **Index Building Cost - Storage Utilization:** The storage utilization describes one of the parameters for quality of the generated index. A good indexing algorithm always consumes less memory in the internal nodes and leaf nodes. However, there lies a contrast between storage utilization and query search performance. The index, that uses a small storage, allows overlap among data objects. The overlap causes the number of nodes to be searched, and hence, decreases search performance.

(ix) **Insertion Cost:** The insertion cost of an index depends on the number of nodes accessed during inserting data objects. It is another metric that measures the quality of the index. It contrasts storage utilization. The extra time spent in processing the logic for better insertion leads to a good structure of the index.

(x) **Query Performance for Uniformly Distributed Data:** The performance of an index against various spatial queries measures the strength of the index structure for a particular type of query. Various spatial queries, such as point query, region query, intersection query, spatial join query and nearest neighbor query are generally used.

Table 2.1: Comparison of traditional spatial dynamic indexes against various parameters

Approach	**Index**	**Heuristic Applied**	**Index Development**	**Splitting Procedure**	**Effect of Data Skew**	**Number of Nodes to be Searched**	**Performance in High Dimensions**	**Benchmark**
Basic R-Tree variants	R-Tree quadratic split [4, 2]	Minimizing coverage and overlap	Incremental	D-dimensional	Decreases	High	Low	Independent evaluation
Basic R-Tree variants	R*-Tree [3]	Minimizing coverage, margin and overlap	Incremental	D-dimensional	Increases	Medium	Low	R-Tree
Basic R-Tree Variants	R+-Tree [5]	Disjoint decomposition	Incremental	D-dimensional	Stable	Low	Low	R-Tree
Improved R-tree variants	Revised R*-Tree [6]	Minimizing perimeter and overlap at all directory levels	Incremental	D-dimensional	Stable	Low	Good	R-Tree, R*-Tree and Hilbert R-Tree
Improved R-tree variants	WeR R-Tree [7]	Partial rebuilding of unbalanced node	Incremental	D-dimensional	Increases and better than R*-Tree	Better than R*-Tree	Better than R*-Tree	R*-Tree
Improved R-tree variants	X-Tree [8]	Supernode approach	Incremental	D-dimensional	ND	Low	High	R*-Tree

ND-Not Determined

Table 2.2: Comparison of traditional spatial dynamic indexes against various parameters (extended)

Approach	**Index**	**Storage Utilization**	**Insertion Cost (No. of disk accesses per insertion**	**Query Rectangle/Enclosure Query**	**Point Query Performance**	**Intersection Query**	**Spatial-Join Query**	**Nearest-Neighbor Query**
Basic R-Tree variants	R-Tree quadratic split [4, 2]	Comparable but less than R*-Tree	More than R*-Tree	Higher than R*-Tree	Higher than R*-Tree	Higher than R*-Tree	Higher than R*-Tree	ND
Basic R-Tree variants	R*-Tree [3]	Comparable but more than R*-Tree, Less than R-Tree	Less than R-Tree	Less than R-Tree	Less than R-Tree	Less than R-Tree	Less than R-Tree	ND
Basic R-Tree variants	R+-Tree [5]	Less than R*-Tree	High	High	Less than R-Tree	ND	ND	ND
Improved R-tree variants	Revised R*-Tree [6]	Better than R*-Tree	Better than R*-Tree but not better than Hilbert R-Tree	Better than all	ND	ND	ND	ND
Improved R-tree variants	WeR R-Tree [7]	Better than r*-Tree	Less than R*-Tree	Better than R*-Tree	Better than R*-Tree	ND	ND	Better than R*-Tree
Improved R-tree variants	X-Tree [8]	Better than R*-Tree	ND	Better than R*-Tree	Better than R*-Tree	ND	ND	ND

ND-Not Determined

Table 2.3: Comparison of traditional spatial static indexes against various parameters

Approach	**Index**	**Heuristic Applied**	**Index Development**	**Splitting Procedure**	**Effect of Data Skew**	**Number of Nodes to be Searched**	**Performance in High Dimensions**	**Benchmark**
Data clustering approach	cR-Tree [9]	Space utilization	Incremental	D-dimensional	Better than R*-Tree	Low	Low	R-Tree and R*-Tree
Data clustering approach	R-Tree by iterative optimization [10]	k-way clustering of data rectangles by iterative optimization	Batch oriented	D-dimensional	Low	Increases with data dimensionality due to clustering	R*-Tree, Hilbert R-Tree and Dimension sort curve R-Tree	Improves with data skew
Sorting MBRs approach	Hilbert R-Tree [12, 11]	Hilbert curve	Batch oriented	Linear	Better than R*-Tree for clustered data distribution	Low	Low	R*-Tree
Sorting MBRs approach	STR R-Tree [14]	Minimization of area and perimeter	Batch oriented	d-dimensional	Decreases linearly and becomes comparable to Hilbert R-Tree for highly skewed data	ND	Low	Hilbert R-Tree
Sorting MBRs approach	Lowx R-Tree [15]	Minimization of area	Batch oriented	D-dimensional	Decreases	ND	Low	Independent evaluation
Sorting MBRs approach	Hilbert curve on a Tree structure [13]	Minimization of overlap	ND	ND	ND	ND	High	ND
Parallel R-Tree construction	Multiplexed R-Tree [109]	Minimum load and uniform spread	Incremental	D-dimensional	Increases	Better than X-Tree	Low	Other single processor multiple disk methods, such as Supernode methos

ND-Not Determined

Table 2.4: Comparison of traditional spatial static indexes against various parameters (extended)

Approach	**Index**	**Storage Utilization**	**Insertion Cost (No. of disk accesses per insertion**	**Query Rectangle/Enclosure Query**	**Point Query Performance**	**Intersection Query**	**Spatial-Join Query**	**Nearest-Neighbor Query**
Data clustering approach	cR-Tree [9]	Much better than R-Tree and R*-Tree	Less than R*-Tree	Outperforms R*-Tree and much better than R-Tree	Equivalent to R*-Tree and much better than R-Tree	ND	ND	Equivalent to R*-Tree and much better than R-Tree
Data clustering approach	R-Tree by iterative optimization [10]	ND	More than R*-Tree and Hilbert R-Tree	Better than R*-Tree and Hilbert R-Tree	ND	ND	ND	ND
Sorting MBRs approach	Hilbert R-Tree [12, 11]	Better than R*-Tree	Comparable to R*-Tree	Much faster than R*-Tree	Much faster than R*-Tree	ND	ND	ND
Sorting MBRs approach	STR R-Tree [14]	Better than Hilbert R-Tree	ND	Better than Hilbert R-Tree	Better than Hilbert R-Tree	ND	ND	ND
Sorting MBRs approach	Lowx R-Tree [15]	Poor than Hilbert R-Tree	ND	Less than Hilbert R-Tree	Poor than Hilbert R-Tree	ND	ND	ND
Sorting MBRs approach	Representing Hilbert curve on a tree structure [13]	ND	Much better than R-Tree	ND	Much better than R-Tree	ND	ND	ND
Parallel R-Tree construction	Multiplexed R-Tree [109]	High	High	Better than Supernode method	ND	ND	ND	ND

ND-Not Determined

The common characteristic of all the discussed approaches is that a uni-processor system is assumed. The original R-Tree of Guttman has the disadvantage in terms of high load time, sub-optimal space utilization, and a poor R-Tree structure. The storage utilization of R*-Tree is better than R-Tree. The insertion cost of R*-tree is comparable to R-tree for uniformly distributed data, but it is much better for clustered data. The better insertion cost of R*-Tree is because of better structure of the R*-Tree that avoids node splitting during insertion. The query performance of R*-Tree is better than R-Tree. Number of nodes need to be searched in R-Tree is more due to a high overlapping of rectangles. The R-Tree optimization metrics require enclosing rectangle to be of larger size and containing maximum number of data rectangles as per the node capacity. This causes assignment of a large number of entries to a node, and consequently, a high overlap among nodes [110]. The query performance of R-tree deteriorates with skewed data as overlapping increases. The query performance of R*-tree is better for smaller query rectangles as compared to large query rectangles. It is due to a split and forced re-insertion algorithms of R*-tree that preserves proximity of smaller rectangles in a node. Building R-Trees with dynamic insertion algorithms provide a significant dead space in nodes and results in bad performance. If the R-Tree is built statically by applying some heuristic technique to pack input data space, then space utilization improves. This kind of heuristic is not possible for data inserted with the dynamic index algorithm. In overall, the performance of R-Tree depends on the coverage and overlap [3].

The R+-Tree consumes more space than R*-Tree, since the R+-Tree stores a spatial object in more than one node to ensure that bounding rectangles are disjoint. The search performance of the R+-Tree in terms of disk accesses is more than 50% over the R-Tree for point queries, due to disjointness of the search space. For point data, zero overlaps can be achieved, but the same thing is not true for regional data. More disk accesses are required for the R-Tree as compared to R+-Tree for point query. The R-Tree performs better for a segment query when the segment density is low, but the R+-Tree becomes better than the R-Tree for higher segment density, as overlapping increases with density. The R+-Tree performs better for the smaller segment query than large segment query and the performance decreases with rising density of large segments, as more splits to sub-segments are required [5].

The Space utilization is improved significantly when data is ordered according to geographic proximity. And better response time is achieved due to smaller trees with higher fanout. A cR-Tree structure generated with k-means clustering, using average silhouette coefficient, is similar to the R*-Tree structure. The low index building time of the cR-Tree is due to significant time saving on following a simple insertion algorithm as compared

to the R*-trees [9]. The cR-Tree performs equivalent to the R*-Tree and much better than the R-Tree, for range query and nearest-neighbor query. A WeR-Tree uses a packing technique, to organize its structure better than the R*-Tree. This packing causes data points to be stored uniformly in leaf nodes that lead to fewer activated paths for queries. Though, the WeR-Tree takes a significant amount of time to bulk-load and reconstruct the sub-tree in unbalanced node, but still it is better in comparison to the complex insertion method of the R*-Tree for reorganizing the structure. The R*-Tree does not store data points uniformly in leaf nodes as compared to the WeR-Tree that leads to bad packing in the R*-Tree. These bad packing increases with the density of the dataset, and consequently, more paths are activated. Hence, the performance decreases with scalability in the R*-Tree. The WeR-tree performs better against the R*-Tree for number of nodes access as a performance parameter for range and nearest-neighbor queries, update operations, mixed workloads, scalability, storage overhead, and cache performance [7]. An X-Tree is much better for indexing and querying in higher dimensional datasets. It is difficult to get an overlap-free node split with the R-Tree variants in higher dimensions. The portion of data space covered by more than one MBR is very high in the R*-Tree for higher dimensions. The X-Tree index with its overlap-free split and the supernode methods tackle higher dimensional dataset very well [8].

The clustering method for constructing the R-Tree in higher dimensions is compute intensive as compared to the R*-Tree and the Hilbert R-Tree, but it performs better than the two for queries. It is because the latter two assign rectangles from different clusters to same R-Tree node. The earlier indexing methods, based on either space filling curves (SFC) or the R-Tree variants, had some limitations. The SFCs, such as Hilbert and Z-Order, do not preserve spatial locality as well. The R-Tree variants do not exploit the known dataset during insertion. The performance gain increases with data dimensionality and skews. Between the R*-Tree and Hilbert R-Tree, the former is better for uniformly distributed data and the latter is better for clustered data [10]. The RR*-Tree is better than the R-Tree variants due to its overlap optimization at all directory levels. It becomes better for high dimensional space due to a good balance maintaining splitting algorithm and perimeter based optimization. The overlap and perimeter based optimization is more compute intensive as compared to the Hilbert R-Tree for insertions. The RR*-Tree insertion is better than the R*-Tree and requires less leaf accesses, as it considers locality insertions. The structural organization of the RR*-Tree is much better than the two and outperforms other R-Tree variants for queries. The RR*-Tree has strong advantages for systems where standard concurrency protocols work as insertions take place along a single path in RR*-Tree [6]. An overlap free tree node structure of the Hilbert curve of a particular order was provided by [13]. However, it is impractical to store the mapping

of space filling curves to a tree representation explicitly. The traversal from the root to leaf would take excessive node accesses.

The dimensional sort based Lowx Tree provides a good solution for point queries, but performs poorly for large query, such as range query, and the efficiency decreases with data skews. This is due to poor grouping of the dataset, as only one side of the rectangles determines the grouping. The packing produces rectangles with minimum area, but with significantly large perimeters, that results in more node accesses for range query [15]. The Hilbert R-Tree based packing preserves spatial locality in the R-Tree nodes, and minimizes area and perimeter of MBRs in leaf nodes [12, 11]. It leads to higher space utilization in the Hilbert R-Trees. The Hilbert R-Tree outperforms the quadratic R-Tree, the Lowx Tree, and the R*-Tree for point and range query. The efficiency of the Lowx Tree is worst. The performance of the R*-Tree is better than the quadratic R-Tree. The response time of a range query is affected by the time taken to retrieve the nodes touched by the query and the time taken to process the node. The packing method shows higher performance over skewed data, 58% improvement over other packing methods and 36% better than the best R-Tree variants [11]. The Hilbert R-Tree performs better for all types of data than the R*-Tree in terms of the number of node accesses [12]. The Hilbert R-Tree achieves high space utilization but insertion time is comparable to the R*-Tree. The STR R-Tree provides significantly smaller area and slightly better perimeter than the Hilbert R-Tree. The space utilization of the STR R-Tree is better than the Hilbert R-Tree. The Hilbert R-Tree takes 30-40% more time than the STR for different node capacity and for point and range queries. The query performance of the STR decreases with an increase in query size and with increase in data density and becomes comparable to the Hilbert R-Tree [14]. The response time of the MR index is better than the STR index for window queries. The I/O cost of the MR-Tree is lower as compared to the STR-Tree. Two disc accesses take the search to a local STR R-Tree that reduces the search space and the number of node accesses in the MR index [16].

The cache conscious indexes perform better than the others. The results for update performance has revealed that the Hilbert R-tree is better than the R-tree and the R*-tree. The Hilbert CR-tree is better than the CR-tree and the CR*-tree due to the better ordering provided by the Hilbert curve. The Hilbert R-tree is better than the Hilbert CR-tree due to the overhead of maintaining QRMBR. The CR*-tree is worst due to high computational overhead in the R*-tree insertions. Increasing the node size increases the fan-out and consequently the cost of reading a node, but at the same time decreases the overall search cost by reducing the tree height [19].

2.3 Grid-GIS Frameworks

Integrating spatial databases with Grid Technology provides a solution to the huge volume of spatial data storage that is generated by the development of GIS technology. GIS heterogeneity and interoperability are resolved with the developing GIS functions as services, and publishing the services on the grid [111]. The grid platform produces consistent results for spatial queries, such as buffer analysis and polygon merge queries. However, the efficiency of the process is affected by data transfer overhead among data providers, servers, and clients.

Many theoretical Grid-GIS design and architecture had been proposed in the past [112, 113, 114]. However, few researchers have practically implemented Grid-GIS. A prototype for raster Geo-spatial data [115] and a Grid-enabled Urban-CA GIS framework for GIS data on Globus Toolkit 4 [116], are among the implementations that demonstrate Grid technology to be a promising technology for efficiently handling GIS data.

In this section, a survey of various Grid-GIS design, architectures and frameworks based on different technologies are presented.

2.3.1 OGSI, WSDL, and XML Based Grid-GIS

This section describes technologies that help in providing a transparent, effective and reliable access to homogeneous/heterogeneous spatial data distributed in grid nodes of a Grid-GIS. The role of these technologies is basically to provide a storage for spatial data, a platform for publishing and accessing of spatial data, and protocols for resource finding, monitoring and messaging between services in grid nodes.

A Grid-GIS Spatial Data Integration System (Grid-GIS SDI) integrates GIS spatial databases, such as a relational database system (Rational), XML files (Xindice) and binary file (BinX) format database, and improved form of the Open Grid Services Architecture Data Access and Integration (OGSA-DAI) [117] middleware in the grid environment [118]. The transparent spatial data interoperable platform is implemented in a GT 3 grid environment. The improved middleware provides a transparent, effective and reliable access to data source. It is implemented through a dynamic identification and registration process of the data source. It includes a memory buffer between middleware and data source to improve the speed of data search and data connection. It also provides functions for direct access to spatial data source and expressing SQL queries. It provides functions to get automatic update, for a change in data or data address.

In another XML based approach, a data integration middleware consisting of two modules, a dynamic storage agent and an XML based virtual database, are used for spatial data integration in Grid environment [119]. The former provides interfaces for access to heterogeneous spatial data resources and registration mechanism, and a mechanism for converting the former to latter through a mapping and convert module. The latter provides a homogeneous spatial data resource for being used in the Grid. The regular Light Weight Directory Access Protocol (LDAP) of grid is used for mapping physical resources to a mapping table and for creating meta-data during the registration process. This method provides a strong support to GIS inter-operation and seamless data sharing.

The regular LDAP in grid system was incompatible for spatial data. It was also unable to support frequent updates to data and services in Grid-GIS. A Grid-GIS prototype, GeoGrid, implements a relational model based global directory and a global MBR decomposition based distributed query processing strategies [115]. The modified, distributed query processing includes heuristic principles of MBR decomposing strategy. GeoGrid is composed of Resource layer, OGSI protocol layer, Geospatial grid extension layer, and Grid application layer. The Resource layer provides geospatial data storage and services to access geospatial data distributed across grid nodes. The data are uniformly transformed into GML format to handle interoperability issue. The OGSI protocol layer provides basic grid protocols, such as resource finding, monitoring and messaging between services in the grid. The OGSI layer is combined with Web Services Technologies, such as SOAP/WSDL, when following the OGSA architecture. The geospatial grid extension layer is concerned with generating queries and invoking services in grid nodes. The application layer provides results of operations executed on geospatial data. However, the GeoGrid requires validation for complex geospatial applications.

In another SIG data grid, the GML is widely used for providing spatial data inter-operate in Grid-GIS. A meta-data adapter bridges the difference between meta-data producer agencies and meta-data consumers in the SIG data grid, for providing interoperability through standards, such as GML 3.0 specified by OGC [120].

In other research work, a functional model of Grid-GIS considers GML, Middleware and Web services [121]. It provides spatial data distribution and functional services for GIS distribution. The GML is the message carrier in the Grid of Web and supports interoperability. The Middleware constitutes a design of message middleware, transaction processing (TP) middleware, GIS functional middleware, and data middleware. The message middleware provides APIs for application deployment, efficient queue for storing request and reply messages, session management functions for processing messages, and

multiple protocols to support heterogeneity. The TP middleware responds to requests from users, manages logs, security and, mission distribution and load balance. The GIS middleware provides GIS functional services and returns the processed results, from and to the TP middleware, respectively. The data middleware provides data format conversion functions to support heterogeneity. However, the metadata model, using GML to describe complex objects and data information transmission, is not discussed.

A reliable and accurate synchronization of large volumes of spatial data is achieved through methods, Message server queue mode, pull and push mode of sending messages and Extended GML. It achieves geographical information synchronization in GRID-GIS for a stable and efficient data delivery in an unpredictable network environment [122]. A single message server is equipped with a queue to provide a reliable message delivery during breakdowns. The pull and push modes of message delivery confirms synchronization of data source and centralized message server during data updating. Both modes are adopted to lower the frequently occurring time-costing operation and to guarantee update. The extended GML format controls large volumes of spatial data through data fragmentation, into small, flexible and linkable units, to improve sending and receiving of messages and providing reliable, accurate and unambiguous, sending again and assembly of data fragments.

2.3.2 OGSA and WSRF Based Grid-GIS

A Grid GIS architecture built on GT 4 and WSRF technology does not require any change to the OGSA-DAI middleware [123]. It comprises of a Grid portal, Grid Geospatial Data Service, Grid Metadata Directory Service, OGSA-DAI Data Service, WSRF specifications, and Grid security certification in the grid environment GT 4. The architecture improved accessibility, operational flexibility and scalability through Grid GIS services; however, the research on semantic analysis of GIS services for GIS Grid services collaboration proposed an improved system efficiency.

In another WSRF based Grid GIS architecture, WSRF is used for organizing vector map services built on Globus Toolkit 4.0 [124]. The method permits flexible usage of the grid resources and reduces effort for configuring and maintaining grid nodes. It also improves access time of GIS Server for queries, by buffering frequently accessed information on resources. The virtual organization (VO), as shown in Figure 2.3, consists of five kinds of node: Grid Portal, Certificate Authorization (CA), Domain Manager, Global Resource Catalog Server, Medial Node, and GIS Servers.

Grid users submit a global job to Domain Manager through a Grid portal. The authors

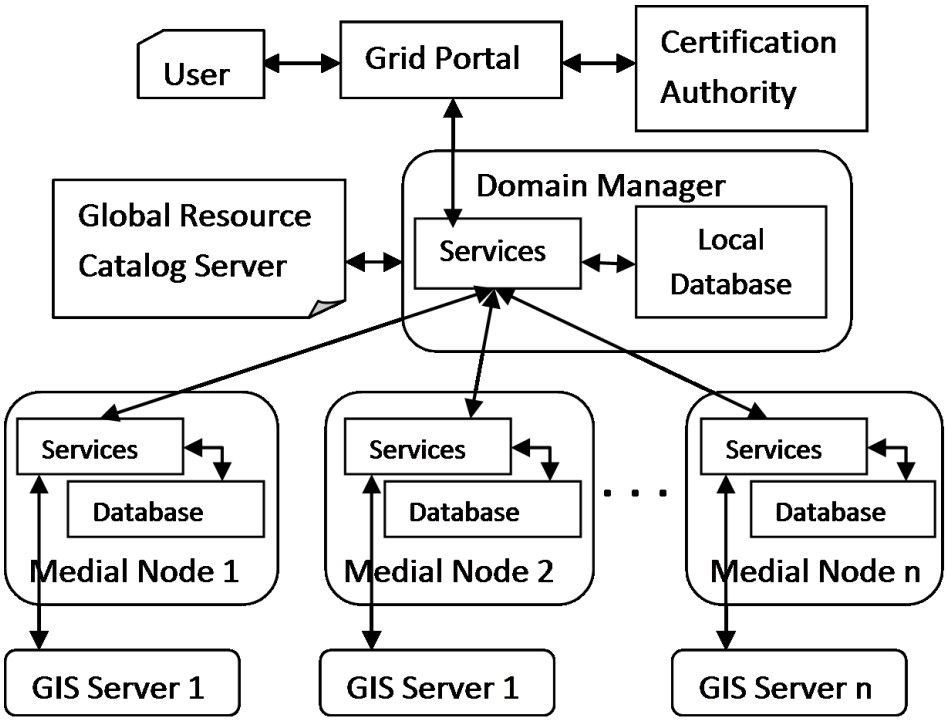

Figure 2.3: An OGSA/WSRF based Grid-GIS VO [21]

segregated Global Resource Catalog Server that manages global resource catalog and Domain Manager that manages the global job parsing and distribution. It manages load balancing and single point of failure over a simple model in which the two modules were put together [125]. Domain Manager parses the global job into a series of subtasks and transfers these subtasks to corresponding Medial Nodes. Medial Nodes complete their subtasks and transfer modified result as per authorization and authentication of Grid users to the Domain Manager. The results are merged and the final result is transferred to Grid Portal for display. The approach kept GIS servers out of the VO and provides access to Domain Manager through medial nodes. The medial node provides interfaces for implementing WSRF services such as spatial data retrieval and processing. These interfaces invoke corresponding functions on GIS server. The authors used a strategy to reduce the time of web service invocation on GIS servers. A web service is deployed on all medial nodes that buffer geospatial information of a connected GIS server. So a repeated geospatial query avoids invoking again the interface of the GIS server by the Medial node. A triggering update mechanism updates any change of information on the GIS server. The procedure requires reducing the time of Web Service invocation by avoiding single point of failure of the Global Resource Catalog Server and Domain Manager, and load balancing the Domain Manager.

In another work, the authors presented an OGC compliant and WSRF based Grid-GIS

framework for Getmap service. The paper demonstrates the complex GIS service, implemented through a chain of service invocations, invocation in an efficient and cooperative manner through stateful intermediate invocations as compared to the traditional Open Web Services (OWS). However, the authors identified the requirement of some WSRF functions, such as lifecycle management, security and notification, and improving the high speed data transfer [21].

In another WSRF based Grid GIS model, OGC compliant Grid Directory Service, Geospatial Data Transfer Service and the QoS-based scheduling service is considered [126]. The QoS-based scheduling service selects services on the basis of the static and dynamic qualities of the GIS services besides the dynamic information about the GIS service providers. This mechanism provides services, selection, performance guarantees and tolerates resource failures that lead to better spatial resource sharing and significant service performance. However, The authors used a simple scheduling strategy for a simple GIS application and found a scope of devising sophisticated scheduling strategies for complex GIS applications.

The MapGIS Grid Workflow Framework (MGWF) identified a single point of failure for the GT 4 based Monitoring and Discovery service [127]. It provided a solution to the problem through the MDS backup node list, higher nodes active polling scheme and member nodes incremental updating scheme. The integration of heterogeneous GIS services and invocation performs well for load balancing and higher throughput. However, the interaction between GIS services and secure Grid services remains to be explored.

2.3.3 Mobile Agent Technology Based Grid-GIS Frameworks

The mobile agent technology suits to mobile Internet applications as it requires low communication bandwidth and supports asynchronous task execution. An architecture of distributed GIS based on Mobile Agent was presented to solve the problem of limited communication bandwidth and unstable connectivity for GIS applications under Internet environment. The architecture used Concordia as the platform to establish distributed environment and to accomplish mobile agents move and collaboration [128].

In one such approach, the authors discussed the issue of data distribution in WebGIS and proposed grid computing and mobile agent based web services as a solution to the problem [129]. In a similar work, the authors presented a distributed mobile GIS grid model (DMGG). The model is based on mobile agent technology dealing with mobile spatial information in a distributed grid computing. It provides a good load balance, high processing efficiency and less network communication [130].

In one approach, the agent technology is applied to developing intelligent user interfaces, for solving problems where programs interact with each other distributed agent technology, and where programs migrates in the network for solving a problem - mobile agent technology. The agent technology in the field of GIS improves usability of GIS software, users access to geospatial data and services [131]. A distributed GIS in a mobile agent and GML technologies, overcome limitations caused by traditional distributed computing paradigm and heterogeneous GIS data sources, respectively [132]. Various agent environments exist in spatial data processing, such as the Aglets, Concordia, Grashopper and Voyager, for geospatial data management. However none satisfies all the needs of a particular system.

Based on agent technology, the authors proposed an effective integrated grid and agent technology based framework for parallelizing operations in a distributed and heterogeneous environments, where autonomous and intelligent applications can easily be built and deployed. The proposed architecture is an extension of the Globus Toolkit. Master-slave methodology has been proposed to develop the agent based grid computing framework. One master thread (agent) creates one or many slave threads (agents) during execution. The slave thread will report back to the master after finishing the task assigned to it. The proposed framework has successfully demonstrated an efficient Grid Computing environment by exploiting software agent and exploiting platform independence for simple computations through a matrix multiplication problem. The framework can be tested and even extended for complex computation where large grid systems may be installed and agents may run over it. The performance of the proposed framework has not been tested for scalable sized problems and networks. For different problem sizes, results can be taken, analyzed, and compared when more computing nodes are attached to the grid [133].

In one such approach, the advantage of mobile agent technology is used to improve the efficiency of grid services. It provides a dynamic movement to grid services within a network. The services reach to different locations of the distributed data and perform their tasks. This minimizes network traffic caused due to transfer of huge amounts of data, and hence guarantees performance. It provides a good load balance, high processing efficiency and less network communication. Mobile police spatial information service grid is one such application of mobile agent technology on the Grid GIS [134].

In another approach, a middleware layer for heterogeneous spatial data integration integrates mobile agent technology and spatial data grid [135]. The mobile agent technology becomes effective for addressing mobility in the nodes where the network nodes take part in computation and provisioning of services. It provides scalability by temporal and spa-

tial distribution of service logic. Various types of agents optimize system performance, such as Agency Agent, User Agent, Native Query Agent (NQA) and Collaboration Query Agent. The Agency Agent interacts with all the agents in an authenticated manner and maintains distributed transparency. User Agent provides a friendly user interface for users query and responses. The NQA is founded by a User agent and communicates with native spatial data services in the data grid. The Collaboration Query Agent, a mobile agent, collaborates with the User Agent and Agency Agent, and moves to a target system for carrying out a query task.

In another agent based approach, a multi-agents based system architecture was proposed by combining innovative approaches from software agents, georeferenced data modeling, and content based image retrieval (CBIR). The CBIR was used for spatial data retrieval, management and update of vast amount of spatial data. The framework was tested against a set of images on a web server hosting eight, 32-megabyte images from the Oak Ridge Reservation area. It resulted in a significant reduction in the time and management effort associated with large amounts of image data [70].

In a similar system, the mobile agent technology is integrated into a Grid-GIS [136]. The Grid middleware manages distributed resources, such as spatial data and computation power, and the mobile agent brings high performance with its ability to collaborate and process tasks. The mobile agent based system provides efficient and flexible services to users. However, management of data resources and the security mechanism is to be considered. There is also a need to analyze the developed system for parameters such as dynamic workload estimation for each running agent and resource utilization of each node of the grid in terms of idle processor time and memory usage.

In another integration, two different programming models were integrated to build a component-based framework. One model was the DG-ADAJ (Desktop Grid Adaptive Distributed Applications in Java), which provided a distributed execution platform, and the other model was MAGDA (Mobile Agents Distributed Applications), which supported programming, and execution of mobile agent based distributed applications. The component based programming facilitated designing of application from existing building blocks, avoiding the development of codes when these already existed. These components supported high-level reuse and were loosely coupled through interfaces [137].

2.4 Integration of Grid and MapReduce

The MapReduce implementation-Hadoop framework, used for developing and running data analytic applications, provides a high throughput access to distributed spatial data. A Grid-GIS provides a large and scalable computational environment for efficient sharing of spatial resources, apart from other grid resources, among users and across applications. The former provides a large volume data analytic environment while the latter provides a high computing environment. There is a common thing between the two; both use a high computational infrastructure. However, both do not exist together, as the users or organizations focus on either the data analysis or the high computation at a time. Due to incompatible environments, the two do not cooperate well and results in conflicts or bad performance when used together. However, integration of the Hadoop and Grid engine provides a solution, and both compliments each other with their features [138]. The High Performance Computing (HPC) can be integrated with any framework that can run under scheduling and resource management [139]. The Hadoop cluster is built over suitable nodes selected by the Grid engine. A data analytic Hadoop job is processed in HDFS data blocks of the cluster.

The benefits of the integration are as follows:

(i) High utilization of resource pool.

(ii) High data analytic feature of the Hadoop is complimented with the comprehensive accounting, resource utilization control and policy management features of the Grid. Data analytic and computational environment coexist.

Many hybrid architectures were proposed in the past that integrates MapReduce on Desktop Grid. In one such integration, the authors proposed a HybridMR model using the MapReduce on the Desktop Grid network for large scale data-intensive computation. The model improved the reliability of the distributed storage in the Desktop Grid through a hybrid DFS consisting of the storage of the cluster nodes and the volunteer nodes. A node priority based fair scheduling algorithm (NPBFS) was used to achieve data storage balance and job assignment balance. The performance difference of the cluster nodes and volunteer nodes was reduced through an optimized scheduler [140]. In a related work, the authors provided an environment for running MapReduce applications on the Desktop Grid using BitDew [141]. The BitDew is a programmable environment for automatic and transparent data management on computational Desktop Grids [142]. The main features of the hybrid architecture were reduced latency, better fault tolerance, collective file operations, barrier-free computation, reduced lagger effect, decentralized result certification.

The feasibility of the hybrid infrastructure consisting of either the integrated MapReduce framework and Desktop Grid or the integrated MapReduce framework and Cloud was proposed [143]. In a similar hybrid infrastructure, the authors presented node availability prediction method as a solution to the volatile computing nodes of the Desktop Grid [144, 145]. A hybrid architecture, MOON, provided reliable MapReduce services using adaptive task and data scheduling algorithms in Hadoop [146].

2.5 MapReduce-based GIS Processing

The MapReduce programming model offers a new distributed environment for parallel processing. It is a loosely coupled architecture where computing nodes can be added or removed on the fly to the cluster and provide scalability. In the cluster, one node acts as master node and others work as slave nodes. The cluster makes a distributed file system that harnesses the storage capacity of all nodes in the cluster. The simplicity, ease of use and ability to easily handle large datasets, ease of programming by using the Map and Reduce functions, the fault tolerance nature of MapReduce model and the property of abstracting parallelization from users have motivated researchers to use the model. It provides a good performance through scaling out to computing clusters. The survey work for spatial query processing approaches in MapReduce has considered spatial query processing approaches in MapReduce based on a hierarchically indexed and key-value storage based indexed approaches. Both the approaches use different spatial data partitioning methods in MapReduce environment, as is shown in Figure 2.4.

2.5.1 Hierarchically Indexed Dataset in a Cluster

The hierarchically indexed dataset represents a hierarchical tree structure, generally based on R-tree and variants. The index building is carried out in three steps: partitioning input dataset on cluster nodes using an appropriate partitioning algorithm, building a local index on partitioned data in cluster nodes and merging all local indices. Various spatial data partitioning techniques over hierarchically indexed dataset in MapReduce are employed, such as sampling based uniform partitioning, space-filling-curve based data partitioning and quadtree-based spatial data partitioning. The hierarchical tree structures such as R-Tree and variants are good for queries that access only a particular part of the dataset, such as range search and region search.

(i) **Sampling based uniform partitioning:** It is done by the division of the space

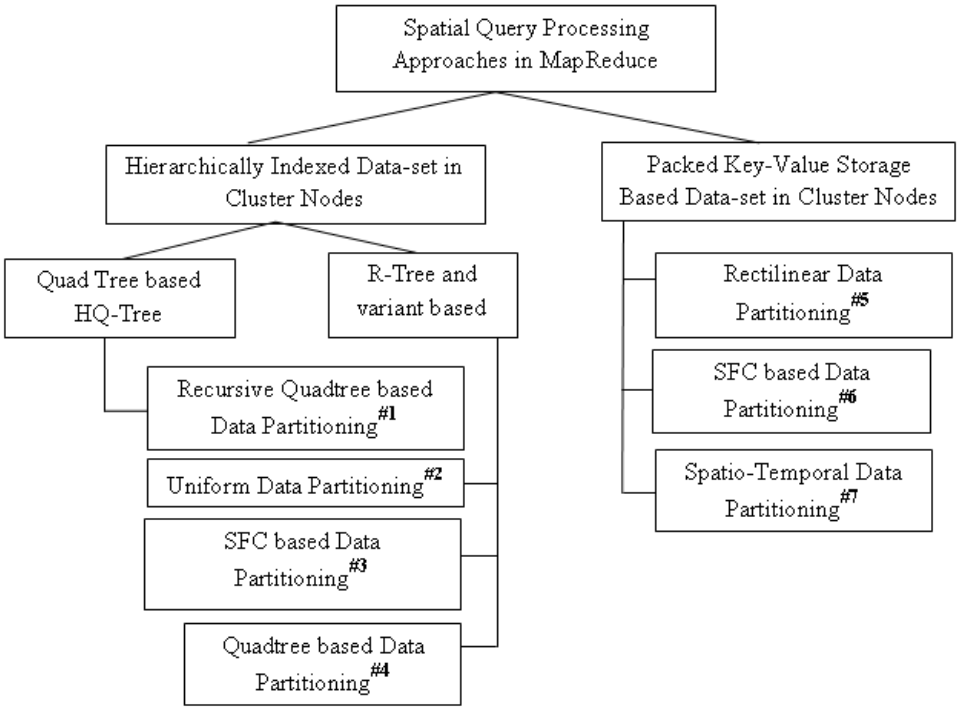

Figure 2.4: Spatial Query Processing Approaches in the MapReduce, where #1-[22], #2-[23, 24, 25, 26], #3-[27, 28, 29, 30, 31, 32, 33], #4-[34, 35, 36], #5-[37], #6-[38, 26, 39] and #7-[40]

into n rectangles and the data is assigned to the overlapping rectangles. The number and size of rectangles are decided by the number of partitions required for the input data. The number of partitions (n) is calculated by dividing the input data size with HDFS block size [24, 26, 22]. The uniform or rectilinear space partitioning approach is easy to implement, but it causes non-uniform data distribution among cluster nodes for processing in MapReduce, especially for non-uniformly distributed and skewed dataset. This affects the load balancing and hence the efficiency for queries. Some of the nodes complete their task early and sit idle, waiting for other heavily loaded nodes to complete their tasks.

(ii) **Space-filling-curve based data partitioning:** The space-filling curve is used to transform multi-dimensional location information into one-dimensional space. A Z-curve based space-filling curve in MapReduce is used for data partitioning during the map-phase [28, 27, 30]. A similar approach for bulk-loading R-Tree over MapReduce using the Hilbert curve is used [147, 33]. The space-filling-curve based data partitioning approaches looses on preserving spatial locality due to a mapping from high-dimension to 1-dimensional space, however the approach works better for high dimensional data.

(iii) **Quadtree based spatial data partitioning:** The quadtree-based data partitioning preserves spatial locality of objects and provides a uniform decomposition of space that makes it highly suitable for parallel processing, but these are difficult to apply for higher dimensional data. One class of MapReduce based approaches for constructing R-Tree used quadtree-based space partitioning [36, 34, 110]. A different technique, HQ-Tree, uses a recursive regular quadtree partitioning for handling point data [22].

Table 2.5: Comparison of MapReduce based hierarchical index structures

Data partitioning for load balancing (Global node) + Indexing on local partitioned dataset	**Data partitioning: Data skew/Load balancing**	**Number of nodes**	**Bulk loading time/Index creation time**	**Execution time for simple spatial queries**	**Performance for simple kNN queries**	**Inter query and Intra query parallelism**	**Benchmark**
Two tier indexing (Quadtree based Global index, Hilbert order based linear local index) [36]	ND	17	Low	Low	stable and better performance for increasing k	Yes	Postgre SQL cluster, bare HDFS, Cassandra and HBase
Uniform sized partitions, R-Tree on Block data from one relation. No indexing on 2nd relation [26]	ND	ND	ND	ND	Better	ND	H-BNLJ
Hilbert Curve for partitioning spatial data and R-tree/Quad-Tree based local indexing [147]	ND	10	ND	Low	ND	ND	Oracle Spatial
non-indexed: tile to bucket mapping with SFC [38]	ND	8	ND	ND	ND	ND	Oracle Spatial and Postgre SQL
R-Tree index construction with Hilbert curve based spatial data partitioning [33]	ND	9	Low	ND	ND	ND	ND
Hilbert and STR packing for constructing R-Tree [29]	ND	9	Low	ND	ND	ND	ND
Z-order based linear data partitioning for constructing R-Tree [28]	ND	2 to 64	Low	ND	ND	ND	Varying number of Reducers and Single Processor system
Z-order based data partitioning and X-means based data partitioning [27]	ND	400	Low	ND	ND	ND	Two data partitioning methods

to be cont'd on next page

Table 2.5: Comparison of MapReduce based hierarchical index structures (Cont.)

Data partitioning for load balancing (Global node) + Indexing on local partitioned dat aset	**Data partitioning: Data skew/Load balancing**	**Number of nodes**	**Bulk loading time/Index creation time**	**Execution time for simple spatial queries**	**Performance for simple kNN queries**	**Inter query and Intra query parallelism**	**Benchmark**
Recursive Tile based data partitioning approach and R*-Tree based RESQUE [34, 35]	Better	8 nodes of 24 cores each	Low	ND	ND	ND	Brute force approach with no indexing, parallel SDBMS - DBMS-X
Hilbert Curve for partitioning spatial data and R-tree based local indexing [30]	Good	6 nodes	ND	ND	ND	ND	Sequential Process construction of R-Tree
QuadTree based Global partitioning and PR QuadTree local indexing [22]	Excellent	5 nodes	Low	ND	ND	ND	Stand alone system

ND-Not Determined

Table 2.6: Comparison of MapReduce based hierarchical index structures (extended)

Data partitioning for load balancing (Global node) + Indexing on local partitioned dataset	**Latency for random access for large number of concurrent reads**	**Latency for sequential access for large number of concurrent reads**	**Communication(Data transfered/Shuffled)**	**Effect of cluster scaling on query execution**	**Effect of index node size**	**Effect of data packet size**	**Performance for uniformly distributed dataset**
Two tier indexing (Quadtree based Global index, Hilbert order based linear local index) [36]	Low	Low	ND	Considered decreasing	ND	ND	ND
Uniform sized partitions, R-Tree on Block data from one relation. No indexing on 2nd relation [26]	ND	ND	Low as compared to H-BNLJ	Considered decreasing	ND	ND	ND
Hilbert Curve for partitioning spatial data and R-tree/Quad-Tree based local indexing [147]	ND	ND	ND	ND	ND	ND	ND
non-indexed: tile to bucket mapping with SFC [38]	ND	ND	ND	Considered decreasing	ND	ND	ND
R-Tree index construction with Hilbert curve based spatial data partitioning [33]	ND	ND	ND	ND	ND	ND	ND
Hilbert and STR packing for constructing R-Tree [29]	Considered	HDFS data transfer protocol is optimized for streaming I/Os	Considered	ND	Considered	Considered	ND
Z-order based linear data partitioning for constructing R-Tree [28]	ND	ND	ND	Considered decreasing	ND	ND	ND

to be cont'd on next page

Table 2.6: Comparison of MapReduce based hierarchical index structures (extended) (Cont.)

Data partitioning for load balancing (Global node) + Indexing on local partitioned dataset	**Latency for random access for large number of concurrent reads**	**Latency for sequential access for large number of concurrent reads**	**Communication(Data transfered/Shuffled)**	**Effect of cluster scaling on query execution**	**Effect of index node size**	**Effect of data packet size**	**Performance for uniformly distributed dataset**
Z-order based data partitioning and X-means based data partitioning [27]	ND	ND	ND	ND	ND	ND	ND
Recursive Tile based data partitioning approach and R*-Tree based RESQUE [34, 35]	high for reading data	ND	High	Considered decreasing	ND	ND	ND
Hilbert Curve for partitioning spatial data and R-tree based local indexing [30]	ND	ND	Considered and found high as compared to sequential process	ND	ND	ND	ND
QuadTree based Global partitioning and PR QuadTree local indexing [22]	Considered	ND	Low	Considered decreasing	Considered	ND	High

ND-Not Determined

Parameters for comparing MapReduce based hierarchical index structures are as follow -

(i) **Data Partitioning for Load balancing (Global Code) + Indexing on local partitioned dataset:** The partitioned dataset for parallel processing on the cluster is managed or tracked by creating indexes. The distributed data between the datanodes is managed through a global index by the master node of the cluster. The partitioned datset on the datanodes also indexes for efficient accessing and it is termed as local indexing.

(ii) **Data Partitioning - data skew/load balancing:** The process used to partition the large volume of spatial data so that the partitioned dataset can be operated upon in parallel by the mapper and the reducer functions in the MapReduce. A uniform data partitioning that distributes the input dataset into uniform size partitions (each partition carries the data of almost similar size) is considered the best. If the partition function does not distribute the input data into partitions of the same size, then some of the datanodes completes the assigned task earlier than the other datanodes. The datanodes that completes the task sits idle, waiting for the other datanodes to also finish their tasks. It causes improper utilization of the cluster resources.

(iii) **Number of cluster nodes considered:** The number of computing nodes used in the cluster for accomplishing a task measures the size of the cluster. The possibility of parallelization of task execution increases with the number of computing nodes.

(iv) **Bulk loading time/Index creation time:** It is the time used to create a complete index of the given dataset. It is a static operation and used only when the dataset is known in advance. In case of unknown dataset, dynamic index is created by inserting data elements one-by-one.

(v) **Execution time for simple spatial queries:** It defines the time needed to accomplish a spatial query that is considered as the basic structural queries in spatial databases. The time is counted since the submission of the job query to the results returned.

(vi) **Performance for kNN query:** Apart from the simple spatial queries, some complex queries are formed when a number of simple spatial queries are performed in sequence. The output of one query serves as the input of the other. Similarly, some queries require a large number of computations such as the join queries. The spatial query to find the k nearest neighbor (kNN) also comes in the category of the complex join query.

(vii) **Inter-query and Intra-query parallelism:** The execution of a query requires the data. Now, in the distributed system, a task may get some part of the data on the local node and the remaining data may be available on the other distributed nodes. The multiple map or reduce functions operating in parallel with the local data is considered intra-query parallelism. If the multiple map or reduce functions parallelize sub-task execution on the distributed data at different nodes, then it is considered as inter-query parallelism.

(viii) **Benchmark:** The work done by the various researchers is tested against other similar existing works to measure the quality of outputs generated and the efficiency of the process used.

(ix) **Latency for random access for large number of concurrent reads:** It is the amount of time taken for executing the parallel spatial queries in the MapReduce that require to access the data randomly.

(x) **Latency for sequential access for large number of concurrent reads:** It is the amount of time taken for executing the parallel spatial queries in the MapReduce that require to access the data sequentially.

(xi) **Communication (data transfer/shuffled):** The number of intermediate data generated as a result of the Map functions causes data to move to the Reducer functions and subsequently, a large number of data shuffling takes place that measures the communication overhead towards completing a task. A high communication overhead brings down the efficiency of the query execution.

(xii) **Effect of cluster scaling on query execution:** The MapReduce in a cluster uses parallelization to accomplish a task. The master node distributes the task into sub-tasks and the datanodes in the cluster get the sub-tasks and later on after completing the sub-tasks return the results to the main master node. The number of nodes determines the scalability feature of the cluster. Generally, the execution time of a task is reduced with an increase in the number of computing nodes. However, after a particular stage, the cluster stabilizes and usually no further gain in efficiency is observed, as the number of data transfer and shuffle overweight the benefit of parallelization.

(xiii) **Effect of index node size:** The index node size is the amount of data items that can fit in the data structure nodes of the index. These structure nodes can be the leaf nodes or the intermediate nodes that act as the bounding box for the data it contain. The data items are accessed through pointers. The pointers can point to leaf nodes or to the next level of intermediate node. The size of the index

node affects the efficiency of the index building as well as the efficiency of the query execution. An increase in the size of the data structure node reduces the height of the tree and consequently the disc access time. However, at the same time, the search time inside the node is increased, which is usually sequential.

(xiv) **Effect of data packet size:** The data packet size determines the amount of data that can move as a single unit during the data transfer, as a result of some query operation. A very small size of the packet causes multiple packet construction and movement over the network. Each packet also carries some supporting or managing data that describes about the packet itself. If more packets are constructed for a dataset then it may be possible that the size of the supporting or managing data become comparable or significant to the original or fruitful data and hence a decreasing performance. However, if the packet size become too large then the latency of moving data and the chances of its failure or crash during data transfer increases.

(xv) **Performance for uniformly distributed dataset:** The performance measurement of the index building or the spatial query execution depends on the distribution of the data in the input dataset. The input data can be uniformly distributed, non-uniformly distributed or skewed. Now, for each category, the data rectangles can be of approximately similar sizes or the size variation can be quite high. The aspect ratio term is used that measures a different kind of size for the data rectangles. It is the ratio of the width to the length of a rectangle. The term data rectangles are being used here because the spatial data is usually represented with its approximation such as the MBRs. It is so because the spatial operation application of the original data is a quite difficult task.

The parallel construction of spatial indexes and parallel query execution are fast as compared to the conventional serialized approaches. Parallelization with a single processor and multiple disks for querying R-Tree performs better as compared to a single disc system [109]. A parallel R-Tree construction using multiple disks and multiple processors on a shared nothing architecture [148] incurs more I/O on nodes leading to high communication costs. However, bulk loading time is reduced due to parallel construction of the tree. Scan and Monotonic Mapping based parallel computation on hypercube architecture uses multiple processors to construct in parallel the index tree structure on a single disc. The MapReduce is the latest parallelization model that uses a cluster consisting of multiple nodes. One node act as the master node, others work as slave nodes. The cluster makes a distributed file system that harnesses the storage capacity of all the nodes in the cluster. Parallelization is achieved with map and reduce functions. Latency for sequential accesses

for a large number of concurrent reads in map-reduce environment does not affect much. However, latency for random reads in map-reduce environment is poor as the HDFS data transfer protocol by default is optimized for stream I/Os. MapReduce in general and with spatial data is best for compute intensive tasks, but not that much suitable for random read tasks [34, 35]. The use of spatial indexes over MapReduce strongly improves the latency for random reads [36, 29].

The most of the MapReduce based spatial indexing and querying system is based on the R-Tree and its variants for spatial indexing. All parallel systems use two-tier partitioning so that the local proximity of objects in partitions can be maintained. The approaches differ in the partitioning method employed for partitioning the input dataset. Many are based on a space filling curve based partitioning. Various techniques employed for the MapReduce based R-Tree construction are such as sampling based uniform distribution based partitioning [26], R-Tree construction with Hilbert curve [147, 29, 33, 30], R-Tree construction with STR packing [29], R-Tree construction with Z-curve [28, 27], R-Tree construction with X-mean algorithm [27].

The Quadtree based approach is difficult to apply for higher dimensions. The Space Filling Curve (SFC) based data partitioning approaches looses on preserving spatial locality due to a mapping from the high dimension to 1-dimensional space. This approach works better for high dimensions also. A quadtree based recursive tile partitioning for R*-tree indexing [34, 35], a quadtree partitioning and Hilbert curve based local indexing [36, 147], quadtree based partitioning for R-Tree and R+-Tree [24], Quadtree based indexing for PR Quadtree based local index [22].

Not much work has been done towards parallelizing traditional spatial index methods on MapReduce like WeR-Tree [7], X-Tree [8], cR-Tree [9], R-Tree with iterative optimization [10], Revised R*-tree[6], Quadtree and its variants such as PMR Quadtree [149]. It has been analyzed that data skew is handled quite efficiently in map-reduce as compared to SDBMS environment [34, 35]. Further, better results for R*-Tree index construction and querying is obtained when the sampling number is increased or in other words number of records per data size is decreased for recursive tile partitioning [30, 148]. Excellent data skew handling is achieved with quadtree partitioning for constructing PR Quadtree [22].

Some of the recent researches on MapReduce based spatial index building and spatial query execution compare the results with parallel SDBMS such as Postgre SQL and Oracle Spatial [36, 147, 38, 34, 35]. Other researches compare results with key-value based storage system such as bare HDFS, Cassandra, and HBASE [36, 28] and with sequential processing system on standalone machine [28, 29, 33, 30, 22]. Analysis and

comparison of building traditional spatial index methods and query execution based on the index is available but not much analysis and comparison of the same has been found over MapReduce environment.

The Quadtree based data partitioning preserves spatial locality of objects and provide a uniform decomposition of space and making it highly suitable for parallel processing. Quadtree and variants have not been used much for spatial indexing over MapReduce. Quadtree structures over MapReduce has been used for data partitioning [36, 147, 24, 22] and the partitioned data is indexed with Hilbert curve [36, 147], R-Tree [4], and R+-Tree[5]. Until recently, most MapReduce oriented researches on spatial indexes have been based on R-tree and variations [147, 28, 33, 27, 26, 30] although, the performance of Quadtree variants based indexes for index building and query processing is well established [150, 151, 152, 151, 153, 154, 155].

It has been due to the regular disjoint decomposition approach of quadtree based indexes which takes less index building time and query processing time as compared to the non-disjoint and irregular disjoint approach of R-tree and variants. However, [22] has used PR Quadtree for spatial indexing of point data. The PMR Quadtree index for Spatial join queries performed better as compared to R-Tree, R*-tree and R+-tree [156]. The index building time for PMR Quadtree over SAM parallel model is less than R-Tree and R-+Tree [152, 151]. PMR Quadtree performs better for join queries [151] and polygonization queries [152] than R-Tree and R+-Tree. However, the PMR Quadtree index has not been realized over the MapReduce framework. The PMR Quadtree index for Spatial join queries performed better as compared to R-Tree, R*-tree and R+-tree [156]. The Quadtree based regular disjoint decomposition techniques for spatial data partitioning gives a stable performance for increasing k in kNN queries as compared to other key-value storage systems such as bare HDFS, Cassandra, and HBASE. kNN for map-reduce based R-tree is more efficient than a simple nested loop join over uniformly distributed dataset[26].

Data transfer or communication overhead for index building is more in parallel programmed MapReduce framework [29, 34, 30, 35] and SAM model [150, 154, 151]. It is due to high data communication that take place during the shuffle and merge operations in map and reduce phases in map-reduce systems. In SAM model, a large number of clipping operations are required before determining which part of the line is associated with the two nodes resulting from the split. Among map-reduce systems the data transfer overhead is more with nested loop join systems as compared to the tree index systems [26]. Data transfer overhead has not been determined for other R-Tree variants in map-reduce environment. Data transfer overhead is high for R-Tree in SAM model as compared to

R+-tree and PMR quadtree due to disjoint decomposition. Build time and communication cost decreases for R-Tree build with a rise in bucket size and the communication cost increases for polygonization queries with a rise in bucket size [153, 154, 155].

The basic feature of traditional tree indexes such as dealing with the boundary object problem and the effect of index node size works in a similar fashion over MapReduce framework as has been discussed in Section 2.5. The SAM model based partitioning replicates objects in nodes to avoid overlapping.

2.5.2 Packed Key-Value Storage Based Dataset in Cluster Nodes

A packed key-value storage based index in the MapReduce does not build a hierarchical index, rather, it partitions the input dataset with a proper space partitioning technique and uses the key-value storage of the MapReduce framework as index for spatial query processing. The key-value storage based indexed approaches in the MapReduce for spatial data partitioning and representation are based on rectilinear space partitioning, SFC based space partitioning and spatio-temporal partitioning.

(i) **Rectilinear space partitioning:** The rectilinear partitioning divides the input spatial space into equal size depending on the number of mappers in MapReduce. Each partitioned space is taken by a cluster node and processed. The key-value storage index is applied to the packed partitioned dataset for spatial queries. The uniform or rectilinear space partitioning approach is easy to implement but it causes non-uniform data distribution among cluster nodes for processing in MapReduce, especially for non-uniformly distributed and skewed dataset. This affects the load balancing and hence the efficiency for queries. Some of the nodes complete their task early and sit idle, waiting for other heavily loaded nodes to complete their tasks.

(ii) **SFC based space partitioning:** The space-filling curve based data partitioning approach in MapReduce arranges the original input spatial data-set according to a SFC and partitions the input spatial data-set into blocks of uniform size. A key-value storage index is applied to the packed partitioned dataset for spatial queries.

(iii) **Spatio-temporal partitioning:** The spatio-temporal data is used to represent the spatial objects with respect to time, such as the trajectory of a moving object. It is represented as (x,y,t), where x and y are the coordinates of the object and t represents the timestamp of the object at the specified position.

Table 2.7: Comparison of MapReduce based Packed key-value storage index

Approach Used	**Data partitioning for load balancing**	**Data partitioning: data skew/load balancing**	**Data partitioning: boundary object problem**	**Number of nodes**	**Type of query tested for execution time**	**Dataset used**
Controlled-Replicate Approach [37]	rectilinear partitioning of uniform size is done for the whole dataset. Project, Split, replicate predicates helps in completing Join Query	Determined	Replicating rectangles	4 nodes each of 16 cores	join query	varying data size
[38]	tiles to bucket mapping with SFCs	Considered	pending file structure and redundant partitioning method	6 nodes	ANN	Large
H-zkNNJ [26]	Z-order based linear data partitioning	ND	Considered	17	kNN query shows stable performance with increasing k	ND
[39]	space filling curve based indexing	SJMR: Z-Curve based data partitioning	ND	ND	Spatial join query	ND

ND-Not Determined

Table 2.8: Comparison of MapReduce based Packed key-value storage index(extended)

Approach Used	**Inter query and Intra query parallelism**	**Benchmark**	**Communication (Data transfered/Data shuffled**	**Effect of clustering scaling on query execution**	**Effect of data packet size**	**I/O cost**	**CPU cost**
Controlled-Replicate Approach [37]	Considered	two-way join and All-Replicate Join	Considered	NC , Fixed : experiments run on 64 reduce processes	Not considered	Considered	Considered
[38]	Considered	Oracle Spatial	Considered	Performance increases	Not considered	Considered	Considered
H-zkNNJ [26]	ND	H-zkNNJ [26], H-BRJ	Low as compared to benchmarks	Considered decreasing	ND	ND	ND
[39]	ND	ND	ND	Considered decreasing	ND	ND	ND

ND-Not Determined

The MapReduce based Packed key-value storage index is also compared to the similar parameters that are discussed for the MapReduce based hierarchical indexed dataset described in Section 2.5.1. Besides, the parameters considered in the hierarchical spatial index, the data partitioning for load balancing, the data partitioning - data skew/load balancing, the number of cluster nodes considered, the of query tested for execution time, the dataset used,the inter-query and intra-query parallelism, the benchmark, the communication overhead (data transfer/shuffled), the effect of cluster scaling on query execution and the effect of data packet size, the parameters for dealing with the spatial objects present at the boundary, I/O costs, CPU cost and data transfer costs are also considered.

The packed key-value storage based indexed approaches are similar to the hierarchical indexed approaches for the first part of data partitioning. They differ from the indexed approaches for the second part. While indexing techniques build a local index and later merges all the local indexes, the packed key-value storage based indexed techniques works in the direct ordering or packing of the partitioned data resulted from the first part in map and reduce phases. The key-value storage based indexed approaches sometimes perform better than the indexed approaches under conditions. Hierarchical tree structures such as R-Tree and variants are good for queries that access only a particular part of the dataset such as range search and region search. These do not perform well for queries like Join queries that require a linear access to a dataset for processing. Complex queries which involve two spatial datasets like Spatial join query, the ANN search query, the traditional nested loop methods and its advanced forms that consider proper clustering of input data is better [157, 158, 26]. Better performance has been proven for join queries over indexed or clustered dataset than simple nested-loop join [159, 160, 161]. Different clustering methods such as space filling curve [26], double transformation methods [38, 162, 157, 39], and hashing method [158] have been used.

The spatial-hash method incurs cost of spatial data partitioning and spatial join while the R-Tree index includes tree-matching costs only. However the lower cost of Spatial-hash join is due to the sequential I/O during the join phase while the tree matching phase of R-Tree uses random I/O. Degree of data clustering affects R-Tree based join due to a change in the number of data nodes accessed while Spatial-hash join incurred almost constant cost. Spatial-hash join costs depend mainly on the input data sizes and can be easily estimated and consequently provide ease of query planning and optimization [158].

The Z-order space filling curve based H-zkNNJ performs better than R-Tree for the increasing number of reducers. It is because R-tree takes significant building and querying

time when the amount of reducers increase. It is because size of data blocks decreases and large number of smaller R-Trees are constructed in parallel that consequently increase the building cost. The cost paid for the lower computation and communication in H-zkNNJ is in terms of accuracy of query results as Z-order based SFCs do not well preserve the spatial locality [26]. A similar result was found in the performance of R-Tree against the mapping of quadtree based partitioned objects to locational keys in a double transformation method for semi-operator based join query [157]. The latter performs better than the former for total execution time of spatial join query. The total execution time includes the time spent in I/O, communication, and CPU execution. The poor performance of R-Tree is due to CPU execution time only otherwise I/O cost and communication cost is much lower for RT as compared to Quadtree approach. Lower CPU execution time for Quadtree is because it requires only one scan of locational keys, which makes it very efficient while in R-Tree the process is random. For very fast processor, R-Tree outperforms Quadtree. When bandwidth decreases, the performance of Quadtree decreases fast as compared to R-Tree as Quadtree transmits more data, locational keys and the data. Quadtree is less effective in eliminating objects that dont contribute in join phase. Consequently, false drop is larger for Quadtree and it incurs more data transmission cost and I/O cost. For small datasets Quadtree is better than R-Tree but for large datasets Quadtree performs poorly due to higher communication costs.

A double transformation method that uses tile to bucket mapping with space filling curves, Partition Based Spatial-Merge join (PBSM) performs better for join query than traditional indexed nested loop and R-Tree index on spatial datasets. The PBSM algorithm based join performs better, when neither of the datasets is indexed or a single smaller dataset is indexed. R-Tree based join performs better when either both the datasets have prebuilt indexes or a single larger dataset is indexed. The traditional nested loop algorithm performs worst for smaller datasets, but comparable to other two for large datasets [162].

Among the double transformation methods, SJMR method partitions the dataset and the elimination of duplicates takes place in the map phase before the join is performed by the reduce task. A single MapReduce task carry out the spatial join. While other methods such as parallel-PBSM uses two MapReduce tasks for the spatial join. First map task performs data partitioning and the reduce task computes the join. Second stage of map-reduce task eliminates duplication. Due to this difference SJMR performs better than parallel-PBSM. Performance of both increases with increase in reducing the task number, but beyond that performance of both deteriorates as the reduce task is not able to complete in one cycle [39].

Chapter 3

The Proposed QUIPSHoT Grid-GIS Architecture and Framework

In research works carried out so far, in the domain of integrated MapReduce and Grid computing, primarily non-spatial data have been used. However, a slightly different hybrid architecture, integrating MapReduce and Grid for spatial datasets, is presented here. To further improve the organization of spatial data and spatial query execution, a spatial indexing in the integrated architecture is implemented. A novel Integrated Grid and Spatially Indexed MapReduce QUiPSHoT Grid-GIS architecture and framework is proposed in this thesis. The architecture is an integration of a Grid Geospatial Database System (GGDS) [163] and the three parallel spatial indexes implemented in the MapReduce SpatialHadoop framework [23, 24, 25].

The inclusion of these three spatial indexes on the proposed Grid-GIS has given it the name QUiPSHoT. Where, 'QU' represents the H-bucket PMR Quadtree spatial index, 'i' represents the term index, 'P' represents the Parallel Priority R-Tree spatial index, 'SH' represents the integrated MapReduce part - SpatialHadoop in the proposed Grid-GIS, 'T' represents the Parallel Hilbert TGS R-Tree spatial index, and 'o' letter actually does not signify anything, it simply completes the name.

The architecture of the QUiPSHoT Grid-GIS accommodates OGSA based grid for spatial data that build a structure consisting of four layers: User layer, Geospatial grid extension layer, Grid protocol layer, and Geospatial resource layer. The four layers are implemented using the four components of the proposed architecture, as shown in Table 3.1.

Table 3.1: Mapping of OGSA based Grid for Spatial Grid for Spatial Data Layers to Components of the QUiPSHoT Grid-GIS Architecture

S.No.	**OGSA Based Grid for Spatial Data Layers**	**Components of the QUiPSHoT Grid-GIS Architecture**
1	User Layer	The component for interpreting the users queries in SQL

to be cont'd on next page

Table 3.1: MAPPING OF OGSA BASED GRID FOR SPATIAL DATA LAYERS TO COMPONENTS OF THE QUiPSHoT Grid-GIS ARCHITECTURE (Cont.)

S.No.	OGSA Based Grid for Spatial Data Layers	Components of the QUiPSHoT Grid-GIS Architecture
2	Geospatial grid extension layer	Catalog Management (CM) module and Query Decomposition and Parallelization (QDP) module
3	Grid Protocol Layer	Geospatial grid interface of Hadoop Enabled Spatial Grid (HESG) node
4	Geospatial Resource Layer	Geospatial data managed on Parallel H-bucket PMR Quadtree Index, Parallel Hilbert TGS R-Tree Index and Parallel Priority R-Tree Index enabled MapReduce

3.1 The Proposed Architecture of the QUIPSHoT Grid-GIS

The architecture is built on n number of Hadoop Enabled Spatial Grid (HESG) nodes. It is composed of a Spatial Grid Control (SGC) node and a number of HESG nodes, as shown in Figure 3.1. The SGC node is the main control node of the grid infrastructure that has two main functionalities. Firstly, the Query Decomposition and Parallelization (QDP) module accepts users queries interpreted in Structured Query Language (SQL) and decomposes into small decomposable forms. So that, each can be invoked in parallel on different HESG nodes through the Parallel Invocation interface. This interface provides a bidirectional communication with HESG nodes. It can insert spatial data in a geospatial database and extracts the required (queried) spatial data for presenting to a user. This decomposition is based on a distributed spatial data and complexity of the operation. Secondly, the Catalog management (CM) module in consultation with former module makes available the information (meta data) about the required spatial data, from the selected HESG nodes, through its Register interface.

The HESG nodes cooperate and communicate with both modules of the SGC node through the provided interfaces. The Register interface (R) and Invocation interface (I) of a HESG node interacts with the Parallel Invocation interface and the Register interface of the QDP module, and the CM module of the SGC node, respectively. The two modules of an HESG node interface with the Geospatial Grid interface that accesses

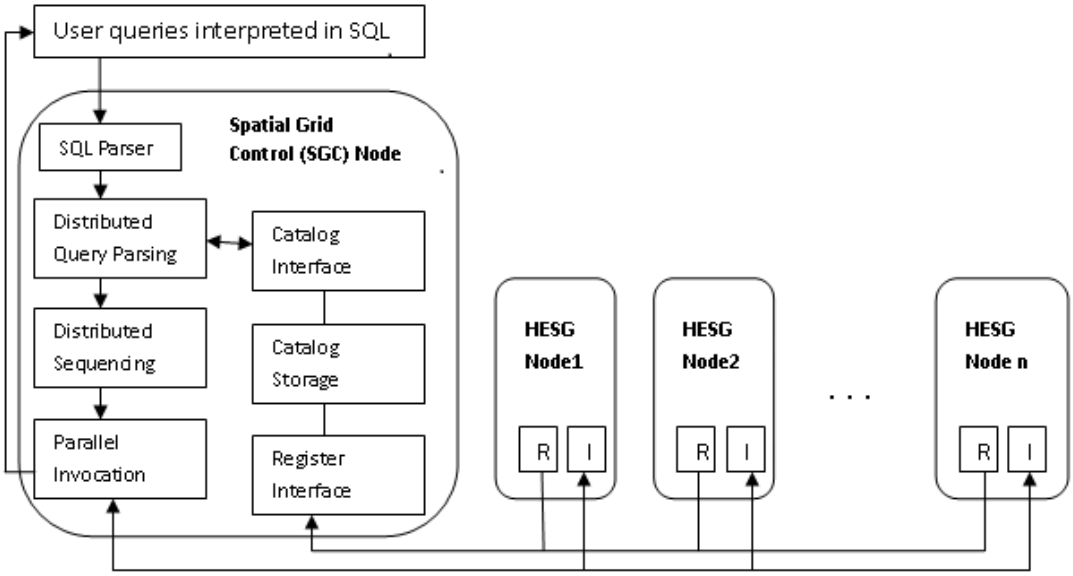

Figure 3.1: The Proposed Architecture of the QUiPSHoT Grid-GIS; R: Register Interface, I: Invocation Interface and the HESG: Hadoop Enabled Spatial Grid

the spatial data efficiently from the HDFS. The HDFS is constituted by a selected group of HESG nodes. The spatial data is Geospatial data accesses through Parallel H-bucket PMR Quadtree Index, Parallel Hilbert TGS R-Tree Index, and Parallel Priority R-Tree Index enabled MapReduce, as shown in Figure 3.2.

3.2 The Proposed Framework of the QUIPSHoT Grid-GIS

The QUiPSHoT Grid-GIS framework realizes the proposed architecture of QUiPSHoT Grid-GIS, presented in the Section 3.1, in practical. Similar to the architecture, the framework is also composed of the QUiPSHoT framework integrated in the Grid-GIS.

The QUiPSHoT framework is realized on the SpatialHadoop [23, 24, 25]. The SpatialHadoop comes equipped with the spatial index, grid file, R-Tree and R+-Tree. The grid index in the SpatialHadoop uniformly partitions input spatial dataset into ($\sqrt{n}$ x $\sqrt{n}$) equal parts, where n is the total number of grid nodes in which the input spatial dataset is partitioned. However, a sampling based data partitioning is used for the R-Tree and the R+-Tree spatial index. For each index, a partitioned dataset is processed by cluster nodes. The SpatialHadoop framework extends its spatial constructs in language, storage, and operations layers in Hadoop. The Language layer extends Pig Latin, a high level

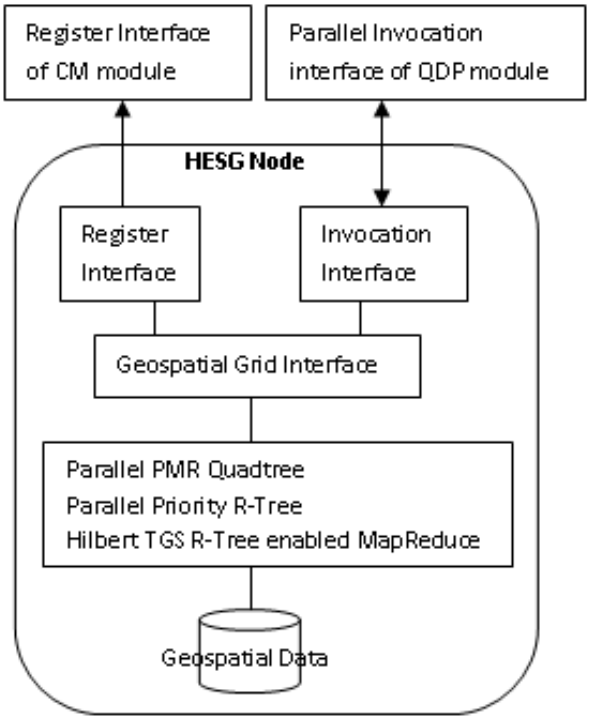

Figure 3.2: The Architecture of the HESG Node: A Component of the proposed QUiPSHoT Grid-GIS Architecture

language for Hadoop, by adding new spatial constructs. The Storage layer provides a two-layered spatial index structure where the global index partitions data across nodes while the local index organizes data in each node. The authors implemented grid index, R-tree and R+-tree as local indexes. The MapReduce layer used SpatialFileSplitter to exploit the global index by pruning partitions that do not contribute to a query answer, while the SpatialRecordReader exploits the local index to efficiently access records within each partition. The operation layer performs spatial operations, such as range query, k-Nearest Neighbors (kNN) and spatial join, implemented using the indexes and new components in the MapReduce layer. The authors found that the grid index takes minimum bulk-loading time as compared to R-Tree and R+Tree index. The R-Tree builds the index in less time as compared to R+-Tree index. However, the spatial range query and kNN query gets better performance on R-Tree, and the grid index shows the lowest performance. But for spatial join queries, the R+-Tree index shows the best performance among the three and the grid index shows the worst performance. The performance of the grid index drops considerably for non-uniformly distributed data.

The proposed three spatial indexes - the parallel H-bucket PMR Quadtree, the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree extends the spatial index dataset of the SpatialHadoop. The framework of the QUiPSHoT is shown in Figure 3.3. The

complete design and implementation is presented in Chapter 4.

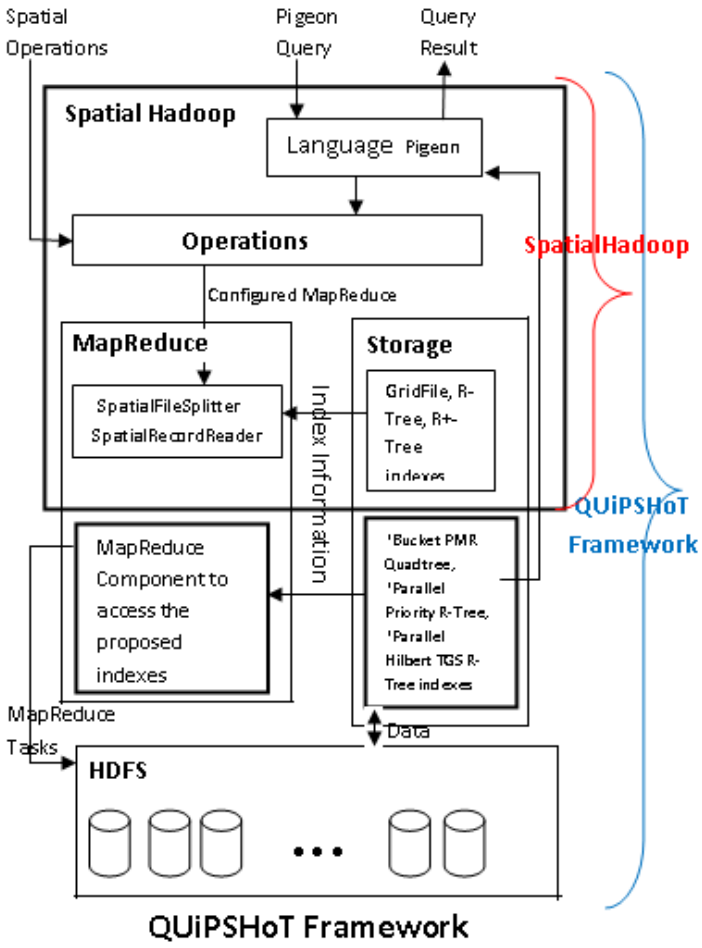

Figure 3.3: The QUiPSHoT Framework: A Component of the proposed QUiPSHoT Grid-GIS Framework

The QUiPSHoT Grid-GIS framework, as shown in the Figure 3.4, is realized by integrating the MapReduce based QUiPSHoT framework and the Grid-GIS. The user's interaction in the proposed framework is accomplished through the component for interpreting the users queries in SQL. The component is implemented using components defined in Java, such as Swing and Awt, for building Java Graphic Interface. The CM and QDP modules of the SGC node, as described in the proposed architecture in the Section 3.1, are implemented on the basis of the Grid Geospatial extension layer of the GGDS prototype GEOBARN [163]. The QDP module uses a Spatial Query Language that is compatible with SQL standards using the added geospatial operations and geospatial functions to manipulate geospatial objects [164]. The CM module is implemented with the Light Weight Directory Access (LDAP) protocol used to set-up a global catalog in the distributed system [165]. The catalogue storage, interface and update parts of

the QDP module are based on [163]. The Geospatial grid interface of a HESG node is implemented with methods and operations defined in Globus Toolkit 4.0.

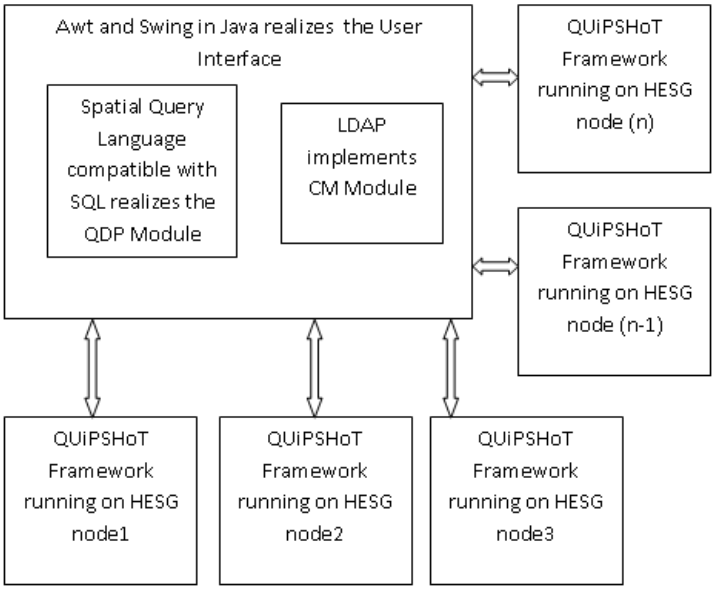

Figure 3.4: The Proposed QUiPSHoT Grid-GIS Framework

3.3 Spatial Query Processing in the Proposed QUIPSHoT Grid-GIS Framework

The spatial query processing in the QUiPSHoT architecture is described in Figure 3.5. The user can input a spatial query through any HESG node. The spatial query is interpreted by an SGC node with the help of its QDP and CM modules. The CM module identifies the potential HESG nodes for handling the spatial query. The potential HESG nodes contain the distributed spatial data that are required for solving the spatial query. One HESG node acts as the Namenode of the Hadoop Cluster, constituted from the identified potential HESG nodes, and form an HDFS for handling the input spatial query. The rest of the HESG nodes, that do not become part of the formed HDFS, provide their storage and compute resources to the Namenode.

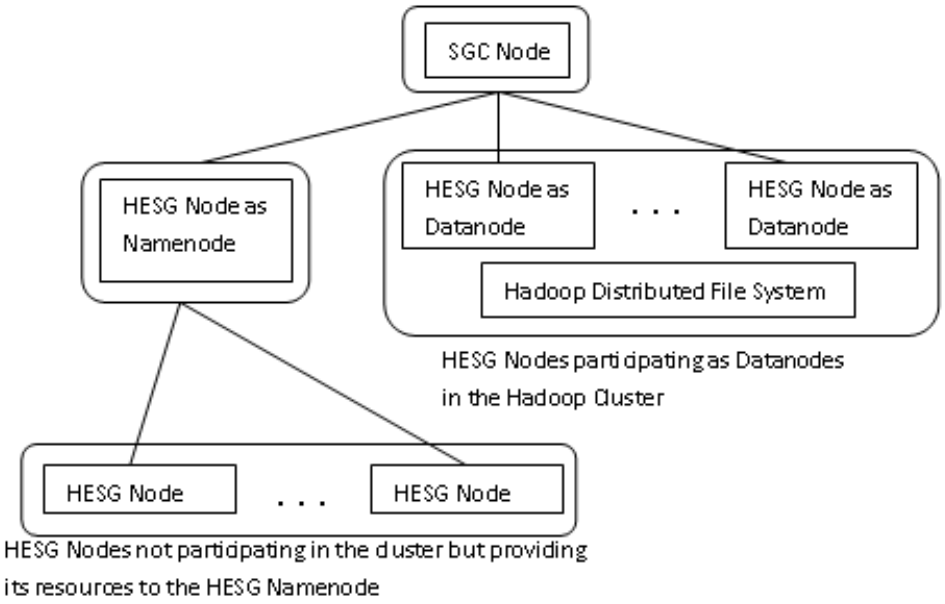

Figure 3.5: Spatial Query Processing in the Proposed QUiPSHoT Grid-GIS Framework

3.4 Performance Measuring Parameters of the Proposed QUiPSHoT Grid-GIS Framework

The efficiency of the proposed QUiPSHoT Grid-GIS is based on the following four parameters:

(i) **Bulk-loading cost of the index:** The cost of bulk-loading an index is a measure of the amount of disk space utilized for storing the indexed spatial dataset in secondary memory. A good indexing algorithm manages data quite efficiently and requires a minimum storage space. Another interpretation of it is the requirement of a minimum number of disk accesses for data insertion into the index for answering a query. The amount of time taken to insert all elements or bulk-loading the whole data is also a relevant term for measuring cost of bulk-loading the index, and consequently, the efficiency of the QUiPSHoT Grid-GIS.

(ii) **Execution Efficiency:** The execution efficiency is proposed for measuring two aspects of the proposed QUiPSHoT Grid-GIS. First, the efficiency gain achieved in building the index with respect to varying node capacity, and second, the efficiency gain achieved for spatial queries with respect to node capacity. The node capacity is the amount of disk space used for inserting a data item in internal nodes or leaf nodes of a spatial index.

(iii) **Scalability:** The scalability is an important feature of any distributed computing

system. A linear rise or decline in throughput with increasing or decreasing load or computing resources is always expected. If the distributed system does not follow this kind of variation, then it shows a poor response towards scalability. In this thesis work, a variation in the computing resources has been taken to test scalability. The scalability testing for the data is carried out by taking different types of data, such as real life data, synthetic data. There is further variation in the representation of the data under each category, a detailed explanation is presented in the Section 4.3.

(iv) **Availability:** The availability of the resources is an important characteristic of a distributed system. Generally, any distributed system is prone to failures due to failures occurring in different part of the system. A good distributed system is expected to tolerate such failures and recover from these. As a measure of availability feature of the proposed QUiPSHoT Grid-GIS, the system is tested by creating synthetic failures by deliberately shutting down few computing nodes at the run time. The test was performed on two categories of computing nodes, first, a master cluster node, and second, datanodes of the cluster.

Chapter 4

Design and Implementation of QUiPSHoT Grid-GIS

The chapter describes the design and implementation of parallel spatial indexes in the QUiPSHoT Grid-GIS, experimental set-up, dataset used, and parallel spatial queries used.

4.1 Design and Implementation of Parallel Spatial Indexes in the QUIPSHoT Grid-GIS

The integration of Grid and MapReduce provides a better architecture for dealing with data intensive and compute intensive spatial data. The two technologies complement each other in the integrated architecture. The former is better for dealing with bag-of-tasks with few I/Os, while the MapReduce computations are data intensive and uses a large number of inputs and intermediate data. A detailed description of the benefits of the integrated environment is presented in Chapter 2, Section 2.4. The MapReduce implementation-Hadoop uses key-value storage and better processes the sequential data. So, to better organize the voluminous spatial data and for the efficient spatial data retrieval, the parallel spatial indexes are used within the integrated Grid and MapReduce. The traditional spatial indexes developed for the sequential programming environment are surveyed in the Chapter 2, Section 2.2. The highly efficient traditional indexes, having high potential for parallelism, are considered for parallelization in the MapReduce. In this thesis work, three such traditional spatial indexing algorithms are considered: first is from the quadtree family, the bucket PMR Quadtree, the second is from the R-Tree variants, the Hilbert TGS R-Tree and third is again from the R-Tree variants, the Priority R-Tree. The parallel versions of these indexes are developed, namely the Parallel H-bucket PMR Quadtree, the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree, respectively, in the MapReduce environment under the QUiPSHoT Grid-GIS framework. The MapReduce implementation for spatial data-SpatialHadoop is used as the parallel programming model. The model programs the logic through the map and

reduce functions in a master-slave SpatialHadoop architecture, that handles the I/O in key-value pair. The model parallelizes the processing by partitioning the input dataset on the distributed file system of the cluster. Each computing node in the cluster processes the part of its task and returns the result to the master node.

4.1.1 Design and Implementation of the Parallel H-bucket PMR Quadtree

The detailed process of inserting line segments in the PMR (Polygonal Map Random) quadtree is described in [106]. The line segments are inserted into the PMR quadtree, starting with inserting the first line segment into an empty block. A block is split into four quadrants of equal sizes until the object count in the block reaches a threshold value. The splitting threshold is the permissible number of line segments in a block. A new line segment is inserted into a block, if it intersects the block, and the block is checked for the splitting threshold. A block is split into sub-blocks only once, as it contains a few very close lines. It is important to keep the particular value of the splitting threshold, as too small value causes many subdivisions that lead to many empty sub-blocks and hence the increased storage requirement. A large value of the splitting threshold causes a decreased construction time and storage requirement, but at the same time it increases the time for performing operations on it. The order of the inserted line segments decides the shape of the resulting PMR quadtree. However, in the parallel construction of the PMR quadtree, the insertion ordering of line segments is not known, as the lines are inserted simultaneously in parallel and therefore a slight modification to the PMR quadtree, known as the bucket PMR quadtree is used. The tree data structures generally access the data from primary memory and pointers are used to access the data stored in the pages of secondary memory, however, it gives rise to page faults. The bucketing method overcomes this problem by collecting the data objects into sets called buckets, and providing access to buckets through appropriate address computation mechanism. In the bucket PMR quadtree, the block or bucket is split repeatedly until a splitting threshold is reached, unlike in the PMR quadtree, which allows splitting a block only once. The splitting threshold of a bucket is also represented as the node capacity of the tree node. The node capacity of a data structure tree node is the data holding capacity of the node in terms of size of the data in the tree. It decides the number of objects in a bounding rectangle. The objects are grouped according to their proximity for better results.

The regular disjoint decomposition methods based on quadtree have more potential for inter-processor communication or in other words for parallelism. However, not much work

is available that uses the quadtree-based spatial data structure of MapReduce model. One such approach uses the quadtree-based global index and Hilbert-curve based local index [36], where the global index searches the data blocks and the Hilbert index locates the spatial objects for efficient data retrieval. The use of quadtree based indexes for building a local index on the MapReduce is used in [22]. The authors proposed a HQ-Tree index on the Hadoop that considers the PR-Quadtree index for spatial point objects. It solves the issues of order of data insertion and space overlap in non-disjoint decomposition approaches. The authors found an improved execution time performance for the index creation, the point query and the range query, for parameter data size, node size and number of query target points. However, the HQ-Tree approach is limited to spatial point objects.

Among many quadtree based data structures for indexing and querying, the PMR Quadtree was found to be the best for representing the curvilinear data [104]. In the domain of traditional serial programming models, the PMR Quadtree index based data structure was proven better for spatial join queries [156] and line segment queries [154] over the R-Tree [4], the R*-tree[3] and the R+-tree[5]. However, the motive of all the concerned research is on reducing the build time and the query execution time. The efforts were done in the past to implement the existing serial spatial indexes on parallel platforms. In one such work, the hypercube architecture, the Scan-and-Monotonic mapping (SAM) model of parallel computation is used [150] for spatial data structure indexing and querying [151, 152, 155]. The hypercube architecture is a tightly coupled architecture that is characterized by the presence of 2^n processors interconnected as n-dimensional binary cube [166]. A scan operation [167] comprising of primitive operations [149] element-wise, permutation and scan, operates on long vectors of data. In one such comparative study on the SAM model, the PMR Quadtree was found to be slightly faster than the R-tree and much better than the R+-tree, for similar tree node capacities (the capacity to hold the number of spatial objects by a node of the tree) for spatial join queries [151, 155] and polygonization queries [152].

However, the MapReduce based parallel programming, Hadoop [95], provides a loosely coupled architecture. The computing nodes can be added on the fly to provide the scalability, which is not possible in the SAM model. The programming by using the Map and Reduce functions is quite easy in the MapReduce, as compared to the scan operations that operates on long vectors of data. Other important things that favor the use of MapReduce model is its simplicity, ease of use and the ability to easily handle large data sets. Besides all these factors, the fault tolerant nature of the MapReduce model and the property of abstracting the parallelization from the user are the reasons

that have motivated researchers to use the model. It provides good performance through scaling out to computing clusters. The ability of the existing bucket PMR quadtree index to efficiently handle the curvilinear data for indexing and querying, and the advantages of the Hadoop parallel processing framework, motivated to carry out this work and to develop H-bucket PMR Quadtree index in QUiPSHoT Grid-GIS, where the letter SH stands for the SpatialHadoop.

The conceptual idea of creating a bucket-PMR Quadtree index in MapReduce is presented in the Figure 4.1. The Figure 4.1(a) shows the input polygonal map dataset of 11 line objects. The dataset is partitioned as per the block size over the Hadoop Distributed File System (HDFS), and the required number of mappers use Algorithm 4.1 to split line objects into different quadrants. The line objects from same quadrant are output to a reducer function, as shown in the Figure 4.1(b) that starts the index building process as per the Algorithm 4.2. A bucket size of three is assumed here. It means only three line objects can be accommodated in the proposed tree node. Each Reduce node checks the number of line objects it has received from the mapper. If the count is less than or equal to three then all the line objects are assigned to the node and the node gets the status of a leaf node (L). Otherwise, the node is vacated with status changed to internal node (IN) and all the entries of the node are given to the four newly created child nodes as per the MBR intersect condition of Algorithm 4.2. This process is repeated until all the nodes are filled-up with at most three line objects or there are no more objects for insertion. The process on completion creates an H-bucket PMR Quadtree as shown in the Figure 4.1(c), and its logical counterpart is shown in the Figure 4.1(d).

This section presents the H-bucket PMR quadtree on the MapReduce. Firstly, the data partitioning into four quadrants in the MapReduce for parallel construction of the proposed index, is carried out. Secondly, the tree node structure of the proposed index and the bulk-loading the proposed index is discussed.

(i) **The Proposed Parallel H-bucket PMR Quadtree Index in the MapReduce:** The split of spatial dataset depends on the size of the data block in the HDFS. If the block size is 'b' and the spatial dataset is of size 's' then n=s/b splits are created and each split is taken care of by one mapper. Each Mapper runs Algorithm 4.1 and segregates line objects in one of the quadrant. For each row of spatial data in the input csv file, each map function checks in parallel both ends of the line coordinates for insertion in four quadrant. For deciding the space of four quadrants, two variables xmid and ymid are computed on the basis of the minimum and maximum values for x and y coordinates from the whole spatial data. For each line having start and end coordinates ((lx-min, ly-min),(lx-max, ly-max)),

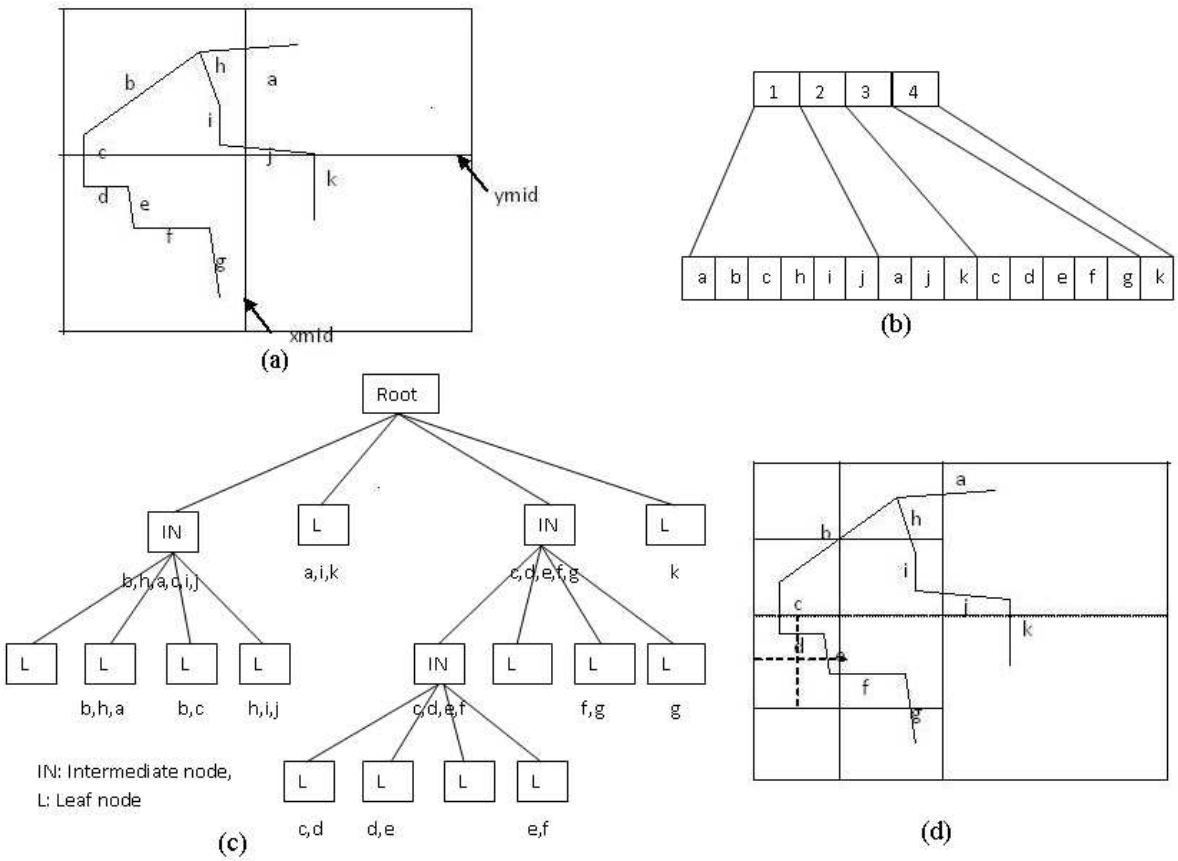

Figure 4.1: The Conceptual View of the H-bucket PMR Quadtree in MapReduce (a) Input spatial line data to MapReduce (b) Map function in MapReduce assigns the decomposed input spatial line objects into four quadrants (c) Data in the quadrants recursively filled-up in bucket-PMR Quadtree nodes for a bucket size=3 (d) Logical structure of the MapReduce constructed H-bucket PMR Quadtree for bucket size=3.

Each mapper function puts the line in one of the four quadrants through finding out lx-min and lx-max against the x-mid, and ly-min and ly-max against the y-mid. Two cases arise when lines are compared against xmid and ymid valuse: (a) lines do not intersect xmid and ymid coordinates and completely falls in one of the four quadrants. (b) lines intersect xmid and ymid coordinates and are put in both the quadrants. The output of Map function for a particular line is (id of the quadrant, line (line-id and MBR)). All lines common to a quadrant become input to the reducer with (id of the quadrant, list(line)). Now, the H-bucket PMR Quadtree index is created for all the lines in a particular quadrant. The generated H-bucket PMR Quadtree index from each reducer is merged outside the reduce function in a serial way to get the complete H-bucket PMR Quadtree index.

(ii) **Data Structure of the H-bucket PMR Quadtree Index Node:** The node structure of the H-bucket PMR Quadree index is represented with the class Buck-

Algorithm 4.1 Parallel H-bucket PMR quadtree index in MapReduce

INPUT: Set of spatial lines
OUTPUT: The H-bucket PMR Quadtree index
Step1:
for each line object ((lx-min, ly-min),(lx-max, ly-max)) **do**
  if (((lx-min && lx-max)<x-mid) && ((ly-min && ly-max)> y-mid)) **then**
    put the line in 1st quadrant
  end if
  if (((lx-min && lx-max)> x-mid) && ((ly-min && ly-max) < y-mid)) **then**
    put the line in 2rd quadrant
  end if
  if (((lx-min && lx-max) < x-mid) && ((ly-min && ly-max) < y-mid)) **then**
    put the line in 3rd quadrant
  end if
  if (((lx-min && lx-max) > x-mid) && ((ly-min && ly-max)> y-mid)) **then**
    put the line in 4th quadrant
  end if
  if ((lx-min < x-mid) && (x-mid < lx-max)) **then**
    include line in both the quadrants
  end if
  if ((ly-min < y-mid) && (y-mid < ly-max)) **then**
    include line in both the quadrants
  end if
  emit (id of the quadrant, line (line-id and MBR))
end for
Step2: The H-bucket PMR Quadtree index is created by the reducer function with input (id of the quadrant, list (line)) for each quadrant according to the Algorithm 4.2

Step3: Set the path for input and output directories, and then start up MapReduce
Step4: A merge process in Hadoop combines all child indexes

etPMRQTreeNode that contains six elements: level - the present level of the BucketPMRQuadTree node, the level describes the stage of the recursive decomposition; maxLevel - the maximum permissible level defined for the BucketPMRQuadTreeNode; BucketCapacity - the maximum number of lines objects that can be kept in a the tree node; Minimum Bounding Rectangle (MBR) the area of the rectangular quadrant in which lines lie in the form of ((xmin,ymin),(xmax,ymax)), childNodes an array of type BucketPMRQTreeNode; and the SpatialData Collection contains the line objects implemented with Collection structure in Java that can store line objects but not more than the BucketCapacity. A class Box is presented separately just to simply the presentation of the tree node structure. It specifies the rectangular region enclosed by the MBR.

```
Class BucketPMRQTreeNode{
Int level; // the level of this node
Int maxLevel; // the maximum number of levels of the quadtree;
Int BucketCapacity;
Box MBR;
BucketPMRQTreeNode childNodes[];
SpatialDataCollection data;}

Class Box{
Double xmin,ymin;
Double xmax, ymax;}

Public class SpatialDataCollection {
ArrayList<BucketPMRQTreeNode> items= new ArrayList< BucketPMRQTreeNode>();}
```

(iii) **The Algorithm for Data Insertion in H-bucket PMR Quadtree Index:** Input to the reducer as described in the Algorithm 4.1, in the step (id of the quadrant, list(line, and the MBR to which it belongs)), creates a local H-bucket PMR Quadtree index for all the line objects falling in this quadrant as per the Algorithm 4.2.

Algorithm 4.2 Bulk-loading H-bucket PMR Quadtree Index

INPUT: Spatial data
OUTPUT: A local H-bucket PMR Quadtree
Step1: Get the spatial data to be inserted in the H-bucket PMR Quadtree
Step2: Check the spatial data against the quadrant MBR and perform step3 if it intersects the MBR
Step3: Check the BucketPMRQTreeNode for a leaf/non-leaf node of the H-bucket PMR Quadtree index
Step4:
if (Isleaf=true) **then**
  perform Step5
else
  goto Step8
end if
Step5: Add this spatial data item to the ArrayList Collection
Step6: After Inserting the spatial data in the ArrayList Collection, check for maximum level reached or the number of data items exceeds the bucket capacity. If either one holds true then split the node into four child nodes and increase the level by 1 and vacate the ArrayList Collection into childnodes ArrayList Collection as per Step7
Step7: Repeat Step2 to Step5. Clear parent nodes ArrayList Collection and set Isleaf=false
Step8: Repeat Step2 to Step7 for each childnodes in the H-bucket PMR Quadtree

4.1.2 Design and Implementation of the Parallel Hilbert TGS R-Tree

The distributed geospatial query processing was implemented to minimize the data transmission and to improve the efficiency of execution through parallel distributed computing. The use of MapReduce technology ensured the reduction in data transmission and the Hilbert TGS R-Tree index on the spatial data improved the efficiency of spatial query execution. Apart from the data transmission and the improved efficiency of query execution, the MapReduce provided a fault tolerant file system, through replica management on the HDFS created, with the HESG nodes. The parallel distributed computing of the traditional Hilbert TGS R-Tree [168] was considered in the MapReduce environment after a thorough research on different potential spatial indexing algorithms. The limitation of the basic R-Tree [18] is that it adds one element at a time to the tree. However, when the elements to be added are large in number and known in advance, it is better to use the known information to quickly build the R-Tree using heuristic that keeps neighboring objects together on the basis of their MBRs, and inserting the objects in leaf nodes in parallel using a bottom-up approach, packed R-tree [169]. Though, the packed R-tree [169] quickly builds an R-tree, but the heuristic uses the center coordinate of the MBRs

along one particular dimension that leads to objects closer in a particular dimension, but actually the objects do not lie close in space (in all dimensions). The problem increases further for higher dimensions. The spatial queries on an R-Tree works in a top-down manner, the performance of the queries would improve if the R-Tree is built in a top-down manner, TGS R-Tree [140]. The authors claimed a better control on the MBR properties, overlap and dead space. The authors also claimed a better proximity of objects in space by considering ordering of objects in all dimensions and selecting the best pair with respect to some function (area for two dimensions and volume for higher dimensions). The drawback of the TGS R-Tree algorithm was the slow construction of the R-tree as compared to the basic R-Tree, as much time is spent in choosing the objects according to the best space proximity; however, the TGS R-Tree achieved the best query time [146]. The Hilbert curve provided a solution to the problem as the ordering of the objects on the curve remains quite similar in all the dimensions and the neighboring objects on Hilbert curve also remains close in the space [33]. Now, the Hilbert coordinates on the curve is used to provide a mapping from d-dimensional coordinates to one-dimensional coordinates.

The Parallel Hilbert TGS R-Tree is achieved using Algorithm 4.3, Algorithm 4.4, Algorithm 4.5 and Algorithm 4.6.

(i) **HilbertTGSBulkLoad algorithm:**

HilbertTGSBulkLoad algorithm, takes a list of n rectangles D=r1, r2, , rn as input and returns a valid Hilbert TGS R-Tree. Each rectangle in the input dataset is sorted on the Hilbert coordinates calculated in the MapReduce, step 1 of the Algorithm 4.3, according to the algorithm described in [33]. The output of the mapper is a function f that assigns any object of D into one of the t possible partitions of equal size. The function f takes as input a Hilbert coordinate and outputs a partition number, i = 1 to t. The function f also preserves the spatial locality by using the Hilbert curve values. Note that no actual partitioning or data moving happens at this point. The next chained mapper phase utilizes f for such purpose. Now, each mapper function calculates the number of rectangles |Di| passed to it and the height of the resulting Hilbert TGS R-Tree for each partition. The output of the mapper function passes each dataset Di and height hi to the reducers, for further building the independent Hilbert TGS R-Tree. Two more parameters N and M are provided as input, where N is the capacity of leaf nodes, and M is the capacity of non-leaf nodes. The function f and the Hilbert values of each object generated by the mapper function of the MapReduce put similar objects from input dataset Di into same partition. The n Reducers, now take these partitioned dataset as input and build n

Hilbert TGS R-Trees according to the Hilbert TGS R-Tree construction algorithm. The 'n' individual Hilbert TGS R-Trees are combined under a single root node to form the final Hilbert TGS R-Tree. This phase is executed outside the cluster as it does not require any computational burden.

Algorithm 4.3 HilbertTGSBulkLoad(D,N,M):

INPUT: a list of n rectangles D = (r1, r2, , rn)
OUTPUT: A Hilbert TGS R-Tree root
Step1: Sort each rectangle in the input data-set D with the Hilbert co-ordinate of the center co-ordinate of each rectangle
Step2: Partition the sorted dataset into t equal size partitions
Step3: Calculate, the height of the resulting R-Tree, hi = max(0, [logM ni/N]), where ni=|Di|, i varies from 1..t
Step4: return HilbertTGSBulkLoadChunk(Di, hi)

(ii) **HilbertTGSBulkLoadChunk(Di, h):**

The Reducer function accepts input from the output of the i^{th} mapper function that also represents the ith partitioned dataset, and returns a root node with an independent Hilbert TGS R-Tree for the i^{th} partitioned dataset. The HilbertTGSPartition takes two values as input (Di,m) and returns a single value that points to the HilbertTGS partition. The logic is presented in Algorithm 4.4. The algorithm builds the tree of height h for ni number of rectangles.

Algorithm 4.4 HilbertTGSBulkLoadChunk(Di, h):

INPUT: The Mappers output from Algorithm 4.3 with input data-set Di and R-Tree height hi becomes input of the Reducer
OUTPUT: A root node with an independent Hilbert TGS R-Tree for the ith partitioned dataset **Step1:**
if (hi==0) **then**
  return A Hilbert TGS R-Tree leaf node
else
  Set m = $N * M^{hi-1}$
end if
Step2: Di=(Di1,Di2,..,Dik) = HilbertTGSPartition(Di, m), where k<=M
Step3:
for (i= 1 to k) **do**
  ni = HilbertTGSBulkLoadChunk(Di, hi-1)
end for
Step4: return an internal non-leaf node (ni1, ni2, , nik)

(iii) **HilbertTGSPartition(Di, m):**

The input to the Algorithm 4.5 is the dataset Di and m, the maximum number of elements that a Hilbert TGS R-tree of height h can store.

Algorithm 4.5 HilbertTGSPartition(Di, m):

INPUT: ith partitioned dataset Di and m values passed from Algorithm 4.4
OUTPUT: A partitioned data-set Di into k<=M subsets
Step1:
if ni <= m **then**
  return Di // ni=|Di|, Single partition
else
  (L, H) = HilbertTGSBestBinarySplit(Di, m)
end if
Step2: return Concatenation of HilbertTGSPartition(L,m)and HilbertTGSPartition(H,m)

(iv) **HilbertTGSBestBinarySplit(Di, m):**

The Algorithm 4.6 splits the dataset, such that, a better binary split is obtained that has a better control on the MBR properties, overlaps and dead space. It results in a better proximity of objects in space by considering ordering of objects in all dimensions and selecting the best pair with respect to some function. The algorithm takes i^{th} dataset and m, the maximum number of elements that a Hilbert TGS R-tree of height hi can store, as input and produces the best binary splits in two variables L and R, on the basis of minimizing a pre-determined cost function.

4.1.3 Design and Implementation of the Parallel Priority R-Tree

The original R-Tree [4] is an index on spatial approximations of spatial data, where the spatial approximations are represented with MBRs. The approximations are required, otherwise it would become very difficult to deal with the complex geometries of spatial objects. The dynamic R-Tree index was a good breakthrough for representing a huge volume of spatial datasets. However, bulk-loading algorithms were used for constructing R-Tree, where a dataset was known in advance. To further improve the efficiency of a statically built R-Tree for storage space, efficiency of index building and efficiency of query execution, many heuristics were used. These heuristics resulted in many R-Tree variants, such as R*-Tree, R+-Tree, Hilbert R-Tree, Packed R-Tree, TGS-R-Tree, Hilbert TGS R-Tree, etc. A detailed explanation and comparison are provided in the Chapter 2, Section 2.2. The Hilbert TGS R-Tree, discussed in the previous section, uses a top-down approach

Algorithm 4.6 HilbertTGSBestBinarySplit(Di, m):

INPUT: D and m values passed from Algorithm 4.5
OUTPUT: The best binary split is obtained, in variables L and H, from the input data-set Di, with the Hilbert co-ordinate of the center co-ordinate of each rectangle, after applying cost function
Step1: Set p = [n/m] // where n=|Di|
Step2: c* = infinity // (Initially, the best cost)
Step3:L and H = empty sets, lists of M-1 rectangles, B is a list of M rectangles
Step4:
for (j = 1 to p-1) **do**
  Compute the bounding box Bij=D(j-1).m+1@ D(j-1).m+2 @ . @ Dmin(|D|, j.m) // where @ denotes an operator that finds the minimum bounding box of two operand rectangles
  Step5:
  for (k=j+1 to p) **do**
    Compute the bounding box Bik=D(k-1).m+1@ D(k-1).m+2 @ . @ Dmin(|D|, k.m)
  end for
  Step6: c = cost(Bij, Bik)
  Step7:
  if (c < c*) **then**
    c* = c
    L = Bij
    Set H = Bik
  end if
end for
Step8: return (L, H)

to achieve a very good query execution time. The H-bucket PMR Quadtree, discussed in the section, also claims to achieve a very good execution time. All these algorithms claimed good average performances in case of different type of spatial datasets, such as uniformly distributed datasets, skewed datasets, datasets with large objects and hence approximated with large rectangles. Usually, in quad-trees the worst case performance is around $O(\sqrt{n})$ but in average case it performs better. However, none of the algorithms claimed a minimum worst case performance. The Priority R-Tree algorithm [17] claimed a guaranteed worst case performance. The algorithm builds an R-Tree by constructing pseudo PR-Tree for a dataset and then leaves of the pseudo PR-Tree is used to finally build a PR-Tree. The pseudo PR-Tree builds four leaf nodes. Each such leaf node bounds B rectangles (number of rectangles that fit in a disk block) of a dataset on the basis of minimum x, minimum y, maximum x, and maximum y coordinates. The rest of the data rectangles are put in two intermediate nodes, which are again explored to build pseudo PR-Tree in a recursive manner. When all the leaf nodes are achieved then the process for building a PR-Tree is started. The leaf nodes so obtained from the pseudo PR-Tree are put in the bounding rectangles. The pseudo PR-Tree building procedure is applied on the bounding rectangles. The process is repeated until the pseudo PR-Tree has all leaves at level 1 (adjacent to the root) and the final PR-Tree is hence produced.

The parallel distributed computing of the traditional Priority R-Tree [17] is considered here to harness the worst-case performance of spatial window queries for non-uniform datasets. A short form of the Parallel Priority R-Tree i.e. Parallel PR-Tree has been used at many places.

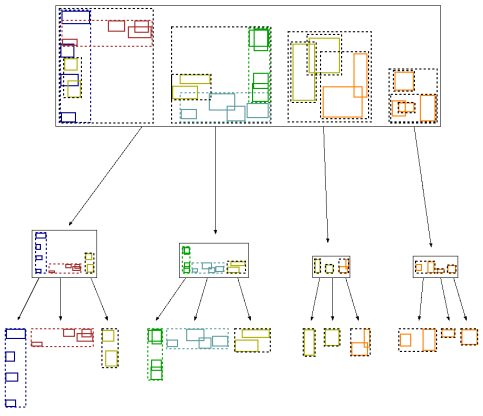

Figure 4.2: The final Parallel PR-Tree constructed [41]

A Priority R-Tree, also well known as the PR-Tree, uses a specialized bulk-loading algorithm for constructing an R-Tree that guarantees a minimum worst-case query performance. It is constructed in two steps: (a) constructing a Pseudo PR-Tree (b) Constructing the actual PR-Tree from the Pseudo PR-Tree. A pseudo PR-Tree is different from an

R-Tree, as it does not have leaves at the same level. However, a pseudo PR-Tree answers window queries efficiently. The PR-Tree is obtained from the pseudo PR-Tree. The steps for constructing a pseudo PR-Tree in a serial programming environment is described in [41], and a constructed PR-Tree is shown in Figure 4.2.

For a spatial dataset, the B number of MBRs (number of rectangles that fit in a disk block) with their minimum x-coordinates, minimum y-coordinates, maximum x-coordinates and maximum y-coordinates under each bounding box represent priority leaves of the pseudo PR-Tree. The remaining data object MBRs partition into parts each containing an equal number of MBRs. Suppose, Ts< represents the one half of the remaining rectangle whose minimum x-coordinate is less than a fixed x-coordinate. Similarly, Ts> represents the other half whose x-coordinate is more than a fixed x-coordinate. However, to reach at the final pseudo PR-Tree, the pseudo PR-Trees are constructed for the Ts< and Ts> nodes of the constructed pseudo PR-Tree, recursively until all the leaf nodes are obtained. The leaves of the finally built pseudo PR-Tree are used to build the PR-Tree through a bottom-up approach. The process starts with finding the minimum bounding rectangles for each leaf. The PR-Tree construction process is repeated until the pseudo PR-Tree has all leaves at level 1 (adjacent to the root) and the final PR-Tree is produced.

This section presents the algorithm for constructing the parallel Priority R-Tree on the MapReduce. Firstly, the pseudo PR-Tree is constructed from the given spatial dataset and then the final Priority R-Tree is constructed from the pseudo PR-Tree. Thirdly, the design of two spatial queries, line search query and the range query, on the proposed index is described.

(i) **The Proposed Parallel PR-Tree Index in the MapReduce:**

The split of spatial datasets depends on the size of the data block in the HDFS. If block size is 'b' and the spatial dataset is of size 's' then n=s/b splits are created and each split is taken care of by one mapper. Each Mapper in Algorithm-1 separates the first B rectangles (the number of rectangles that fits in a disk block) in four different categories based on the minimum and maximum coordinate along x-axis and y-axis, and puts the rest of the rectangles into a fifth category. All the rectangles from the five categories are passed to the reducer functions. Each reducer sorts rectangles based on the strategy used for separating these rectangles and puts the first B rectangles into a bounding rectangle, and puts the rest of the rectangle into fifth category. The bounding rectangles become the leaf nodes of the pseudo PR-Tree. In this whole process, the reducer eliminates the duplicate rectangles generated, if any. The reducer function takes rectangles from the fifth category and divides rectangles into two parts based on either a defined Xmid or Ymid coordinates and

puts these into two bounding rectangles, named Ts> and Ts<. Now each bounding rectangle is forced to repeat all the above mentioned steps through a recursive call until all the leaf nodes are obtained.

When a pseudo PR-Tree is obtained that contains all the leaf nodes, then all the leaves of the pseudo PR-Tree are fed to another mapper function through a chained MapReduce process. Each mapper function separates the data rectangles under four different bounding rectangles. Each bounding rectangle recursively calls the Algorithm-1 for generating pseudo PR-Tree. The process is repeated until all the leaf nodes are obtained at the same level. A merge process combines all the child indexes to obtain the final PR-Tree.

Step1: Initialize B (the number of rectangles that can fit in a disk block), Xmin-d, Ymin-d, Xmax-d and Ymax-d.
Step2: Each mapper function identifies the rectangles from the pseudo PR-Tree and separates into four bounding rectangles based on Xmin-d, Ymin-d, Xmax-d and Ymax-d coordinates, and passes these to four reducer functions.
Step3: Each reducer function checks, if the number of rectangles in any of the bounding rectangle is more than B then call Algorithm 1 for constructing the pseudo PR-Tree for that bounding rectangle.
Step4: Repeat Steps 1-3 until all the leaf nodes are obtained at the same level of the tree.
Step5. A merge process in Hadoop combines all child indexes and produces a parallel PR-Tree.

(ii) **Data Structure of the Parallel PR-Tree Index Node:** The node structure for the Parallel PR-Tree index is represented with the class ParallelPR-TreeNode that contains eight elements: level - the present level of the ParallelPR-TreeNode node, the level describes the stage of the recursive decomposition; Minimum Bounding Rectangle (MBR) the area of the present node with coordinates ((Xmin,Ymin), (Xmax,Ymax)), the four Priority leaves of the Tree are represented with LeafNodeXmin-d, LeafNodeYmin-d, LeafNodeXmax-d and LeafNodeYmax-d. The Xmin-d represents the smallest Xmin coordinate of any data rectangle. The Ymin-d represents the smallest Ymin coordinate of any data rectangle. The Xmax-d represents the largest Xmin coordinate of any data rectangle. The Ymax-d represents the largest Ymax coordinate of any data rectangle. The two intermediate nodes that contain the rest of the data rectangle, apart from the data rectangles contained in the four leaf nodes, are represented with Ts< and Ts>. Ts< contains the data rectangles

Algorithm 4.7 Parallel Pseudo Priority R-Tree index in MapReduce

INPUT: Spatial dataset consisting of spatial objects approximated by their MBRs
OUTPUT: A Pseudo PR-Tree
Step1: The dataset is distributed on the HDFS cluster and a bounding rectangle encloses all the data rectangles. Find B (the number of rectangles that can fit in a disk block)
Step2: Define Xmin-d, Ymin-d, Xmax-d and Ymax-d coordinates for the bounding rectangle
Step3: Mapper functions on datanodes are invoked. Each mapper function does the following:
Step3(a): The first B rectangles among all the rectangles in a datanode having Xmin coordinates close/near to Xmin-d are collected and passed to a reducer function. The other rectangles are put in Category X.
Step3(b): The first B rectangles among all the rectangles in a datanode having Ymin coordinates close/near to Ymin-d are collected and passed to a reducer function. The other rectangles are put in Category X.
Step3(c): The first B rectangles among all the rectangles in a datanode having Xmax coordinates close/near to Ymax-d are collected and passed to a reducer function. The other rectangles are put in Category X.
Step3(d): The first B rectangles among all the rectangles in a datanode having Ymax coordinates close/near to Ymax-d are collected and passed to a reducer function. The other rectangles are put in Category X.
Step4: All the duplicate rectangles are eliminated from the Category X.
Step5(a): A reducer function sorts the rectangles passed by each mapper function on the basis of Xmin-d coordinate and selects the first B rectangles. The rest of the rectangles are put in Category X.
Step5(b): A reducer function sorts the rectangles passed by each mapper function on the basis of Ymin-d coordinate and selects the first B rectangles. The rest of the rectangles are put in Category X.
Step5(c): A reducer function sorts the rectangles passed by each mapper function on the basis of Xmax-d coordinate and selects the first B rectangles. The rest of the rectangles are put in Category X.
Step5(d): A reducer function sorts the rectangles passed by each mapper function on the basis of Ymax-d coordinate and selects the first B rectangles. The rest of the rectangles are put in Category X.
Step6: All the rectangles put in the category X are partitioned in two equal halves and enclosed in bounding rectangles Ts< and Ts> on the basis of a fixed Xmid or Ymid coordinates, respectively.
Step7: For both the bounding rectangles, Ts< and Ts>, repeat steps Step2 to Step6 until all leaf nodes are obtained.

Algorithm 4.8 Parallel Priority R-Tree index in MapReduce

INPUT: All the leaves of the pseudo PR-Tree
OUTPUT: A PR-Tree
Step1: Initialize B (the number of rectangles that can fit in a disk block), Xmin-d, Ymin-d, Xmax-d and Ymax-d.
Step2: Each mapper function identifies the rectangles from the pseudo PR-Tree and separates into four bounding rectangles based on Xmin-d, Ymin-d, Xmax-d and Ymax-d coordinates, and passes these to four reducer functions.
Step3: Each reducer function checks, if the number of rectangles in any of the bounding rectangle is more than B then call Algorithm 1 for constructing the pseudo PR-Tree for that bounding rectangle.
Step4: Repeat Steps 1-3 until all the leaf nodes are obtained at the same level of the tree.
Step5: A merge process in Hadoop combines all child indexes and produces a parallel PR-Tree.

whose minimum x-coordinate is less than a fixed x-coordinate, defined from the bounding rectangle, such that it divides the data rectangles in intermediate nodes into two equal parts. Similarly, Ts> also contains the data rectangles whose minimum x-coordinate is more than the fixed x-coordinate. The data rectangles in Ts< and Ts> are collected using an arraylist SpatialDataCollection. A class BoundingRectangle is presented separately just to simply the presentation of the tree node structure. It specifies the rectangular region enclosed by the MBR.

```
Class ParallelPR-TreeNode {
Int level; // the level of this node
BoundingRectangle MBR;
BoundingRectangle LeafNodeXmin-d[0];
BoundingRectangle LeafNodeYmin-d[0];
BoundingRectangle LeafNodeXmax-d[0];
BoundingRectangle LeafNodeYmax-d[0];
SpatialDataCollection Ts>;
SpatialDataCollection Ts<; }


Class BoundingRectangle {
Double Xmin,Ymin;
Double Xmax, Ymax; }


Public class SpatialDataCollection {
ArrayList<ParallelPR-TreeNode> items= new ArrayList< ParallelPR-TreeNode>();
```

}

4.1.4 The Conceptual View of the R+-Tree Index in the MapReduce

The R+-tree is a variant of the R-tree that allows disjoint decomposition of the space by not allowing overlapping among intermediate nodes. It leads to a good improvement in the search time, but requires more storage as compared to the R-tree and R*-tree. The drawback of R+-tree is that the decomposition is data-dependent, which makes it difficult to perform some tasks that require composition of different operations and data sets. A recursive top-down approach is used to insert a line segment into an R+-tree that places it into every leaf node that it intersects [154]. If the leaf node in which the line segment is inserted overflows, then the leaf node is split. The split approach minimizes the total number resulting portions of line segments. To fulfill this requirement, all possible vertical and horizontal split lines are considered, and for each split line, the number of intersected line segments is counted. Finally, the split line having a minimum number of intersections is taken, and the splits are propagated up the tree. For the input spatial line dataset of the Figure 4.1(a), there can be many spatial representations of the extents of the bounding rectangles. One such representation of the bounding rectangles is presented in the Figure 4.3(a) and the corresponding R+-Tree is shown in the Figure 4.3(b). The structure of the R+-Tree of order (m,M) is such that all the intermediate nodes and leaf nodes contain between m<=(M/2) and M entries. However, it is not guaranteed until a complicated record insertion and deletion procedure is followed. The root node has at least two entries, if it is not a leaf node. The procedure for building the R+-tree of the Figure 4.3(a) is described here. Initially, the root node consists of R1 and R2 rectangles. The R1 contains line segments a,b,h,i,j,c and the R2 contains line segments c,d,e,f,g,k. Notice that the line segment c is replicated in both the rectangles. The number of entries is much greater than the permissible range, so the rectangles R1 and R2 are further divided into rectangles R3, R4 and R5, R6 respectively. Now, the rectangle R3 contains entries a,b,h and the rectangle R4 contains entries c,b,i,j. The line segment b is replicated in both the sub-rectangles. Similarly, the rectangle R5 contains entries c,d,e and the rectangle R6 contains entries f,g,k. The R+-tree has been investigated over many parallel models such as the SAM [151, 155, 5] and MapReduce [36]. The R+-tree implementation on the MapReduce [36] is considered here for building the R+-tree, and the queries are executed on the index.

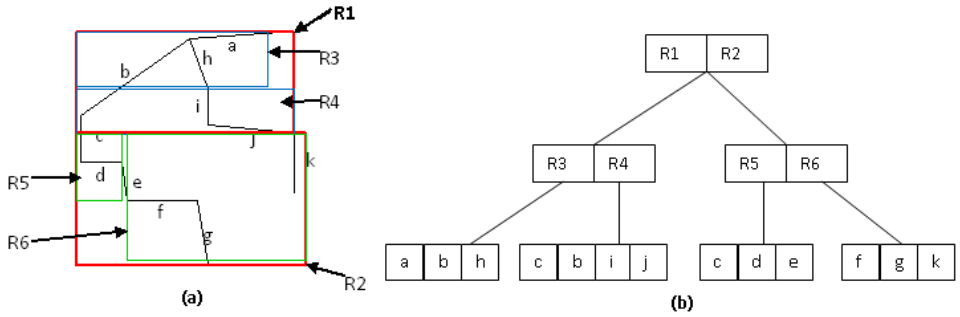

Figure 4.3: The Conceptual View of the R+-Tree Index in the MapReduce (a) The spatial area representation of the bounding rectangle (b) The corresponding R+-Tree for the collection of line segments

4.2 Experimental Set-up

A virtual organization was set-up on fifteen computers, having Ubuntu 12.0 operating system, using the Globus toolkit 4.0 (GT4). All the machines were also configured with Java-6-openjdk and SpatialHadoop The computers used for the experiment had dual core processor 2.1 GHz, 1 GB RAM (2 GB RAM for the Master node of the Grid Cluster) and were assigned class C private IP addresses. The machines are used from different laboratories, connected through switches, and form a start topology LAN that runs at 100 Mbps as shown in Figure 4.4.

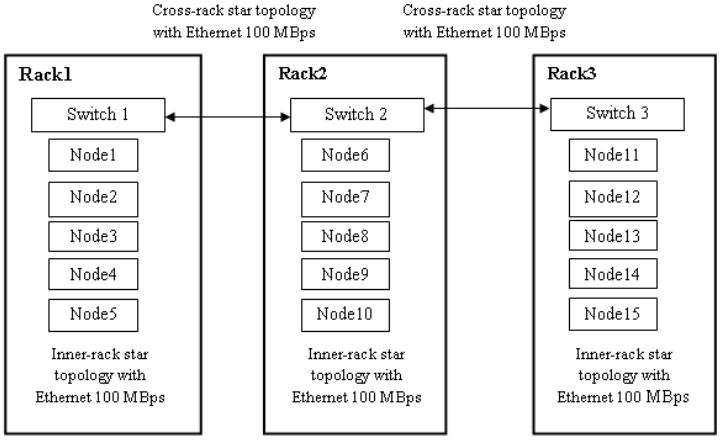

Figure 4.4: Topology of the machines connected in QUiPSHoT Grid-GIS

A detailed description of the hardware configuration, operating system, Java platform, and SpatialHadoop based on Hadoop-1.0.3 [23], used in the experimental set-up, is presented in 4.1.

Table 4.1: Configuration of machines used for building the QUiPSHoT Grid-GIS

Sr. No.	**Hostname**	**IP Address**	**Processor**	**Primary Memory**	**Operating System**	**Ethernet Speed**	**Java Version**	**Hadoop Version**	**Globus Toolkit**
1	Node1	192.168.10.11	Dual Core @ 2.1 GHz	2GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
2	Node2	192.168.10.12	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
3	Node3	192.168.10.13	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
4	Node4	192.168.10.14	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
5	Node5	192.168.10.15	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
6	Node6	192.168.10.21	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
7	Node7	192.168.10.22	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
8	Node8	192.168.10.23	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
9	Node9	192.168.10.24	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
10	Node10	192.168.10.25	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
11	Node11	192.168.10.43	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
12	Node2	192.168.10.44	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
13	Node13	192.168.10.45	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
14	Node14	192.168.10.46	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0
15	Node15	192.168.10.47	Dual Core @ 2.1 GHz	1GB	Ubuntu 11.04	100 Mbps	Java-6-Openjdk	SpatialHadoop	Globus Toolkit 4.0

4.3 Datasets

Both real-life and synthetic datasets have been used in experimental works.

4.3.1 Real Life Data

The road intersection data for the five counties of the California state of the tiger/line dataset has been used in the experimentation [107]. The following counties in the state of California have been considered: The Fresno County, Sierra County, Santa Cruz County, Lake County, and Butte County. The comma-separated-value (CSV) data for each county is considered in the experimentation. This data is easily available on [107]. The datasets are converted to pictorial form using QGIS software and shown in Figures 4.5, 4.6, 4.7, 4.8 and 4.9. The dataset are of varying sizes and varying data distribution. The Sierra county dataset is quite uniformly distributed. The Butte, Lake and Santa Cruz county datasets are uniformly distributed, except at a few places where these become non-uniform and dense. The Fresno county dataset is uniformly distributed and dense.

The dataset is used for the proposed QUiPSHoT Grid-GIS framework. The dataset helped to measure various parameters, such as the storage efficiency of the indexes, query execution efficiency, Scalability and availability. The parameters are observed for a large number of test runs on each county dataset for better accuracy. The mean value from these test runs is plotted for presenting the analysis. The whole dataset is approximated with minimum bounding rectangles to represent the spatial data objects. The tiger dataset is relatively nicely distributed. It consist of relatively small rectangles, long roads are divided into smaller segments, that are somewhat clustered around urban areas.

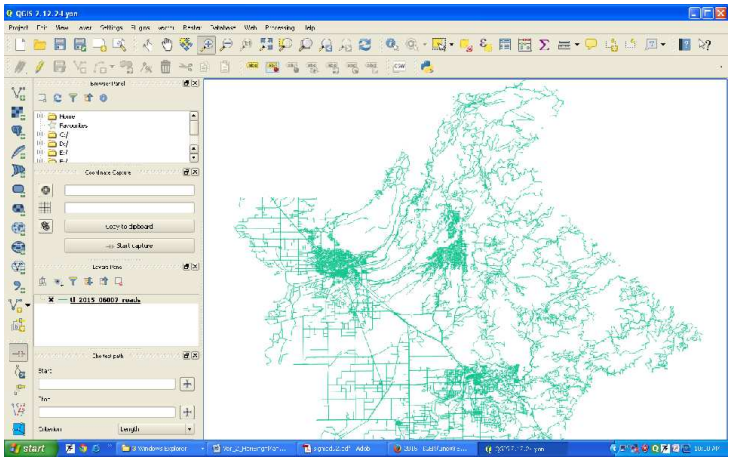

Figure 4.5: Butte County Dataset

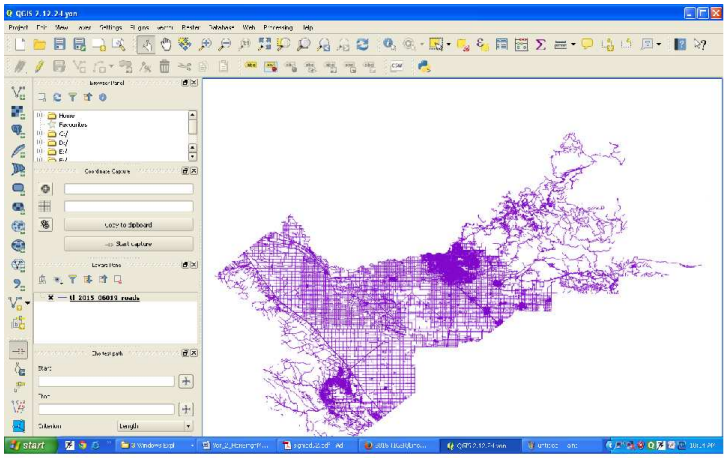

Figure 4.6: Fresno County Dataset

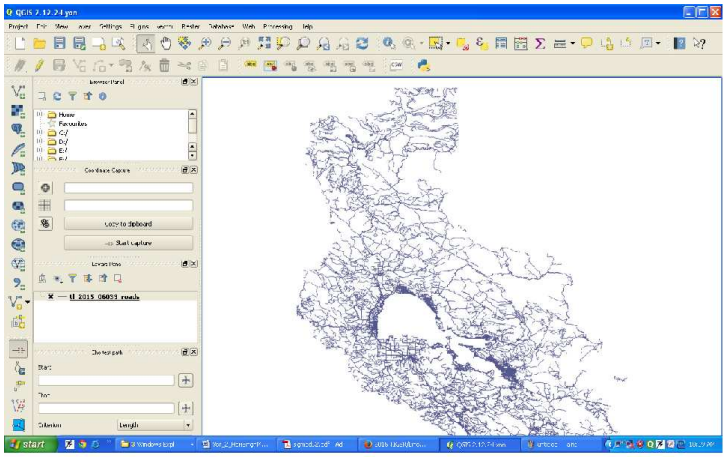

Figure 4.7: Lake County Dataset

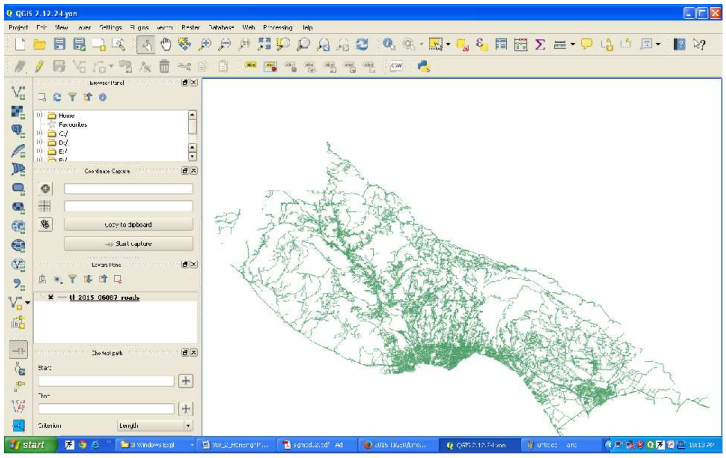

Figure 4.8: Santacruz County Dataset

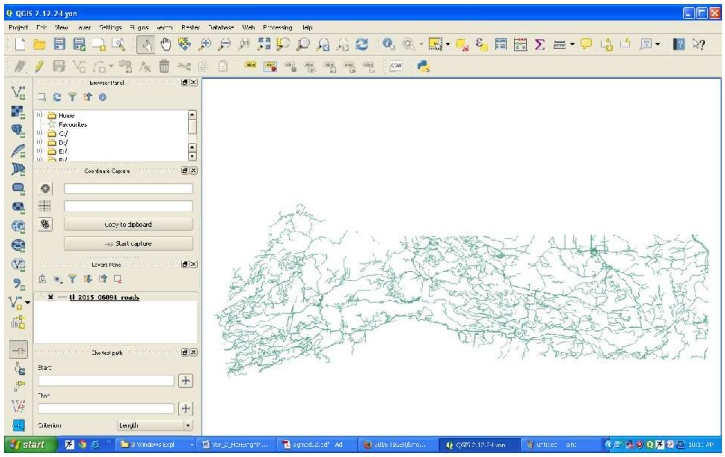

Figure 4.9: Sierra County Dataset

4.3.2 Synthetic Data

To investigate how the different spatial trees perform on more extreme datasets than the Tiger data, we generated a number of synthetic dataset using R language.

(i) **SIZE:** This type of dataset contains uniformly distributed data rectangles around their centers. However, the lengths of the sides of the rectangles vary between 0 and a maximum defined range and theses are uniformly and independently distributed in space. The rectangles whose sides intersect the sides of the bounding rectangle are discarded. One such sample is shown in the Figure 4.10. The dataset helps to investigate the behavior of different indexes for data rectangles of different sizes.

Program: Code to generate a synthetic dataset for the rectangles distributed uniformly and the lengths of their sides uniformly and independently distributed between 0 and max side (Generated in R).

```
x1 <- runif(50,10,490)
print(x1)
hline y1 <- runif(50,10,490)
print(y1)
hline x2 <- runif(50,10,490)
print(x2)
hline y2<- runif(50,10,490)
print(y2)
plot(c(0, 500), c(0, 500), type= "n", xlab = "x-axis", ylab = "y-axis")
for(i in 1:50)
rect(x1[i],y1[i],x2[i],y2[i])
```

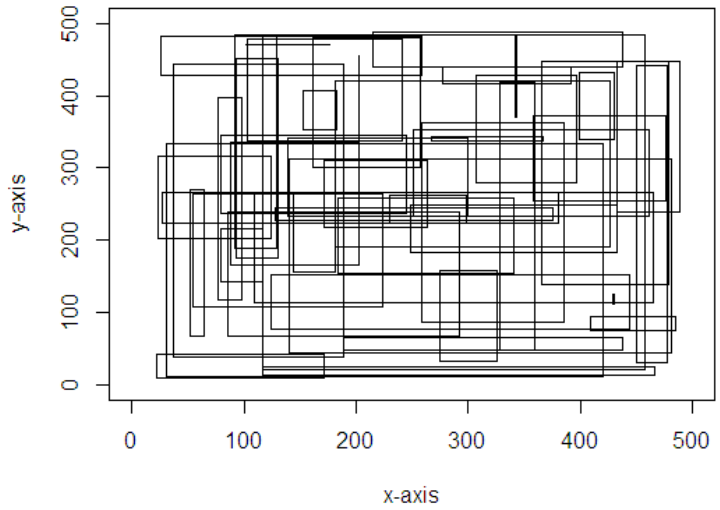

Figure 4.10: Synthetic dataset for the rectangles distributed uniformly and the lengths of their sides uniformly and independently distributed between 0 and max side (Generated in R).

(ii) **Aspect:** The aspect of a rectangle is the fixed ratio of its length to breadth. The dataset contains rectangles in a fixed aspect ratio, however the rectangles are uniformly distributed around their centers. The longer side may be taken horizontally or vertically with equal probability. The dataset helps to investigate the behavior of different indexes for data rectangles with different aspect ratio. One such sample is shown in Figure 4.11.

Program: Code to generate synthetic dataset with aspect=10

```
x1 <- runif(500,10,410)
print(x1)
y1 <- runif(500,10,410)
print(y1)
plot(c(0, 500), c(0, 500), type= "n", xlab = "x-axis", ylab = "y-axis")
for(i in 1:500)
if(i%%2==0)
x2[i] = x1[i] + 100
y2[i] = y1[i] + 10
rect(x1[i], y1[i],x2[i],y2[i], border = "red")
else
hline x2[i] = x1[i] +10
y2[i] = y1[i] +100
rect(x1[i], y1[i],x2[i],y2[i],border = "blue")
```

print(x2)
print(y2)

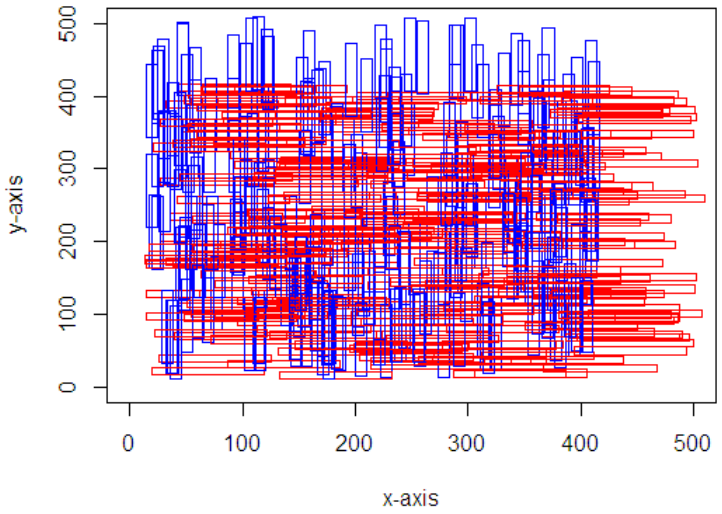

Figure 4.11: Synthetic dataset with aspect=10.

(iii) **Skewed:** The dataset in which data rectangles are inclined or concentrated in a particular area or along an axis is termed as skewed data. In a multidimensional dataset, data rectangles may be uniformly distributed along one dimension; however, these may be skewed along a different dimension, as shown in the Figure 4.12 . The dataset helps to investigate the behavior of different indexes for skewed data.

Program: Code to generate synthetic dataset where the rectangle centers are skewed along one side.

x1 <-(rbeta(500,5,2))
x1=x1*1000
print(x1)
y1 <-(rbeta(500,5,2))
y1=y1*1000
print(y1)
x2 <-(rbeta(500,5,2))
x2=x2*1000
print(x2)
y2 <-(rbeta(500,5,2))
y2=y2*1000
print(y2)
plot(c(0, 1000), c(0, 1000), type= "n", xlab = "x-axis", ylab = "y-axis")

for(i in 1:500)
rect(x1[i],y1[i],x2[i],y2[i])

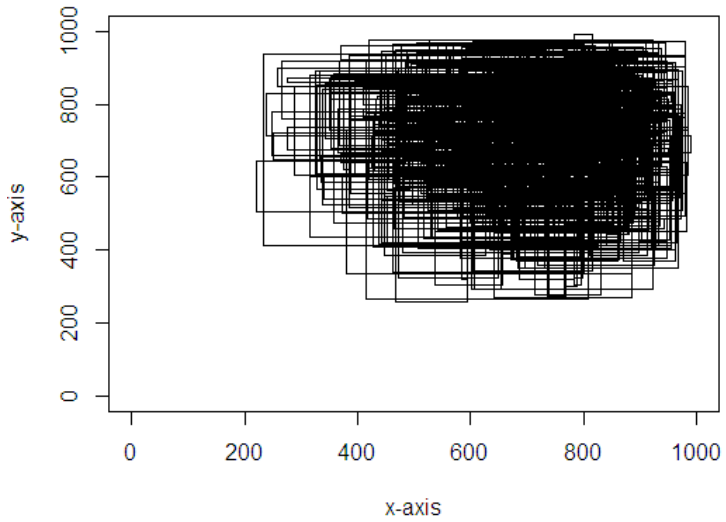

Figure 4.12: Synthetic dataset where the rectangle centers are skewed along one side.

(iv) **Clustered:**A clustered dataset contains data rectangles which are spread in groups. The concentration of these groups, called clusters, is uniformly distributed, as shown in the Figure 4.13. The dataset helps to investigate the behavior of different indexes for clustered data. Program: Code to generate synthetic dataset for clustered data with aspect 10.

x1 <- runif(500,10,410)
print(x1)
y1 <- runif(500,10,410)
print(y1)
plot(c(0, 500), c(0, 500), type= ”n”, xlab = ”x-axis”, ylab = ”y-axis”)
for(i in 1:500)
if(x1[i]<150.0 || x1[i]>350)
if(i%%2==0)
x2[i] = x1[i] +100
y2[i] = y1[i] +10
rect(x1[i], y1[i],x2[i],y2[i],border = ”red”)
else
x2[i] = x1[i] +10
y2[i] = y1[i] +100
rect(x1[i], y1[i],x2[i],y2[i],border = ”blue”)
print(x2)

print(y2)

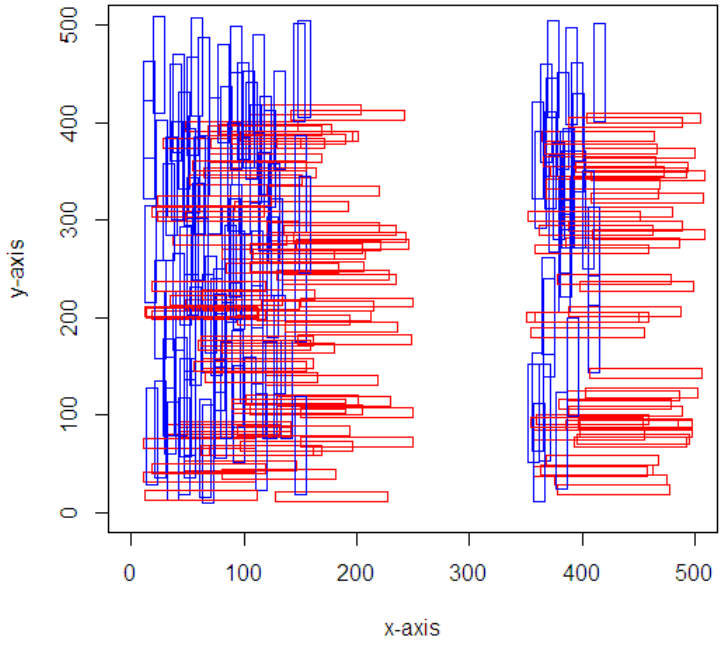

Figure 4.13: Synthetic dataset for clustered data with aspect 10.

4.4 Parallel Spatial Queries

The efficiency of the proposed index is demonstrated through implementing and executing spatial queries, such as line search and range search, on the proposed index. The query algorithms in MapReduce are as follow:

4.4.1 Line Search Query

The line search query returns true for the input line object to be searched when the search process finds the line object in the index. The following Algorithm 4.9 splits the input lines spatial data and then passes the split data (line_id coordinates((xmin,ymin), xmax,ymax))) to the map function. The map function carries out the line search using the conventional PMR Quadtree line search [104]. The reduce function counts the total number of line objects.

Algorithm 4.9 Line search query

INPUT: Set of spatial lines
OUTPUT: The boolean value true for lines which are found in the index
Step1: The set of lines is partitioned into splits and input to the program.
Step2: In the Map function, line search query is implemented. For a particular (line(xmin.ymin),(xmax,ymax)), the function returns 1 if found, otherwise returns -1.
Step3: In the reduce function, the input from the map function is (Found,list(identifiers of existing lines). Count the number of lines that are found and output the result onto HDFS.
Step4: Set the path for input and output directories, and then start up MapReduce.

4.4.2 Range Search Query

The Algorithm 4.10 finds out all the line objects that intersect or lie in a particular input range or region. Firstly, it finds the index space that intersects with or includes the query range and then, the Map function passes the index root node of the sub-tree to the Reduce function. The Reduce function computes all the line objects which overlap with the query range and merge the result to form the overall query result.

Algorithm 4.10 Range search query

INPUT: MBR of the query range
OUTPUT: The line objects
Step1: Split the region into several parts.
Step2: For the overlapped area of the split part of the region and the index space, Map function outputs the (index, split part of the region).
Step3: Reduce function executes range query on the index space provide by the Map function as input to the Reduce.
Step4: Set the path for input and output directories, and then start up MapReduce.
Step5: Merge the outputs of the split parts of the region, and return the whole result.

Chapter 5

Test and Demonstration of the QUIPSHoT Grid-GIS

This chapter presents tested results and demonstrations of the proposed QUiPSHoT Grid-GIS architecture and framework. Section 5.1 demonstrates superiority of integrated Grid and MapReduce technologies in the proposed architecture and framework. Sections 5.2, 5.3 and 5.4 present tests and demonstrations of the effect of H-bucket PMR Quadtree index, Parallel Hilbert TGS R-Tree and Parallel Priority R-Tree on the proposed Grid-GIS system. The thesis work deeply analyzes the proposed architecture and framework with respect to parameters storage eficiency, execution efficiency, scalability and availability on the real life spatial data [107] described in Section 4.3.1 and the synthetic spatial dataset described in Section 4.3.2.

5.1 Test and Demonstration of QUIPSHoT Grid-GIS Towards Spatial Queries

This section compares the Central, Grid and Basic QUiPSHoT Grid-GIS and, presents effectiveness of the Basic-QUiPSHoT Grid-GIS through tests and demonstrations with line and range search spatial queries. The Central system uses distributed data spread over computing nodes and accessed through general client-server architecture. The Grid system uses a grid of computing nodes. The Basic-QUiPSHoT Grid-GIS consists of an integrated system of grid and MapReduce technologies. Efficiency comparison of the Central, Grid and Basic QUiPSHoT Grid-GIS on the evaluation parameters and line search query has been conducted and average response time, from a set of ten experimental runs, has been plotted in Figure 5.1. The Basic-QUiPSHoT Grid-GIS utilizes default indexing of the Hadoop architecture. The line search query has been tested with a parallel task count of 5, 10, 15, 20, 25, 35 and 40. An analysis of the three types of system reveals a reduction of approximately 36-41% average response time of the Grid system over the Central system. It has also been observed that gain in execution efficiency is more for queries with higher number of parallel task counts. A further decrease in the response

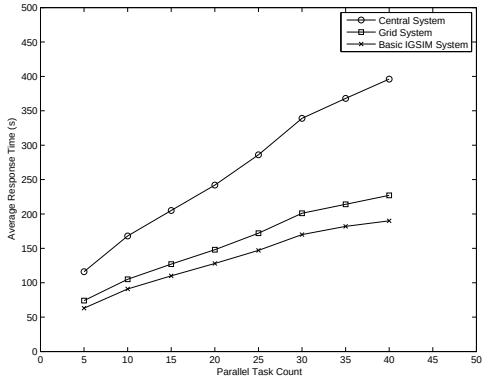

Figure 5.1: Execution efficiency comparison of the Central, Grid and Basic-QUiPSHoT Grid-GIS framework for a line search query

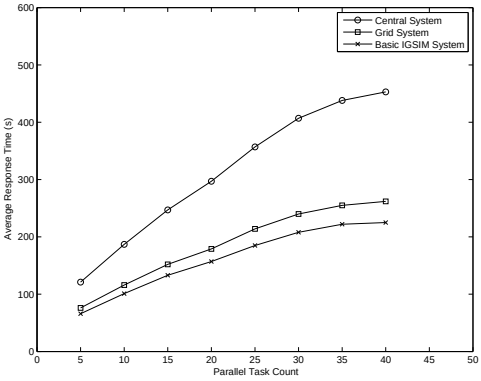

Figure 5.2: Execution efficiency comparison of the Central, Grid and Basic-QUiPSHoT Grid-GIS framework for a range search query

time, by 45-52%, has been observed for the Basic-QUiPSHoT Grid-GIS in comparison to the Central system. In a quite similar pattern to the Grid system, the Basic-QUiPSHoT Grid-GIS also shows a higher gain in efficiency for queries with higher number of parallel task counts. These observations clearly show the superiority of the Basic QUiPSHoT Grid-GIS among the three. A quite similar output is observed for range search queries over the Central, Grid and Basic-QUiPSHoT Grid-GIS, as presented in Figure 5.2. An increased performance of 37-42 % and 42-50% has been recorded in the Grid system and Basic QUiPSHoT Grid-GIS, over the Central system, respectively. Similar to line search query, the performance of the Grid system and Basic QUiPSHoT Grid-GIS framework has been found to be more when parallel task count increases.

5.2 Test and Demonstration of the Parallel H-bucket PMR Quadtree Index in QUIPSHoT Grid-GIS

This section presents tested results and demonstrations of the parallel H-bucket PMR Quadtree index in QUiPSHoT Grid-GIS on parameters storage efficiency, execution efficiency and scalability. The availability parameter is discussed in Section 5.4. The proposed index has been compared with the R+-Tree in QUiPSHoT Grid-GIS environment.

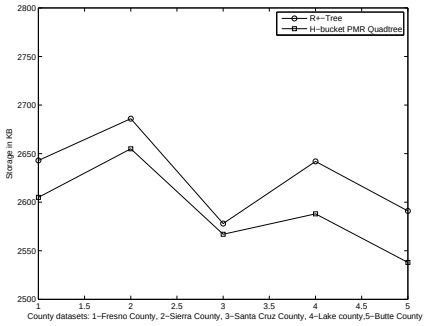

Figure 5.3: Storage required

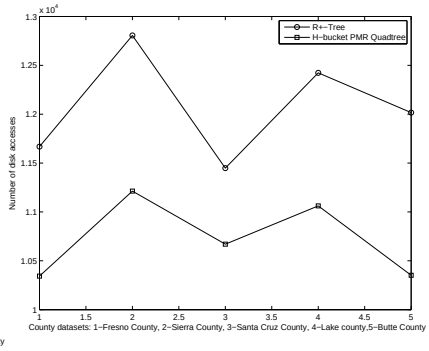

Figure 5.4: Number of disk accesses

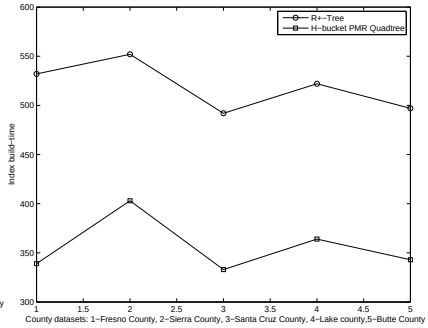

Figure 5.5: Index build-time

5.2.1 Bulk-loading Cost of the Index

Costs of building indexes with respect to parameter storage, number of disk accesses and execution time is analyzed on a cluster of ten computing nodes. A set of 30 experiments has been conducted, three experiments for each index (R+-tree and H-bucket PMR Quadtree) and for each county dataset (five counties), for analyzing the results. Figure 5.3 shows that storage requirement of the MapReduce implemented R+-tree and H-bucket PMR Quadtree is almost similar, but the later one is slightly better than the former. The disk access time of the H-bucket PMR Quadtree is almost similar to the R+-tree, as shown in Figure 5.4, but the former takes a slightly less number of disk accesses as compared to the latter for all the polygon maps. However, the data structure building time of former has been found to be significantly smaller as compared to the latter, and the former takes approximately 20-50% less time than the latter. A comparison of the index build-time is shown in Figure 5.5. It is due to the regular disjoint decomposition in case of H-bucket-PMR Quadtree, as the decision to split an overflowing node effectively requires only two candidate split axis/coordinate pairs. The R+-tree requires testing a possibly large number of split axis/coordinate pairs in determining a locally optimal node split. This node split is an iterative work that depends on the population of objects in a tree node. A large number of clipping operations are required before determining which part of the line is associated with the two nodes resulting from a split. While, in the case of H-bucket-PMR Quadtree, clipping operations are constant due to a regular disjoint decomposition of spatial space.

5.2.2 Execution Efficiency w.r.t. Node Capacity

The execution efficiency of the proposed parallel index H-bucket PMR Quadtree for the existing serial programming algorithm bucket PMR Quadtree index is tested and

demonstrated in the proposed integrated framework. The index building time or bulk-loading time for the spatial data on the proposed spatial index and the spatial queries, line search query and range search query, are used to check the efficiency of execution.

5.2.2.1 Execution Efficiency of Index Building:The Effect of Tree Node/Bucket Size on Index Building Time

The spatial index building is dependent on the bucket size or node capacity of the index. The bucket size or the node capacity is the data holding capacity of a tree node and is decided by the node size of an index tree. It significantly impacts the performance of queries. A set of 180 experiments has been conducted, three experiments for each node capacity (16KB, 32 KB, 64 KB, 128 KB, 256 KB and 512 KB) and for each index, for each of the five counties. The index building time, for the H-bucket PMR Quadtree index and the R+-tree index in MapReduce, decreases with increasing the bucket size up to 64 KB. However, the results in Figures 5.6 and 5.7 show that the index building time decreases sharply for both the indexes on a cluster size of 10, when the node size increases gradually up to a certain limit. It is due to the reduced computations required for testing split/axis coordinate pairs with an increase in node capacity, but the behavior is limited till the size of data packets being transferred in HDFS becomes 64 KB. When the tree node size is increased further and becomes larger than 64 KB, the number of network transfers also increases. For tree node sizes more than 128 KB, 256 KB and 512 KB, the number of network transfers continue to grow. So, the node size calibration is important and setting it below 64 KB produces optimized query response performance. Further, a comparison between the two indexes in Figures 5.8, 5.9, 5.10, 5.11, 5.12 show that the index build time of the H-bucket PMR Quadtree is faster than the R+-tree in QUiPSHoT Grid-GIS by a factor of 1.39-2.01, 1.27- 2.00, 1.31-2.36, 1.29- 2.24, 1.21- 2.26 for the counties Fresno, Sierra, Santa Cruz, Lake and Butte, respectively. In overall, the H-bucket PMR Quadtree index build-time is faster than the R+-tree index build time in QUiPSHoT Grid-GIS by an average factor of approximately 1.29 - 2.17.

5.2.2.2 Execution Efficiency of Spatial Queries (Line and Range Search Queries)

A set of 180 experiments has been conducted, three experiments for each node capacity (16KB, 32 KB, 64 KB, 128 KB, 256 KB and 512 KB) and for each index, for each of the five counties, for a cluster size of 10 nodes. The spatial queries, line search and range search, are executed for the H-bucket PMR Quadtree index and R+-tree index

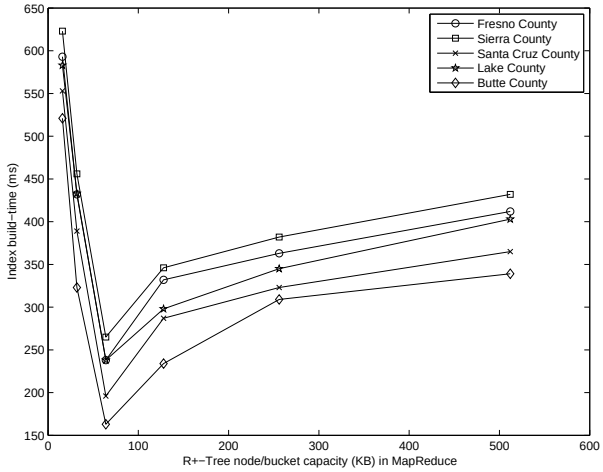

Figure 5.6: R+-Tree index build-time in MapReduce

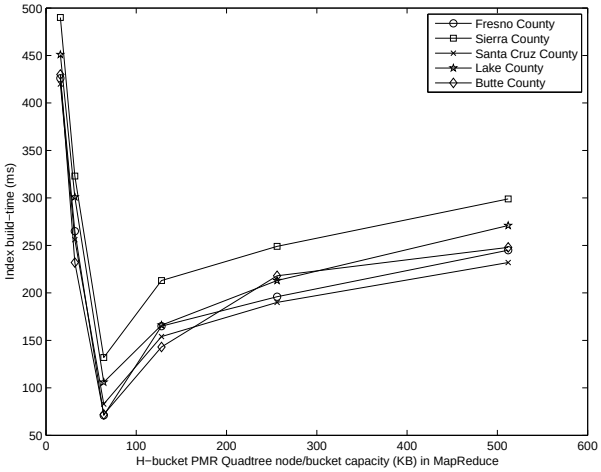

Figure 5.7: H-bucket PMR Quadtree index build-time in MapReduce

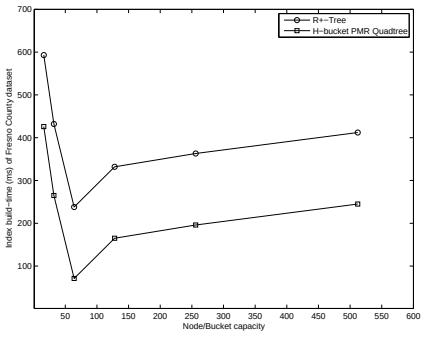

Figure 5.8: Index build-time for Fresno County

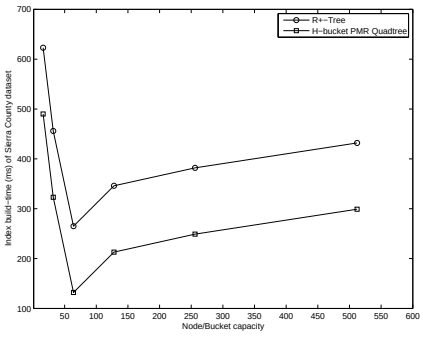

Figure 5.9: Index build-time for Sierra County

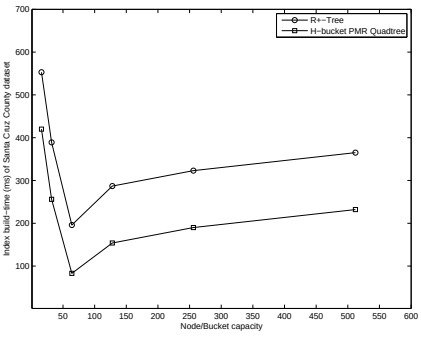

Figure 5.10: Index build-time for Santa Cruz County

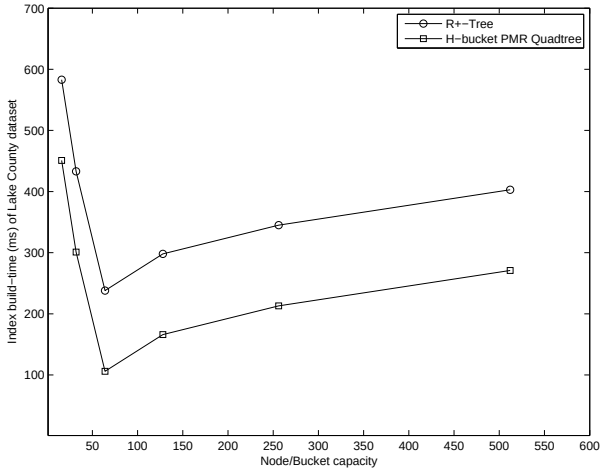

Figure 5.11: Index build-time for Lake County

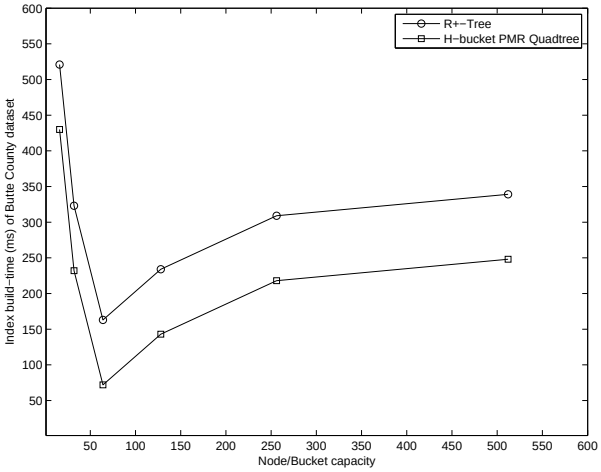

Figure 5.12: Index build-time for Butte County

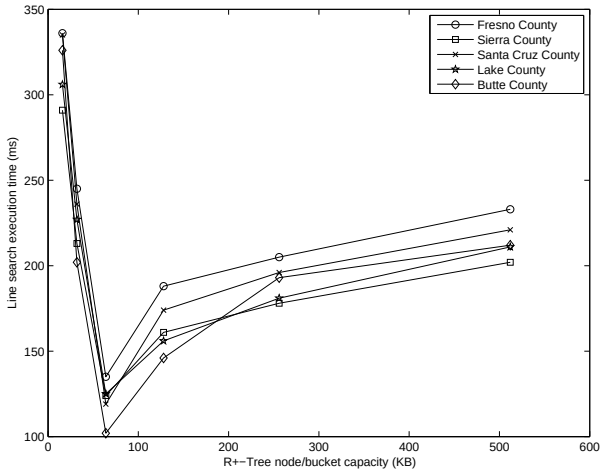

Figure 5.13: Line search query for the R+-Tree in QUiPSHoT Grid-GIS for varying node capacity over a cluster of size 10.

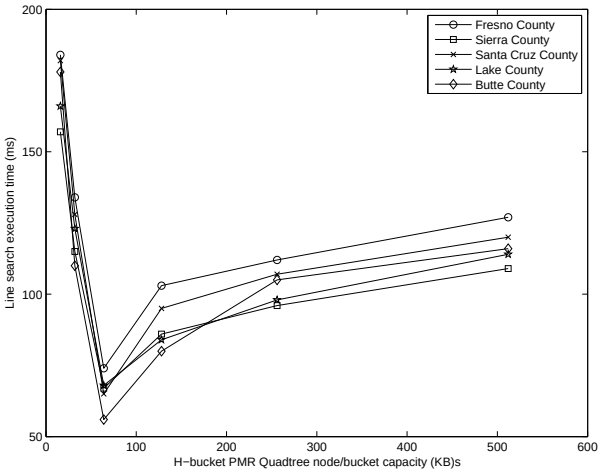

Figure 5.14: Line search query for the H-bucket PMR-Quadtree in QUiPSHoT Grid-GIS for varying node capacity over a cluster of size 10.

in the QUipSHoT Grid-GIS framework. The comparison of query execution efficiency of different datasets show an almost similar pattern, as has been seen for building the spatial indexes in Section 5.2.2.1. The query execution time decreases with increasing the bucket size up to 64 KB for both the indexes over the five real datasets, as shown in Figures 5.13 and 5.14 for a line search query and Figures 5.20 and 5.21 for a range search query. The reason for the pattern is same. It is due to a reduced computation required for testing split/axis coordinate pairs with an increase in node capacity, but the behavior is limited till the size of data packets being transferred in HDFS becomes 64 KB. When the tree node size increases further and becomes larger than 64 KB, the number of network transfers increases. For tree node sizes more than 128 KB, 256 KB and 512 KB, the number of network transfers continue to grow. So, the node size calibration is important and setting it below 64 KB produces the optimized query response performance. Spatial query search with the two indexes on the real spatial dataset reveals a better query efficiency, line search and range search, for the H-bucketPMR-Quadtree index, as shown in Figures 5.15, 5.16, 5.17, 5.18, 5.19 and 5.22, 5.23, 5.24, 5.25, 5.26, respectively.

5.2.3 Scalability: The Effect of Cluster Size on the Search Queries Execution Time

A set of 450 experiments has been conducted, three experiments for different sizes of the cluster varying from 1 to 10 computing nodes and for each county dataset, for a tree-node/bucket size of 64 KB. The mean values are used to draw the graphs presented

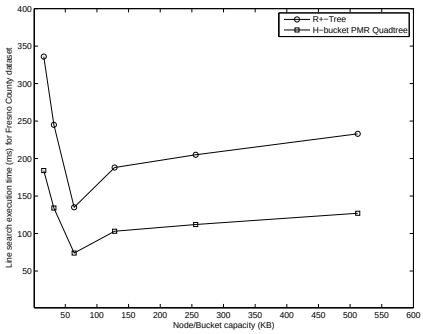

Figure 5.15: Line search execution time vs node/ bucket capacity for Fresno County

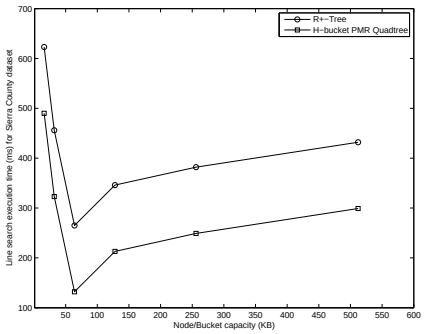

Figure 5.16: Line search execution time vs node/ bucket capacity for Sierra County

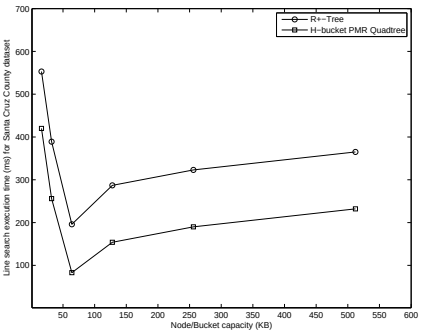

Figure 5.17: Line search execution time vs node/ bucket capacity for Santa Cruz County

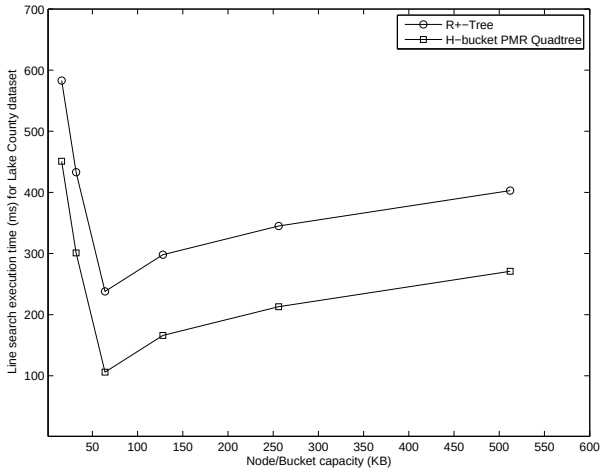

Figure 5.18: Line search execution time vs node/bucket capacity for Lake County

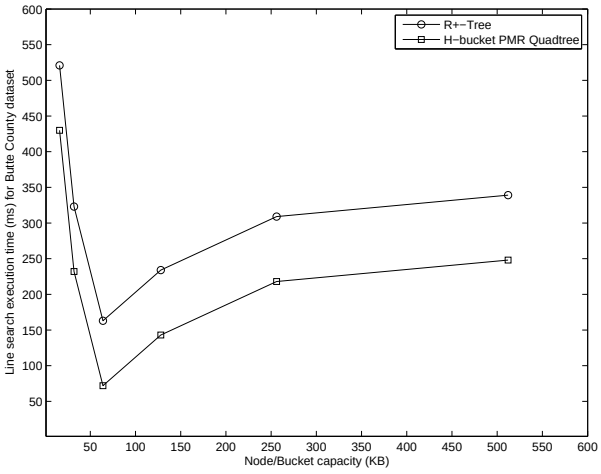

Figure 5.19: Line search execution time vs node/bucket capacity for Butte County

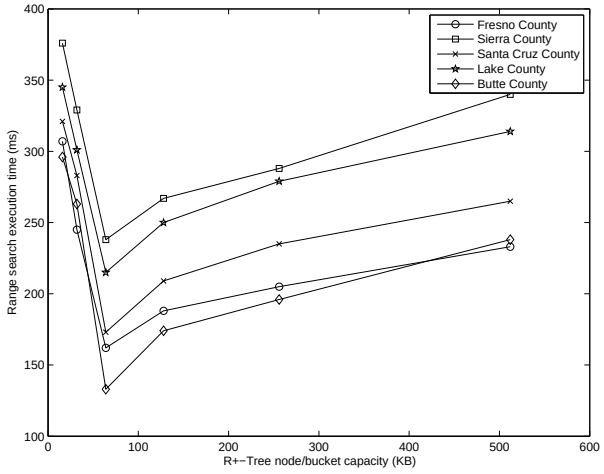

Figure 5.20: Range search query for the R+-Tree for varying node capacity over a cluster of size 10.

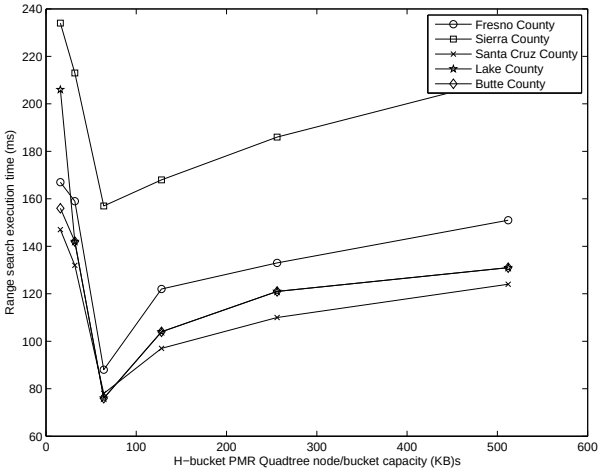

Figure 5.21: Range search query for the H-bucketPMR-Quadtree for varying node capacity over a cluster of size 10.

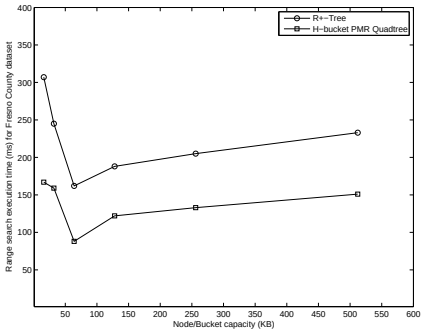

Figure 5.22: Range search execution time vs node/ bucket capacity for Fresno County

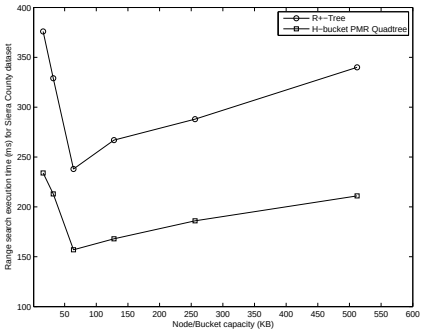

Figure 5.23: Range search execution time vs node/ bucket capacity for Sierra County

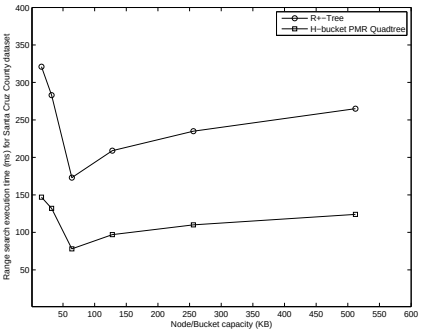

Figure 5.24: Range search execution time vs node/ bucket capacity for Santa Cruz County

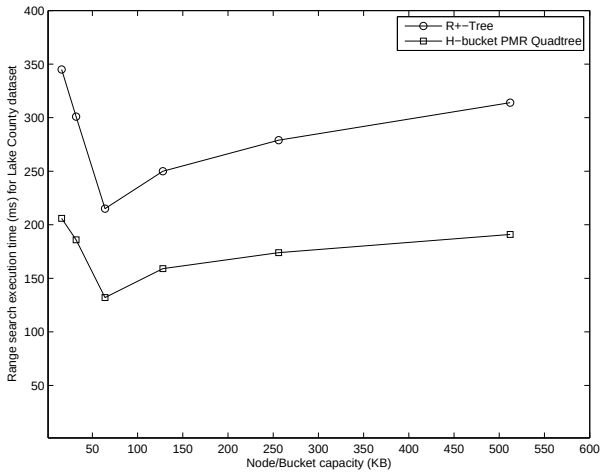

Figure 5.25: Range search execution time vs node/ bucket capacity for Lake County

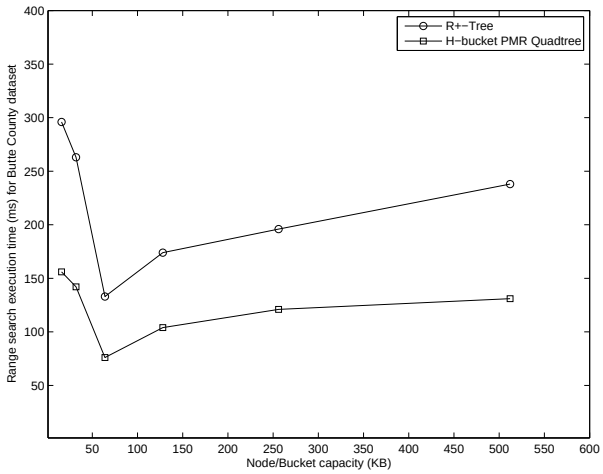

Figure 5.26: Range search execution time vs node/ bucket capacity for Butte County

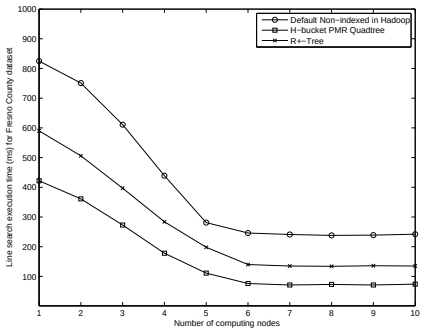

Figure 5.27: Line search execution time vs cluster size for Fresno County

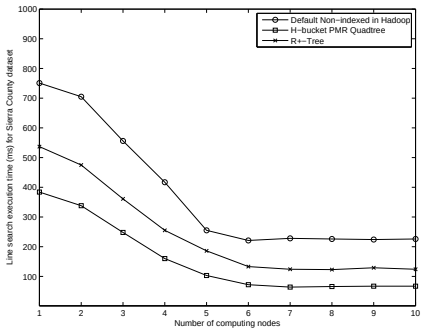

Figure 5.28: Line search execution time vs cluster size for Sierra County

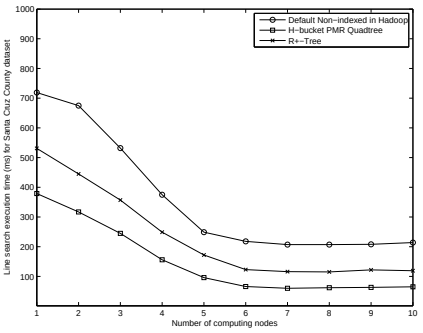

Figure 5.29: Line search execution time vs cluster size for Santa Cruz County

in Figures 5.27, 5.28, 5.29, 5.30, 5.31 and Figures 5.32, 5.33, 5.34, 5.35, 5.36 for line search and range search queries, respectively. The query search behaves just like a search operation on a stand-alone system, when there is a single node in the cluster. But on the addition of a second node in the cluster, the search time decreases slightly. With the addition of a third node, there is a sharp decrease in the execution time. With the addition of fourth, fifth and sixth node, the search time shows a continuous decreasing trend. However, on addition of seventh, eighth, ninth and tenth node, the search time does not decrease further and it remains almost constant. A slight decrease in the execution time, when second node is added, is due to the increased shuffling of intermediate data and files, which overcomes the performance gain due to parallelism in the cluster. However, later on when more nodes (up to six in number) are added, the effect of computation due to clustering overweighs the shuffling of intermediate data. But, later on there does not seem any change in the execution time when more nodes are added (from 7th to 10th node), it is so because the size of the data is considered large enough for keeping busy the six computing nodes. On further adding more nodes to the cluster, these nodes sit idle and do not contribute in reducing the execution time. Figures 5.27, 5.28, 5.29, 5.30, 5.31 show a line search query execution time for non-indexed-Hadoop and the two indexes. A significant performance gain is observed for running queries once the indexes are established. A similar kind of observations is noticed for range search queries, shown in Figures 5.32, 5.33, 5.34, 5.35, 5.36. From the analysis, it can be stated that the H-bucket PMR Quadtree query execution time is better than 1-4-1.9 times as compared to the MapReduce based R+-tree and the MapReduce based R+-tree query execution time is better than 1-4-1.8 times as compared to the default non-indexed Hadoop. In overall, the query performance becomes better for indexing dataset and the proposed index is better than the state-of-the-art MapReduce based R+-tree index.

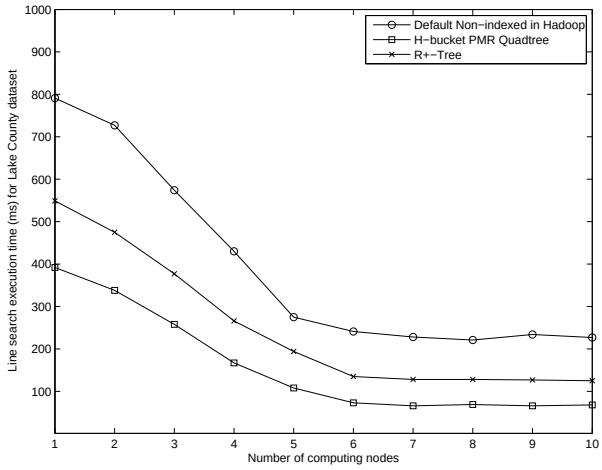

Figure 5.30: Line search execution time vs cluster size for Lake County

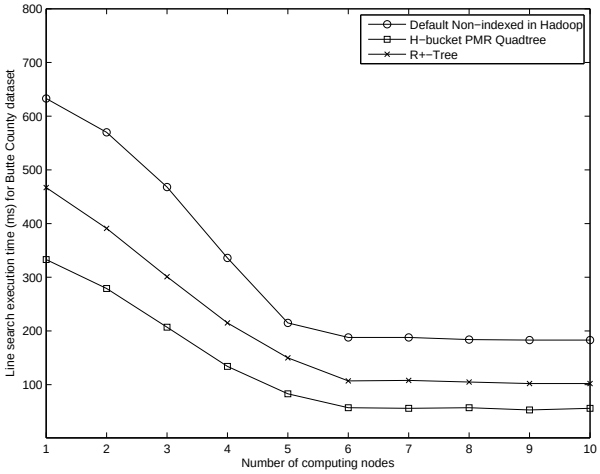

Figure 5.31: Line search execution time vs cluster size for Butte County

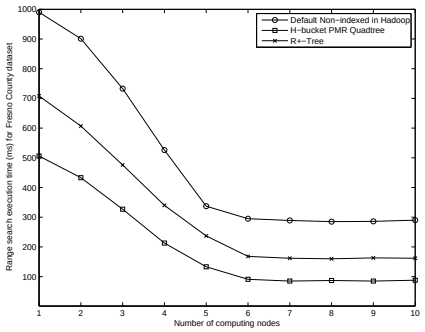

Figure 5.32: Range search execution time vs cluster size for Fresno County

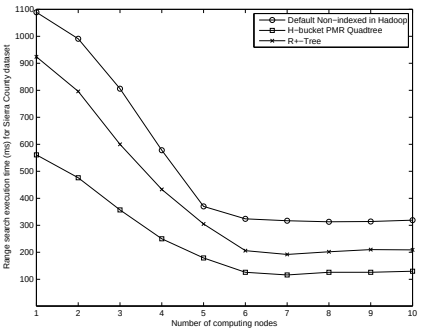

Figure 5.33: Range search execution time vs cluster size for Sierra County

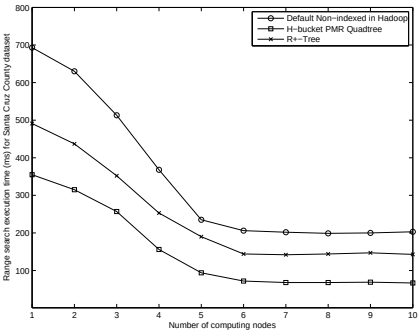

Figure 5.34: Range search execution time vs cluster size for Santa Cruz County

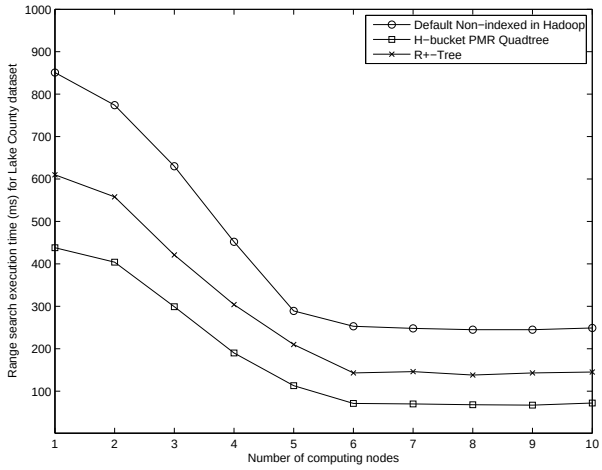

Figure 5.35: Range search execution time vs cluster size for Lake County

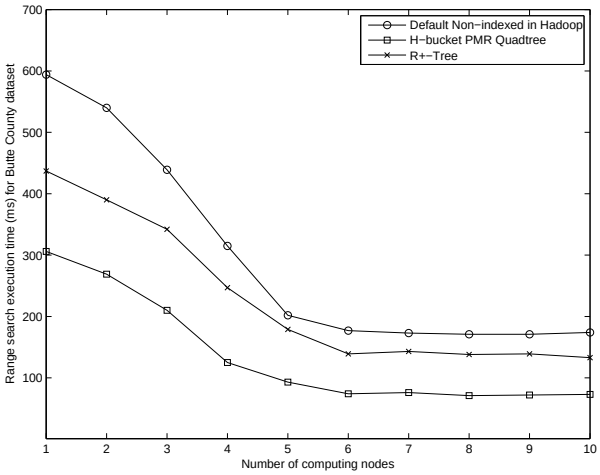

Figure 5.36: Range search execution time vs cluster size for Butte County

5.3 Test and Demonstration of the Parallel Hilbert TGS R-Tree Index on QUIPSHoT Grid-GIS

This section presents tested results and demonstrations of the parallel Hilbert TGS R-Tree index on the QUiPSHoT Grid-GIS for parameters, bulk-loading cost of the index, execution efficiency and scalability. The availability parameter is discussed in Section 5.4.

5.3.1 Bulk-loading Cost of the Index

The costs of building the index with respect to parameter storage, number of disk accesses and index building time is analyzed on the QUiPSHoT Grid-GIS that forms a cluster of ten computing nodes. A set of 15 experiments was conducted, three experiments for the Parallel Hilbert TGS-R-Tree index and for each county dataset (five counties), for analyzing the results. Figure 5.37 shows the storage requirement of the MapReduce implemented Hilbert TGS R-Tree. The Fresno, Sierra, Santacruz, Lake and Butte county takes 2657 KB, 2701 KB, 2579 KB, 2644 KB and 2593 KB of storage, respectively. Similarly, the number of disk accesses for the counties in the same sequence are 28090, 30832, 27562, 29885 and 28935, respectively, as shown in the Figure 5.38. The Hilbert TGS R-Tree build time of the counties in the same sequence are 1227 ms, 1287 ms, 1088 ms, 1156 ms and 1195 ms, respectively, as shown in Figure 5.39.

A comparison of the parallel Hilbert TGS R-Tree with another proposed spatial index Parallel Priority R-Tree in QUiPSHoT Grid-GIS is presented in 5.4.

5.3.2 Execution Efficiency w.r.t. Node Capacity

The execution efficiency of the proposed parallel index Parallel Hilbert TGS R-Tree is tested and demonstrated in the proposed integrated framework. The index building time or bulk-loading time of spatial data in the proposed spatial index and spatial queries, line search and range search queries, are used to check the efficiency of execution.

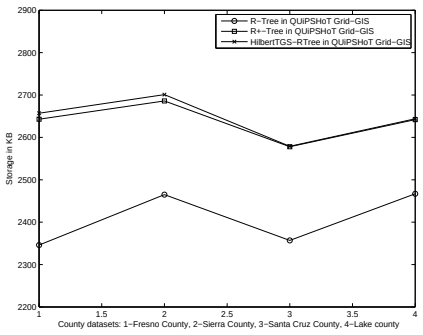

Figure 5.37: Storage utilization of R-Tree, R+-Tree and Parallel TGS R-Tree in QUiPSHoT Grid-GIS for five county spatial dataset

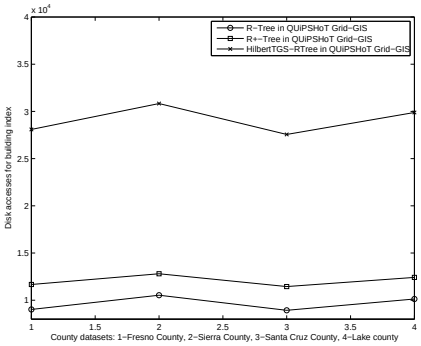

Figure 5.38: Number of disk accesses required for building R-Tree, R+-Tree and Parallel TGS R-Tree in QUiPSHoT Grid-GiS for five county spatial dataset

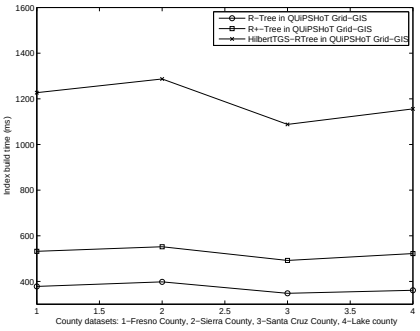

Figure 5.39: Index build-time of R-Tree, R+-Tree and Parallel TGS R-Tree in QUiPSHoT Grid-GIS

5.3.2.1 Execution Efficiency of Index Building: The Effect of Tree Node/Bucket Size on Index Building Time

The spatial index building is dependent on the bucket size or node capacity. The bucket size or node capacity is the data holding capacity of a tree node and is decided by the node size of an index tree. It significantly impacts the performance of queries. A set of 75 experiments has been conducted, three experiments for each node capacity (16KB, 32 KB, 64 KB, 128 KB, 256 KB and 512 KB), for the Parallel Hilbert TGS R-Tree in QUiPSHoT Grid-GIS, for each of the five counties.

The index building time for the R+-Tree and Parallel Hilbert TGS R-Tree in the QUiPSHoT Grid-GIS decreases with increasing bucket size up to 64 KB. However, the results in Figure 5.41 and 5.40 for the former and later index show that index building time decreases sharply for a cluster size of 10, when the node size increases gradually up to a limit. It is due to the reduced computations required in the construction of the spatial index with increasing node capacity, but the behavior is limited till the size of data packets being transferred in HDFS becomes 64 KB. When the tree node size increases further and becomes larger than 64 KB, the amount of network transfers increase. For tree node sizes more than 128 KB, 256 KB and 512 KB, the number of network transfers continue to grow. So, the node size calibration is important and setting it below 64 KB produces the optimized query response performance over the cluster.

A comparison of the parallel Hilbert TGS R-Tree with parallel Priority R-Tree in QUiPSHoT Grid-GIS is presented in Section 5.4.

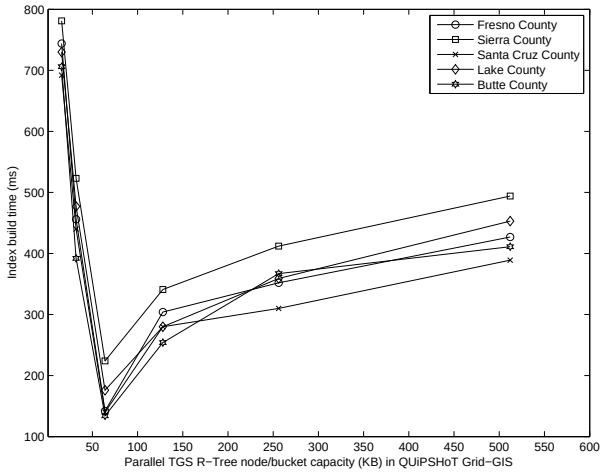

Figure 5.40: The Parallel TGS R-Tree build-time for five county dataset with varying node capacity/bucket capacity

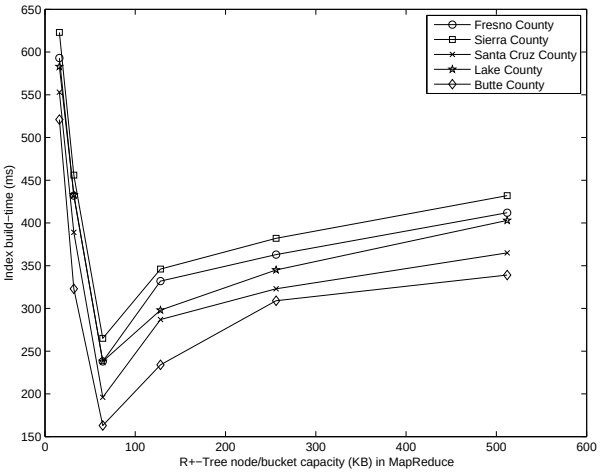

Figure 5.41: The Parallel R+-Tree build-time for five county dataset with varying node capacity/bucket capacity

5.3.2.2 Execution Efficiency of Spatial Queries (Line and Range Search Queries)

Ten experimental runs has been used to compare the spatial query response time of R-Tree, R+-Tree and Hilbert TGS R-Tree in the QUiPSHoT Grid-GIS. It has been observed that line search query response time of the Parallel Hilbert TGS R-Tree is the best among the three indexes on the QUiPSHoT Grid-GIS. The following observations have been noticed for varying task count from 5, 10, 15, 20, 25, 30, 35 and 40 for line search query. The R-Tree index brings 4-6% gain in execution efficiency over the Basic QUiPSHoT Grid-GIS. The R+-Tree performs better than the R-Tree by 8-11% and the Hilbert TGS R-Tree performs even 5-8% better than R+-Tree. These observations are shown pictorially in Figure 5.42. Similarly, the performance of range search query for a similar task count set-up has been observed. The R-Tree performs by 5-7% better than the Basic QUiPSHoT Grid-GIS. The R+-Tree performs 9-12% better than the R-Tree. The parallel Hilbert TGS R-Tree performs 3-6% better than the R+-Tree. It is shown in Figure 5.43. It is again due to the better arrangement and placement of spatial objects in space, which makes the query execution fast.

A set of 75 experiments has been conducted, three experiments for each node capacity (16KB, 32 KB, 64 KB, 128 KB, 256 KB and 512 KB) and the parallel Hilbert TGS R-Tree, for each of the five counties, for a cluster size of 10 nodes. The spatial queries, line search and range search, are executed for the H-bucket PMR Quadtree index and the R+-tree index in the QUipSHoT Grid-GIS framework.

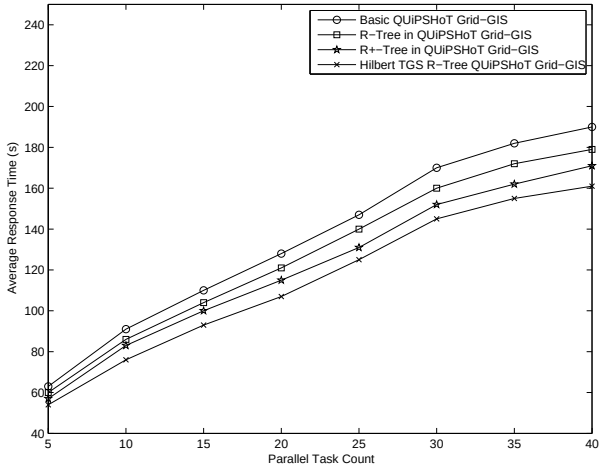

Figure 5.42: Efficiency comparison of Basic QUiPSHoT Grid-GIS and, R-Tree, R+ -Tree and Hilbert TGS R-Tree in QUiPSHoT Grid-GIS for window line search query

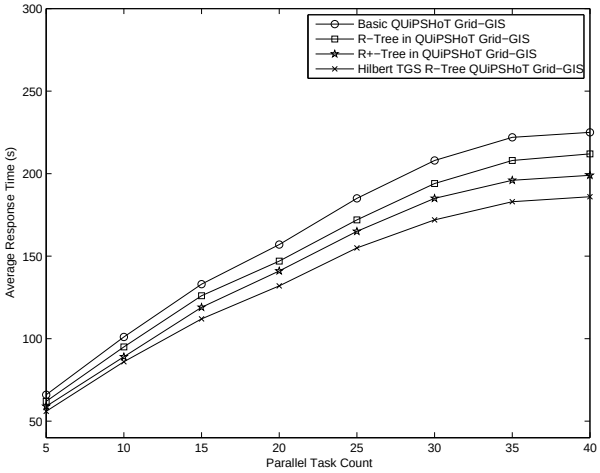

Figure 5.43: Efficiency comparison of Basic QUiPSHoT Grid-GIS and, R-Tree, R+-Tree and Hilbert TGS R-Tree in QUiPSHoT Grid-GIS for window range search query

The query execution time decreases with increasing the bucket size up to 64 KB for the proposed spatial index over the five real datasets, as shown in Figures 5.44 and 5.45, for the line search query and the range search query, respectively. The reason for the pattern is similar to the reason given for building the spatial index i.e. computations are reduced. But the behavior is limited till the size of data packets transfer in HDFS becomes 64 KB. When the tree node size increases further and becomes larger than 64 KB, the number of network transfers increases. For tree node sizes more than 128 KB, 256 KB and 512 KB, the number of network transfers continue to grow. So, the node size calibration is important and setting it below 64 KB produces the optimized query response performance over the cluster.

5.3.3 Scalability: The Effect of Cluster Size on the Search Queries Execution Time

A set of 150 experiments has been conducted, three experiments for different sizes of the cluster varying from 1 to 10 computing nodes and for each county dataset, for a tree-node/bucket size of 64 KB. The mean values are used to draw the graphs presented in the Figure 5.46 and Figure 5.47 for the line search query and the range search query, respectively. The scalability test shows a similar pattern that was observed for the Parallel H-bucket PMR Quadtree spatial index.

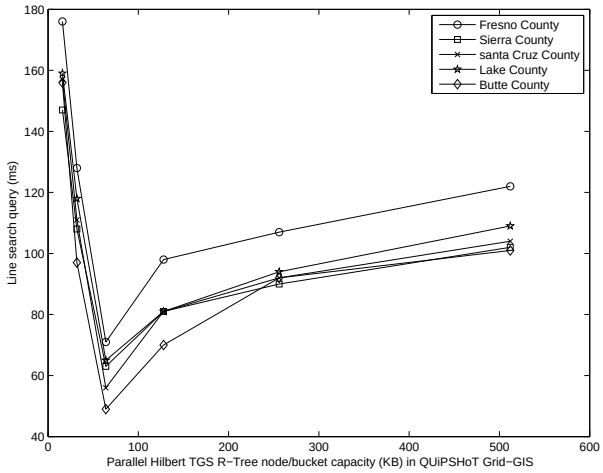

Figure 5.44: The window query for line search on the Parallel Hilbert TGS R-Tree on QUiPSHoT Grid-GIS, for varying node/bucket capacity

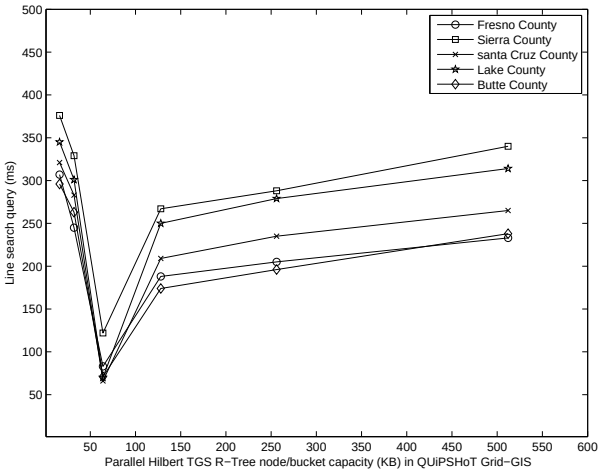

Figure 5.45: The window query for range search on the Parallel Hilbert TGS R-Tree on QUiPSHoT Grid-GIS, for varying node/bucket capacity

A comparison of the parallel Hilbert TGS R-Tree with the Parallel Priority R-Tree in the QUiPSHoT Grid-GIS is presented in Section 5.4.

5.3.4 Window Query Efficiency in the Parallel Hilbert TGS R-Tree for Synthetic Dataset

The efficiency of window search queries in terms of execution time has been measured for some synthetic dataset. The parallel Hilbert TGS R-Tree in the QUiPSHoT Grid-GIS has been considered in checking the query efficiency. The synthetic dataset has been generated in language "R" for (a) uniformly distributed data rectangles with varying side lengths, (b) uniformly distributed data rectangles with variable aspect ratio, and (c) non-uniformly distributed data rectangles with skewed data.

A set of ten experiments has been conducted for each type of synthetic dataset for the spatial queries, and average values have been used to plot the graph for studying the index. The experiment for the synthetic dataset with uniformly distributed rectangles and varying side lengths, 1 mm, 5 mm, 10 mm, 17 mm, 27 mm and 36 mm, for a window query of fixed area of input data rectangles is shown is Figure 5.48. The graph shows a good execution time for smaller data rectangles and the performance keeps on decreasing with large data rectangles. It is due to more number of I/Os are required for a dataset of large rectangles. The number of output rectangles produced for each test run is also shown below the size.

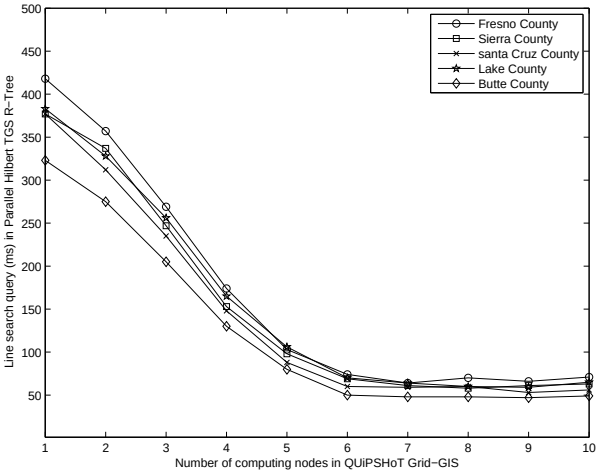

Figure 5.46: The window query for line search on Parallel TGS R-Tree on QUiPSHoT Grid-GIS, for varying number of computing nodes

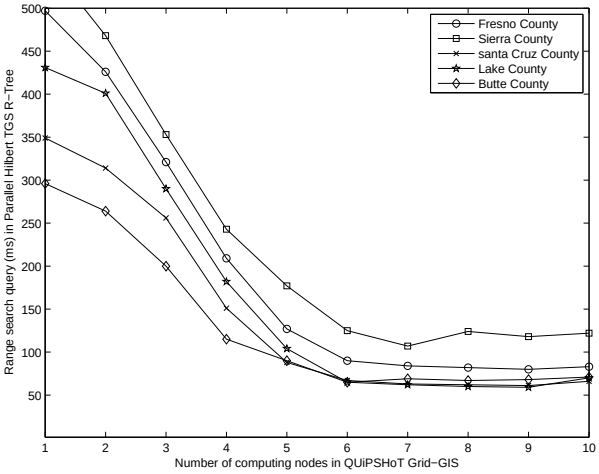

Figure 5.47: The window query for range search on Parallel TGS R-Tree on QUiPSHoT Grid-GIS, for varying number of computing nodes

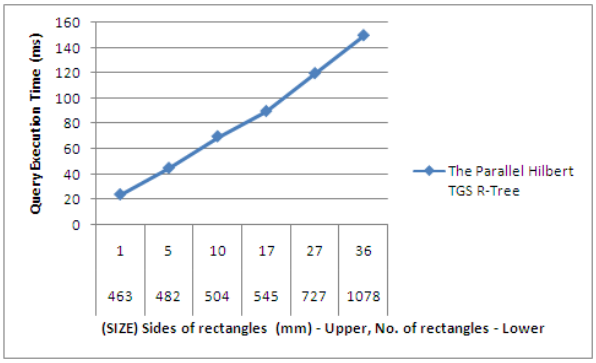

Figure 5.48: The window query of different areas on the Parallel TGS R-Tree on QUiPSHoT Grid-GIS for uniformly distributed data rectangles with varying side lengths

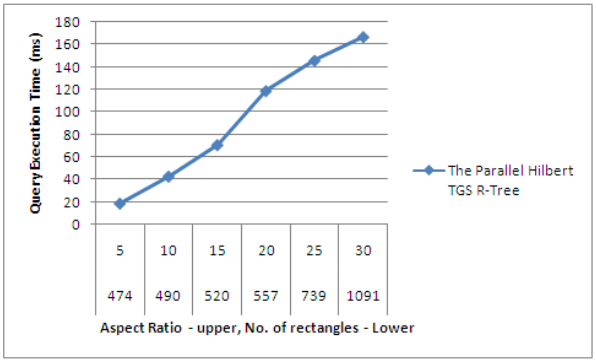

Figure 5.49: The window query of different areas on the Parallel TGS R-Tree on QUiPSHoT Grid-GIS uniformly distributed data rectangles with varying aspect ratio

The experimental results with uniformly distributed data rectangles with varying aspect ratio shows a similar pattern as the aspect ratio rises, as shown in Figure 5.49. The graph presents the execution time of a window query of fixed area on input data rectangles with aspect ratio 5, 10, 15, 20, 25, 30. It is again due to more I/Os are required for large rectangles because of the large aspect ratio. The number of output rectangles produced for each test run is also shown below the aspect ratio.

The performance with the non-uniformly distributed data rectangles with skewed data is presented in Figure 5.50. The graph shows results for six different levels of data skews taken. The higher value represents more skewed data. The graph shows that as the concentration of input data rectangles increases along a particular axis or in a particular region of the bounding rectangle, the performance decreases. It is due to more number of I/Os are required on the overlapping data rectangles because of high data skewness. A similar performance pattern is observed for clustered data rectangles.

A comparison of the window query efficiency of the parallel Hilbert TGS R-Tree and parallel Priority R-Tree is presented in Section 5.4.5.

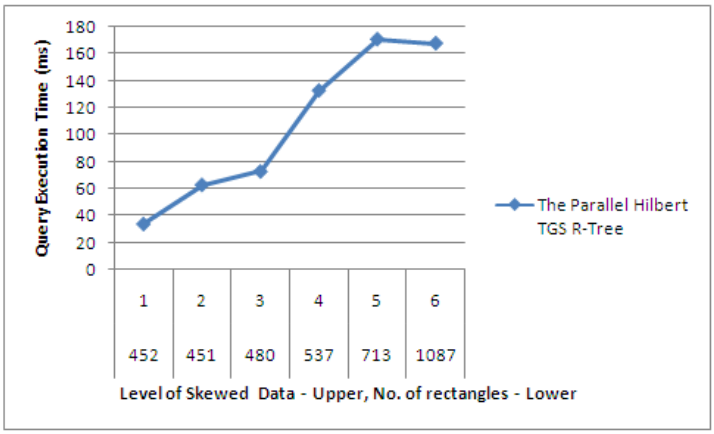

Figure 5.50: The window query of different areas on the Parallel TGS R-Tree on QUiPSHoT Grid-GIS non-uniformly distributed data rectangles with skewed data

5.4 Test and Demonstration of the Parallel Priority R-Tree Index in the QUIPSHoT Grid-GIS

This section presents results and demonstrations of the parallel Priority R-Tree index in the QUiPSHoT Grid-GIS, and a comparison with the parallel Hilbert TGS R-Tree index is done for parameters bulk-loading cost of the proposed index, execution efficiency and

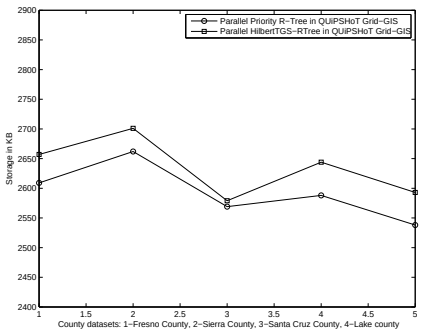

Figure 5.51: Storage Comparison of the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on the QUiPSHoT Grid-GIS, for five county spatial dataset

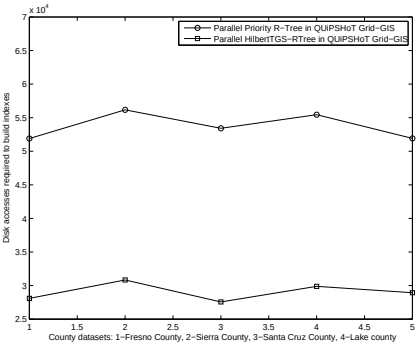

Figure 5.52: Number of disk accesses comparison of the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on the QUiPSHoT Grid-GiS for window query, for five county spatial dataset

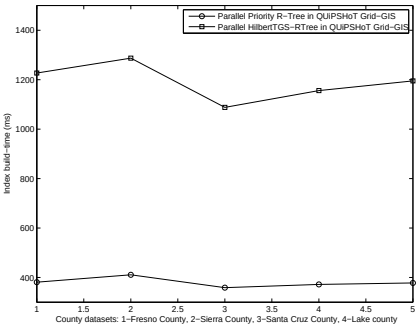

Figure 5.53: The Comparison of the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree build time on the QUiPSHoT Grid-GIS

scalability. The availability parameter for the entire QUiPSHoT Grid-GIS framework is also discussed here.

5.4.1 Bulk-loading Cost of the Index

The costs of building the index with respect to the parameter storage, number of disk accesses and execution time has been analyzed on a cluster of ten computing nodes. A set of 30 experiments has been conducted, three experiments for each index (the Parallel Hilbert TGS R-Tree described in previous Section and Parallel Priority R-Tree) and for each county dataset (five counties), for analyzing the results.

The Figure 5.51 shows the storage requirement and the Figure 5.52 shows the number of disk accesses for queries for the two spatial indexes in the QUiPSHoT Grid-GIS. The two indexes show almost similar pattern for the given dataset. However, the former takes more storage and number of disk accesses, because the algorithms makes a huge number of binary partitions. Similarly, due to the same reason, the index building time of the parallel Hilbert TGS R-Tree is significantly more than the parallel Priority R-Tree.

5.4.2 Execution Efficiency w.r.t. Node Capacity

The execution efficiency of the proposed parallel index parallel Priority R-Tree is tested and demonstrated in the proposed integrated framework. The index building time or

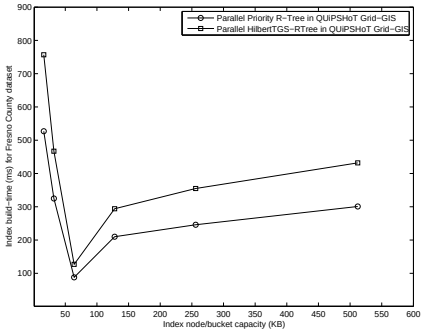

Figure 5.54: Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Fresno County dataset

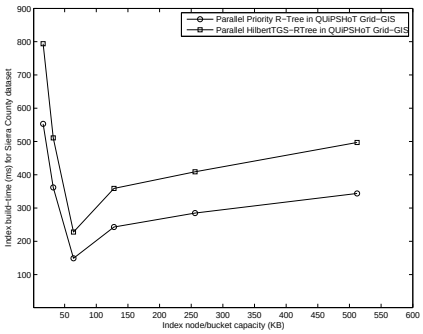

Figure 5.55: Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Sierra County

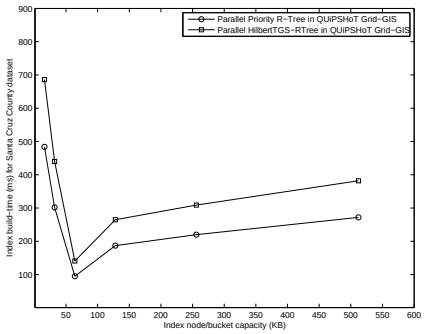

Figure 5.56: Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Santa Cruz County

bulk-loading time of spatial data in the proposed spatial index and spatial queries, line search and range search queries, are used to check the efficiency of execution.

5.4.2.1 Execution Efficiency of Index Building: The Effect of Tree Node/Bucket Size on the Index Building Time

A set of 180 experiments has been conducted, three experiments for each node capacity (16KB, 32 KB, 64 KB, 128 KB, 256 KB and 512 KB) and for each index, for each of the five counties. In a quite similar pattern to the index building seen in the Parallel H-bucket PMR Quadtree and the Parallel Hilbert TGS R-Tree, the index building time decreases with increasing the bucket size up to 64 KB. However, the results in the Figures 5.54, 5.55, 5.56, 5.57 and 5.58 show that the index building time decreases sharply for both the indexes for a cluster size of 10, when the node size increases gradually up to a limit. It is due to the reduced computation requirement as the bounding box area increases that can accommodate more data rectangles with an increase in node capacity. However, this trend does not continue for a long time, it persists till the size of data packets transfer in HDFS becomes 64 KB. When the tree node size increases further and becomes larger than 64 KB, the number of network transfers increases. For tree node sizes more than 128 KB, 256 KB and 512 KB, the number of network transfers continue to grow.

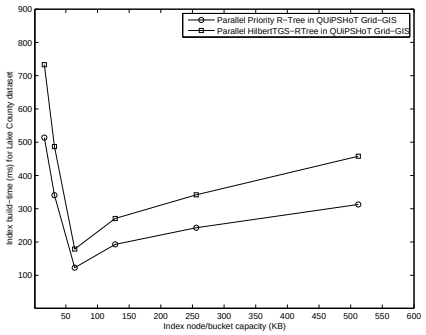

Figure 5.57: Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Lake County

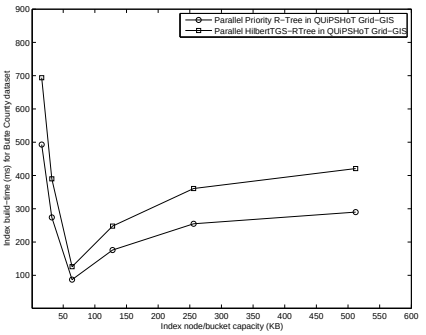

Figure 5.58: Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Butte County

5.4.2.2 Execution Efficiency of Spatial Queries (Line and Range Search Queries)

A set of 180 experiments has been conducted, three experiments for each node capacity (16KB, 32 KB, 64 KB, 128 KB, 256 KB and 512 KB) and for each index, for each of the five counties, for a cluster size of 10 nodes. The spatial queries, line search and range search, are executed for the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree index in the QUipSHoT Grid-GIS framework. The comparison of the query execution efficiency for different dataset show a similar pattern, as has been seen for building the spatial index in Section 5.2.2.1. The query execution time decreases with increasing the bucket size up to 64 KB for both the indexes over the five real datasets, as shown in Figures 5.59, 5.60, 5.61, 5.62 and 5.63, and 5.64, 5.65, 5.66, 5.67 and 5.68 for the line search query and for the range search query, respectively. The reason for the pattern is same. It is due to the reduced computation required with an increased node size in the algorithm. However, for node size greater than 64 KB, the number of network transfers overtakes the reduced computation due to increased node size.

5.4.3 Scalability: The Effect of Cluster Size on the Search Queries Execution Time

A set of 450 experiments has been conducted, three experiments for different sizes of the cluster varying from 1 to 10 computing nodes and for each county dataset, for a tree-node/bucket size of 64 KB. The mean values are used to draw the graphs presented in

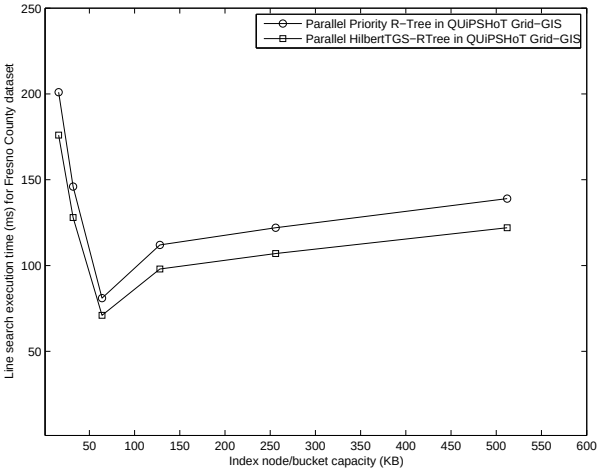

Figure 5.59: Line search execution time (ms) with Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/ bucket capacity (KB) for Fresno County dataset

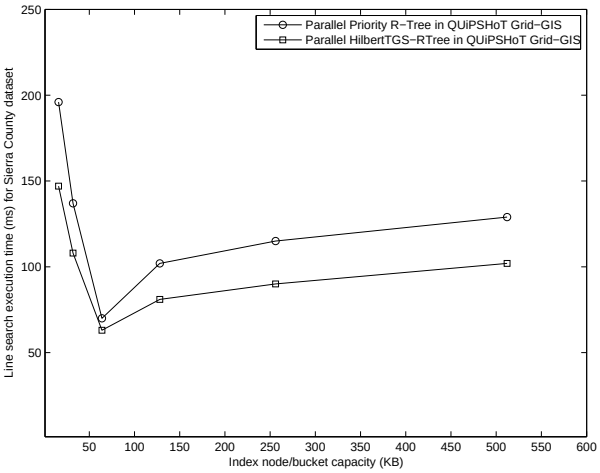

Figure 5.60: Line search execution time (ms) with Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/ bucket capacity (KB) for Sierra County

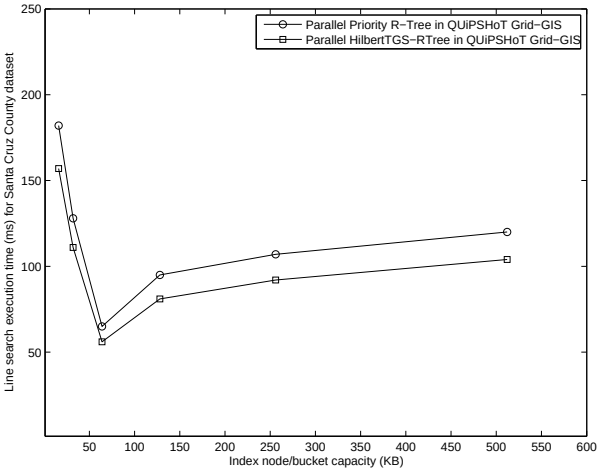

Figure 5.61: Line search execution time (ms) with Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/ bucket capacity (KB) for Santa Cruz County

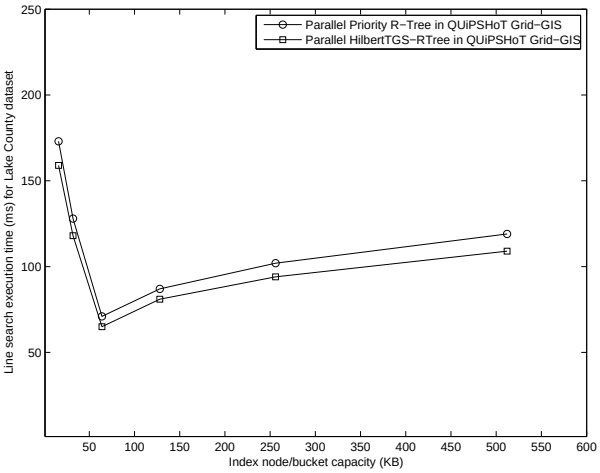

Figure 5.62: Line search execution time (ms) with Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/ bucket capacity (KB) for Lake County

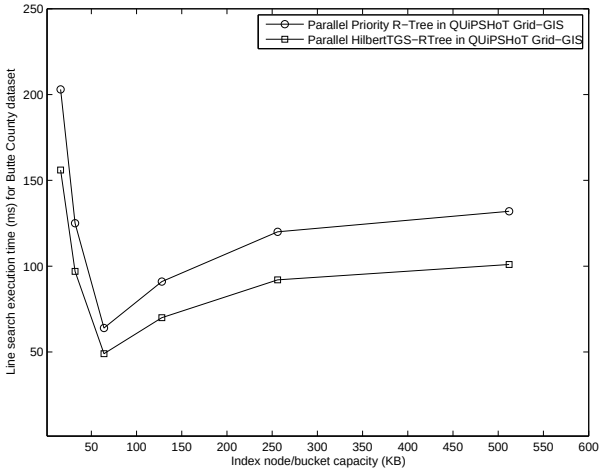

Figure 5.63: Line search execution time (ms) with Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/ bucket capacity (KB) for Butte County

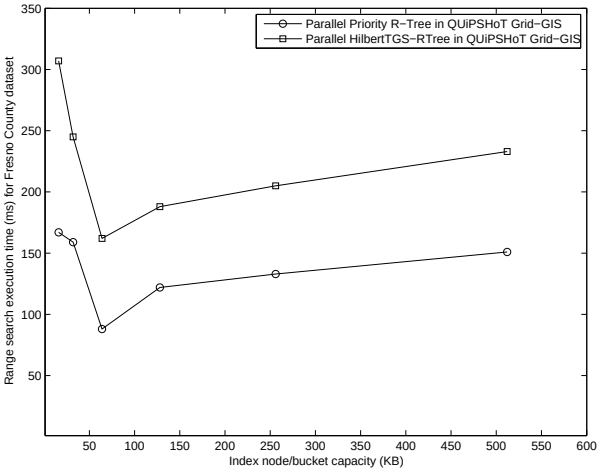

Figure 5.64: Range search execution time (ms) through Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Fresno County dataset

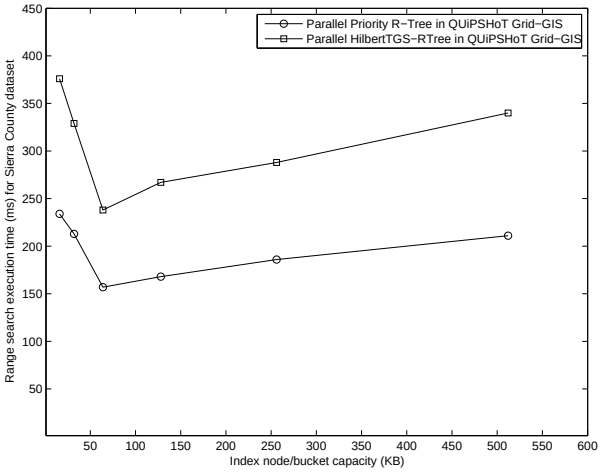

Figure 5.65: Range search execution time (ms) through Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Sierra County

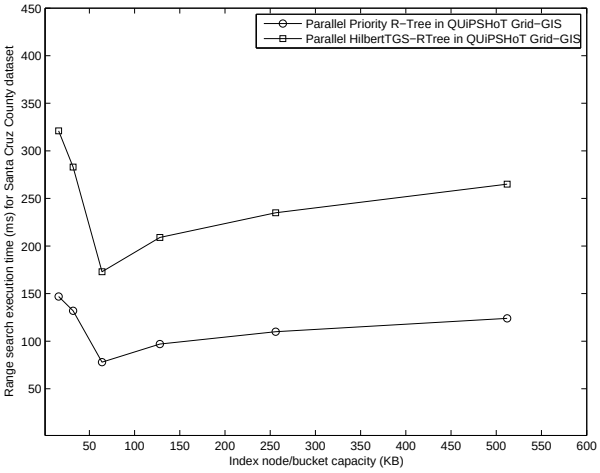

Figure 5.66: Range search execution time (ms) through Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Santa Cruz County

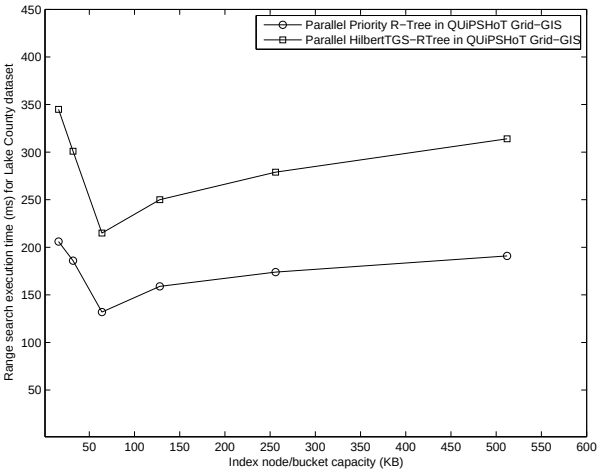

Figure 5.67: Range search execution time (ms) through Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Lake County

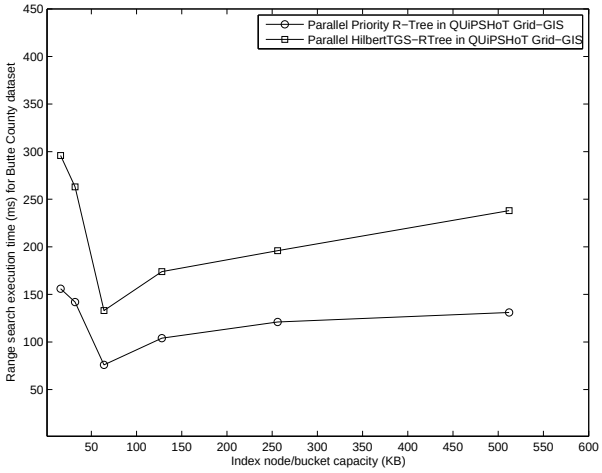

Figure 5.68: Range search execution time (ms) through Parallel Hilbert TGS R-Tree and Priority R-Tree build time (ms) on QUiPSHoT Grid-GIS with varying node/bucket capacity (KB) for Butte County

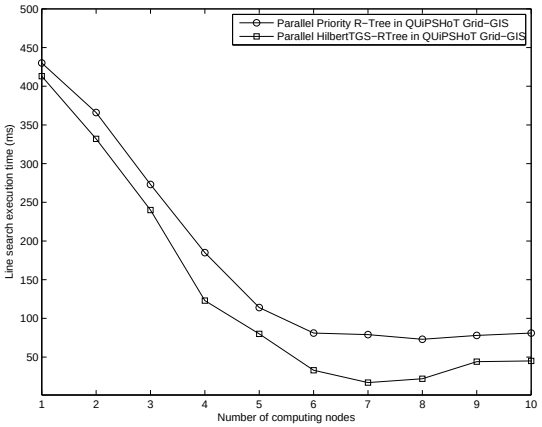

Figure 5.69: A window query for line search on the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on QUiPSHoT Grid-GIS for varying number of computing nodes

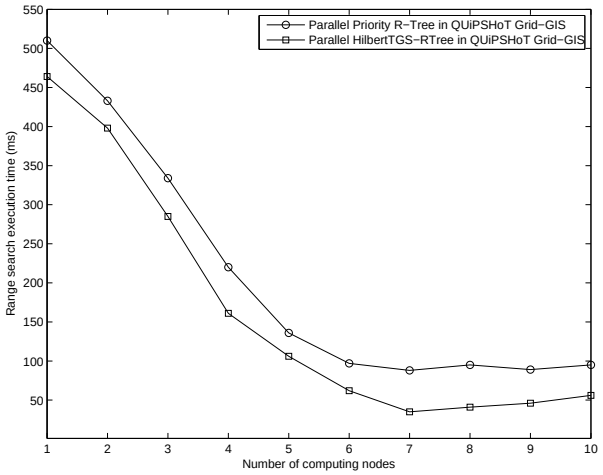

Figure 5.70: A window query for range search on the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on QUiPSHoT Grid-GIS for varying number of computing nodes

the Figures 5.69 and Figures 5.70 for the line search query and the range search query, respectively. The experiment shows a good scalability with the addition of computing nodes. The computation time decreases due to the distributed computations in the QUiPSHoT Grid-GIS. However, in the current experimentation, the computation time becomes stable for ten computing nodes. It is due to the size of the dataset. If the size of the dataset is taken quite big then the computation time can drop for number of nodes and becomes stable after that. A stable point comes when the data transfer overhead overtakes the computation due to parallelism.

The experimental results show that the query execution time for the Parallel Hilbert TGS R-Tree is 1-1.5 less time as compared to the Parallel Priority R-Tree. However, the build time for the former was more than the latter, as was presented in the previous Section.

5.4.4 Availability: Fault Tolerance

The availability or fault tolerance in the proposed QUiPSHoT Grid-GIS framework is for the three components. First, the Spatial Grid Control (SGC) node of the Grid, second, the master node of the dynamically constituted cluster of the Hadoop Enabled Spatial Grid (HESG) nodes, as described in the Section 3, and third, the HESG nodes holding the distributed spatial data.

5.4.4.1 Availability of the Manager Nodes

There are two main nodes in the QUiPSHoT Grid-GIS framework, the SGC node and the master HESG node. These two nodes are the controlling part of the Grid and the dynamically constituted cluster, respectively. The Grid is characterized as having the capability to handle the bag-of-tasks with few I/Os, however the MapReduce is characterized for handling a large volume of inputs and intermediate data. Though, the shuffling of the huge volume of intermediate data is possible in the Grid, however it is a difficult operation. The integrated framework utilizes the built-in capability of the MapReduce to redistribute the intermediate data through a well-known "shuffle" phase in the MapReduce. Besides it, the MapReduce - Hadoop internally handles the availability of the master node (namenode) of the cluster and the other datanodes of the cluster, through a replication process.

To test the availability of the Manager nodes, the nodes are shut down when the work has been going-on in the QUiPSHoT Grid-GIS. The purpose of this abnormal shutdown has been to check that whether the Grid-GIS can tolerate it and automatically switching from the manager node to the back-up manager node takes place or not. The tests show that the Grid-GIS system survives without failure. The whole process has been observed during the run-time when the manager node goes-off, the normal execution hangs for a little while. However, it restarts in few seconds and behaves normally after that, as if nothing would have happened.

5.4.4.2 Availability of the Distributed Spatial Data for Spatial Queries

The task submitted by the user through any Grid Node comes to the Manager Grid Node. The Manager Grid Node identifies the nodes on which resources lie, so that these can be used for accomplishing the submitted task. The HESG Nodes (Datanodes) of the dynamically formed cluster communicates with the master node (Namenode) of the cluster. These datanodes hold the distributed spatial data and works in the Map and Reduce fashion to complete sub-tasks assigned to these by the master node. The replica management feature of the Hadoop automatically creates replicas of the data chunks on the namenodes. These replicas are now distributed on the other datanodes of the cluster. At the time of failure of one datanode, the other datanode holding the data chunk comes into effect automatically. To test the availability of the HESG Nodes, one of the datanode was chosen randomly and shut down while the Grid-GIS operations were going-on. The effect was noticed on the master node; the execution reported failures of the tasks concerned to the failed or shut-down node. The Hadoop cluster in the Grid-

GIS automatically managed the failure and searched the replicated resources on the other datanodes and completed the tasks.

The replica feature in Hadoop takes care of the fault tolerance by replicating data blocks among cluster nodes and improves the data availability. We set the number of replicas (x) for a data block through the configuration file. It was found that when we set x=1 and run the task, then sometimes the task completes successfully, but for a few times the task failed to complete. When we set x=2 then the task runs successfully all the times baring a negligible number of times. When we set x=3, then the task takes abnormally long time. And when we further increase x=4, the task does not complete and the machine/cluster is hanged. The reason for this kind of behavior is due to the number of replicas of data blocks that influences the execution. When insufficient numbers of replicas are in the cluster, then sub-tasks fail as data blocks are not found due to locality problem or there are more failures during data transfer. In another case, when the numbers of data block replicas are larger in number than the size of HDFS data becomes larger and more data transfer takes place over the network that causes consumption of more network resources and puts load over the network and consequently slow down or sometimes even hang the whole system.

5.4.5 Window Query Efficiency on the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on Synthetic Data

The response of window queries are observed on the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree in QUiPSHoT Grid-GIS framework for synthetic data described in the Section 4.3.2. The experiments are conducted over three different type of synthetic datasets. The different dataset help to investigate the behavior of different indexes.

Firstly, the uniformly distributed data rectangles of varying length and breadth of their sides. Secondly, the uniformly distributed rectangles in a fixed aspect ratio, however the rectangles are uniformly distributed around their centers. The aspect of a rectangle is the fixed ratio of its length to breadth. The longer side may be taken horizontally or vertically with equal probability. Thirdly, the data rectangles are inclined or concentrated in a particular area or along an axis. This kind of dataset is termed as skewed data. In a multidimensional dataset, data rectangles may be uniformly distributed along one dimension; however, these may be skewed along a different dimension.

In one category of experiments, observations have been recorded in the specified two

indexes in the framework, for window queries on input dataset, with length of sides 1 mm, 5 mm, 10 mm, 17 mm, 27 mm and 36 mm, respectively. For each specified side length of rectangles, a set of ten experiments has been conducted for each index per side length size and the average values are used to plot the behavior on the graph. The average query execution time with the number of output rectangles is presented in Figure 5.71.

The graph in the Figure 5.71 describes that for small rectangles, having a smaller side length and breadth, the window search query performance is better for the parallel Hilbert TGS R-Tree than the parallel Priority R-Tree. The difference in performance narrows as rectangles become bigger in size, and for large rectangles the latter descriptor index behaves much better than the former index. A similar kind of pattern is observed for rectangular input data with aspect ratio, as shown in the Figure 5.72. A set of ten experiments was conducted for each index per aspect ratio and the average value was used to plot the behavior on the graph. When the aspect ratio is smaller, the ratio of length to breadth of the input rectangles, the performance of the parallel Hilbert TGS R-Tree is much better than the parallel Priority R-Tree, however, when the aspect ratio increases, the parallel Priority R-Tree's performance becomes much better than the parallel Hilbert TGS R-Tree. Another similar pattern is observed for these two indexes, when the skewed data are considered, as shown in the Figure 5.73. A set of ten experiments was conducted for each index per skewed dataset and the average value was used to plot the behavior on the graph. For a low skewed data, having a density almost uniform across the main bounding rectangle, the performance of the parallel Hilbert R-Tree index is much better than the parallel Priority R-Tree index. However, as the skewness of data increases, i.e. the input data rectangles are more concentrated around one particular area in the bounding rectangle, then the performance of parallel Priority Tree is observed much better than the parallel Hilbert TGS R-Tree. A similar performance pattern is observed for the clustered data rectangles.

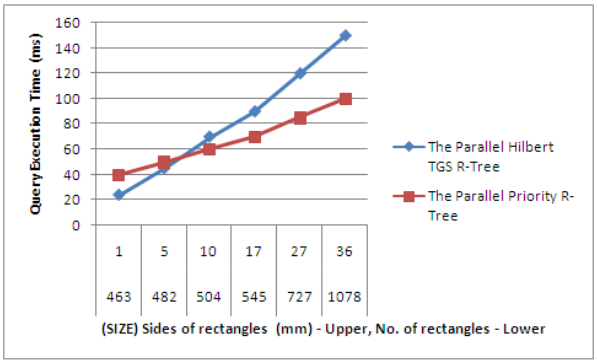

Figure 5.71: The window search query on the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on QUiPSHoT Grid-GIS for synthetic data with varying sizes of the data rectangles

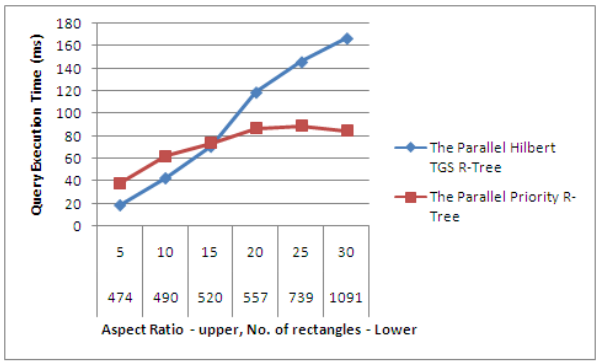

Figure 5.72: The window search query on the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on QUiPSHoT Grid-GIS for synthetic data with varying aspect ratio of data rectangles

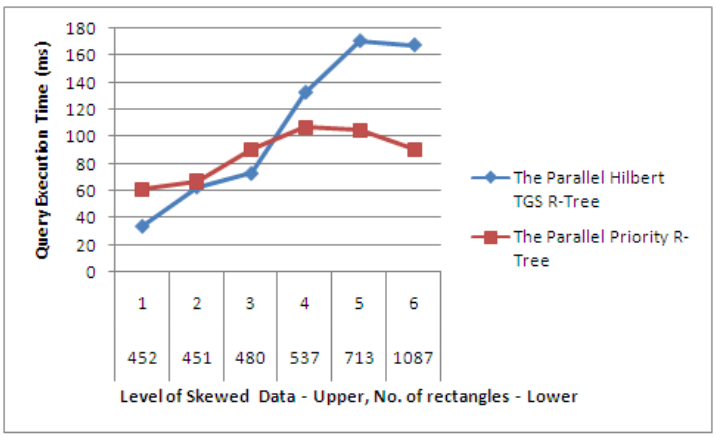

Figure 5.73: The window search query on the Parallel Hilbert TGS R-Tree and the Parallel Priority R-Tree on QUiPSHoT Grid-GIS for varying amount of skewed data rectangles

Chapter 6

Conclusions and Future Scope

This chapter presents the concluding part of the thesis and also proposes some suggestions towards which the present work can be further extended. The Section 6.1 brings out the overall conclusions of the research work carried out and the Section 6.2 presents the future scope.

6.1 Conclusions

An integrated QUiPSHoT Grid-GIS framework is developed that consists of the Grid and the MapReduce technologies, applied in the field of spatial data. The grid provides huge computational power and storage, and the MapReduce provides huge data analytic capability to the proposed architecture and framework. The three proposed parallel spatial indexes, of the existing traditional serial spatial indexes, provide efficient organization of spatial data, and the efficiency of spatial data retrieval for spatial queries. The work carried out in the thesis is as follows:

6.1.1 To study and explore Grid, GIS, and related work done so far in Grid-GIS

(i) The concepts of grid computing, such as the grid architecture, the types of grids and their characteristics, have been studied and analyzed. The Globus toolkit 4.0 has been studied, analyzed and practically used for setting-up a grid of computing nodes.

(ii) The concepts of Geographic Information System (GIS), such as components of the GIS, data for the GIS, distributed GIS, Web-GIS and Grid-GIS, have been studied and analyzed. The spatial databases, such as MySQL and Postgre SQL have been studied, analyzed and used for managing and querying the spatial data objects. The ArcGIS 10.2 GIS has been studied, analyzed and used for various GIS operations and queries.

(iii) The work done so far in Grid-GIS has been studied and analyzed through a detailed literature survey presented in (Chapter 2).

6.1.2 To analyze existing Grid-GIS design architectures and propose an efficient Grid-GIS architecture and framework

(i) The effort has been put to study and analyze the existing Grid-GIS design and architectures. It includes the OGSI, WSDL and XML Based Grid GIS, the OGSA and WSRF Based Grid GIS and the Mobile Agent Technology Based Grid GIS. The mobile agent technology is used for parallel processing.

(ii) For the efficient discovery, retrieval, etc. of spatial data, technologies, such as the WSRF, the Mobile Agent and the MapReduce technologies have been studied and analyzed. The MapReduce has been considered for integrating with the grid for parallel processing of huge volumes of spatial data and analysis. The simplicity, ease of use and the ability to easily handle large datasets, the ease of programming by using the Map and Reduce functions, the fault tolerant nature of the MapReduce model and the property of abstracting the parallelization from the user are the reasons that have motivated researchers to use the model. It provides a good performance through scaling out to computing clusters. The MapReduce implementation Hadoop has been studied, analyzed and practically used for setting-up a Hadoop cluster.

(iii) The index buffer component has been addressed for a better management of the spatial data and fast query retrieval. A literature survey of the traditional and the MapReduce based approaches for spatial query processing (management and retrieval) has been carried out. The traditional spatial indexes have been surveyed for a comparative analysis of the existing sequential processing algorithms and to identify the algorithms that have the potential for parallelization. The MapReduce based spatial query processing approaches have been surveyed for finding the scope for proposing the parallel indexing for the identified algorithms.

(iv) The availability of the Domain Manager components is addressed with the MapReduce implementation, Hadoop. It is realized through creating the data replicas and the secondary Master node in the cluster.

(v) The efficient Grid-GIS framework QUiPSHoT Grid-GIS is proposed. It integrates the proposed spatial indexing techniques in the MapReduce and the Grid-GIS.

6.1.3 To design and implement the proposed Grid-GIS architecture and framework

(i) A QUiPSHoT Grid-GIS architecture is proposed.

(ii) A QUiPSHoT Grid-GIS framework is proposed.

(iii) The Parallel H-bucket PMR Quadtree spatial index in the MapReduce is designed and implemented in the proposed framework.

(iv) The Parallel Hilbert TGS R-Tree in the MapReduce is designed and implemented in the proposed framework.

(v) The Parallel Priority R-Tree in the MapReduce is designed and implemented in the proposed framework.

(vi) The availability of the Domain Manager is taken care of in the proposed framework.

(vii) The design and implementation of the integrated spatially indexed MapReduce and Grid-GIS is carried out towards the proposed QUiPSHoT Grid-GIS framework.

6.1.4 To test and demonstrate the Grid-GIS Framework

The proposed QUiPSHoT Grid-GIS framework has been tested and demonstrated on the real life and the synthetic dataset for the following parameters.

(i) Bulk-loading cost of the index : The amount of storage used for organizing the spatial data. Better the storage management, less will be the number of disk accesses, and hence better will be the query execution time.

(ii) Execution Efficiency: It is the amount of time taken for executing a query.

(iii) Scalability: The linear increase in efficiency with a linear increase in the resources for handling a spatial query.

(iv) Availability: It is the measure of fault tolerance of the system. it signifies, how the system maintains its resources available at all times irrespective of any failure.

An efficient Grid-GIS framework called QUiPSHoT Grid-GIS has been developed that demonstrates the application of the framework in the domain of spatial data.

6.2 Future Scope

Research is an iterative and continuous procedure. The work presented in the thesis focuses on developing an efficient Grid-GIS framework for spatial data. There are several directions in which this work can be extended.

The survey of the existing Grid-GIS design and architectures revealed that various technologies had been integrated on the Grid in the past, for realizing the design and architecture of an efficient Grid-GIS. The OGSI, WSDL and XML Based Grid GIS, the OGSA and WSRF Based Grid GIS and the Mobile Agent Technology Based Grid GIS, were such technologies that effectively addresses the issue of spatial data management and retrieval on the Grid. This thesis work has integrated the MapReduce technology for spatial data on the Grid. The mobile agent and the MapReduce based work, basically provides the parallelization of the spatial data management and queries. Various other parallel processing technologies exist, such as CUDA programming, Pig, Hive, etc. that can be replaced with the MapReduce-Spatial Hadoop in the proposed framework. The Cassandra, Pig, Hive actually extends the Hadoop for making it more user friendly and business accessible. These technologies can be replaced with the SpatialHadoop integrated in the proposed architecture and framework for making the proposed framework more user friendly.

The MapReduce-SpatialHadoop is a significant component of the proposed QUiPSHoT Grid-GIS architecture. However, the Hadoop is good for organized data processing and batch processing jobs. Recently, the demand has risen for real time streaming analysis and in-memory analysis, which is not provided by the Hadoop. The technologies, such as Storm and Kafka, can be used for such purposes.

The results obtained through the experimental work are highly encouraging. The integration of the grid computing and the MapReduce provides a new benchmark in terms of efficiency of processing huge volumes of spatial data. The spatial data is considered as big data due to its inherent characteristics. There is a good scope for designing and implementing efficient parallel spatial indexing on the integrated architecture from the existing traditional spatial indexes that have strong potential for parallelization. Through the surveyed traditional spatial indexes, various spatial indexes are marked that are having a good potential for parallelism. Besides, the Priority R-Tree [17], the TGS R-Tree [18] and the bucket PMR Quadtree [104] spatial indexes explored in this thesis, the parallel versions of the cR-Tree [9], the X-Tree [8], the WeR-Tree [7], the RR*-Tree [6] can be developed for an efficient Grid-GIS architecture and framework.

The Hadoop is characterized with a high communication overhead. It was also observed during the experimental runs of the proposed Grid-GIS for spatial queries. However, not an exclusive analysis was done for it. The parameter can be considered as a future scope of the work. Either the middleware of the Hadoop can be improved or other technologies that provide low communication overhead can be used.

In the thesis, the spatial data deletion or modification has not been considered. The analysis has been done according to the data inserted in the write-once and read-many design. So, this aspect can be considered for future extension and improvements.

In the thesis, the effect of index node size of the proposed and implemented spatial index has been considered and reported. An increase in the size of the data structure node reduces the height of the tree and consequently the disk access time. However at the same time the search time inside the node is increased which is sequential. This can be improved using secondary structures within the node and used in range trees, interval trees etc..

In the thesis, plots for the performance measures for PMR Quad-tree, TGS R-Tree and Priority R-Tree for the parameters [Preprocessing(Build Time) + Query Execution Time * Number of Queries] vs Bucket Size may reflect their actual performance for uniformly distributed and clustered data sets. So, this aspect can be considered for future extension and improvements.

The framework can be extended for other complex spatial queries and dataset for further strengthening and validating the obtained results.

References

[1] Andrew Swift Grimshaw, Marty Allen Humphrey, and Anand Natrajan. A philosophical and technical comparison of Legion and Globus. *IBM J. RES. & DEV.*, 48, 2004.

[2] R-tree, http://en.wikipedia.org/wiki/R-tree .

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD 90 Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, volume 19(2), pages 322–331, 1990.

[4] Autonin Guttman. R-trees: a dynamic index structure for spatial searching . In *Proceeding SIGMOD 84 Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, volume 14(2), pages 47–57, 1984.

[5] Christos Faloutsos Timos Sellis, Nick Roussopoulos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings VLDB '87 Proceedings of the 13th International Conference on Very Large Data Bases*, volume 19(2), pages 507–518, 1987.

[6] Norbert Beckmann and Bernhard Seegar. A Revised R*-tree in Comparison with Related Index Structures. In *Proceeding, SIGMOD 09 Proceedings of the 2009 ACM SIGMOD international conference on Management of data*, pages 799–812, 2009.

[7] Panayiotis Bozanis and Panayiotis Foteinos. WeR-trees. *Data & Knowledge Engineering*, 63(2):397–413, 2007.

[8] Daniel Keim, Benjamin Bustos, Stefan Berchtold, and Hans-Peter Kreigel. X-tree. *Encycopedia of GIS*, pages 543–547, 2008.

[9] Sotiris Brakatsoulas, Dieter Pfoser, and Yannis Theodoridis. Revisiting R-tree Construction Principles. In *Proceedings of the 6th East European Conference on Advances in Databases and Information System, Springer Verlag*, pages 149–162, 2002.

[10] D. M. Gavrila. R-Tree Index Optimization. In *In 6th International Symposium on Spatial Data Handling*, pages 771–791, 1994.

[11] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. In *Proceedings of the second international conference on Information and knowledge management, New York, NY, USA, ACM*, pages 490–499, 1993.

[12] Ibrahim Kamel and Christos Faloutsos. Hilbert R-tree: An Improved R-tree Using Fractals. In *Proceedings of the 20th International Conference on Very Large Data*

Bases, ACM DL, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, pages 500–509, 1994.

[13] Jonathan K Lawder and Peter King. Using Space-filling curves for multi-dimensional indexing. *Advances in databases, Lecture Notes in Computer Science*, 1832:20–35, 2008.

[14] Scott Thomas Leutenegger, Jeffrey Michael Edgington, and Mario Alberto Lopez. STR: A simple and efficient algorithm for r-tree packing. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE), Birmingham, UK, IEEE Computer Society*, pages 497–506, 1997.

[15] Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed R-trees. In *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, volume 14(4), pages 17–31, 1985.

[16] Xiaogian Wu and Chuanqin Zang. A New Spatial Index Structure for GIS Data. In *Third International Conference on Multimedia and Ubiquitous Engineering*, pages 471–476, 2009.

[17] Lars Arge, Mark de Berg, Herman J. Haverkort, and Ke Yi. The Priority R-Tree: A Practically Efficient and Worst-Case Optimal R-Tree. In *ACM Transactions on Algorithms*, volume 4(1), pages 270–273, 2008.

[18] Yvan J Garcia R, Mario Alberto Lopez, and Scott Thomas Leutenegger. A greedy algorithm for bulk loading r-trees. In *In Proceedings of the 6th ACM International symposium on advances in geographic information systems, ACM Press*, pages 163–164, 1998.

[19] Sangyong Hwang, Keunjoo Kwon, Sang K. Cha, and Byung S. Lee. Performance Evaluation of Main-Memory R-tree Variants. *The VLDB Journal*, 2750:10–27, 2003.

[20] Kihong Kim, Sang K. Cha, and Keunjoo Kwon. Optimizing Multidimensional Index Trees for Main Memory Access. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, volume 30(2), pages 139–150, 2001.

[21] Zhiming Gui, Zhou Huamg, Xuetong Xie, Dongxuan Tian, Lixia Liu, and Xiaoning Wang. Building Grid GIS Service with WSRF. In *Second International Conference on advanced Geographic Information System, Applicationa, and Servives, IEEE Computer Society*, 2010.

[22] Feng Jun, Tang Zhixian, Wei Mian, , and Xu Liming. HQ-Tree: A Distributed Spatial Index Based on Hadoop. In *China communications*, pages 1009–1020, 2014.

[23] SpatialHadoop, http://spatialhadoop.cs.umn.edu.

[24] Ahmed Eldawy and Mohamed F. Mokbel. A Demonstration of SpatialHadoop:An Efficient MapReduce Framework for Spatial Data. In *Proceedings of the VLDB*

Endowment, volume 6(12), pages 1230–1233, 2012.

[25] Ahmed Eldawy and Mohamed F. Mokbel. The ecosystem of spatialhadoop. In *SIGSPATIAL*, volume 6(3), pages 3–10, 2014.

[26] Chi Zhang, Feifei Li, and Jeffery Jestes. Efficient Parallel kNN Joins for Large Data in MapReduce. In *Proceeding EDBT 12 Proceedings of the 15th International Conference on Extending Database Technology*, pages 38–49, 2012.

[27] Ariel Cary, Yaacov Yesha, Malek Adjouadi, and Naphtali Rishe. Leveraging Cloud Computing in Geodatabase Management. In *Proceeding GRC 10 Proceedings of the 2010 IEEE International Conference on Granular Computing*, pages 73–78, 2010.

[28] Ariel Cary, Zhengguo Sun, Vagelis Hristidis, and Naphtali Rishe. Experiences on Processing Spatial Data with MapReduce, Scientific and Statistical Database Management. *Scientific and Statistical Database Management*, 5566:302–319, 2009.

[29] Haojun Liao, Jizhong Han, and Jinyun Fang. Multi-dimensional Index on Hadoop Distributed File System. In *Fifth IEEE International Conference on Networking, Architecture, and Storage, IEEE Computer Society*, pages 240–249, 2010.

[30] Yi Liu and Huizandhong Chen Ning Jing Luo Chen. Parallel bulk-loading of spatial data with MapReduce: An R-tree case. *Wuhan University Journal of Natural Sciences*, 16(6):513–519, 2009.

[31] Shoji Nishimura, Sudipto Das, Divyakant Agarwal, , and Amr El Abbadi. MD-Hbase design and implementation of an elastic data infrastructure for cloudbased location services. *Distributed Parallel Databases*, 31:289–319, 2013.

[32] Yonggang Wang and Sheng Weng. Research and implementation on spatial data storage and operation based on Hadoop platform . In *Second IITA International Conference on Geoscience and Remote Sensing (IITA-GRS), IEEE*, volume 2, pages 275–278, 2010.

[33] Li Xun and Zheng Wenfeng. Parallel Spatial Index Algorithm Based on Hilbert Partition. In *In International Conference on Computational and Information Sciences, IEEE Computer Society*, pages 876–879, 2013.

[34] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. In *Proceedings of the VLDB Endowment*, volume 6(11), pages 1009–1020, 2013.

[35] Ablimit Aji and Fusheng Wang. High Performance Spatial Query Processing for Large Scale Scientific Data. In *SIGMOD12 PhD Symposium*, pages 9–14, 2012.

[36] Yunqin Zhong, Jizhong Han, Tieying Zhang, Zhenhua Li, Jinyun Fang, and Guihai Chen. Towards Parallel Spatial Query Processing for Big Spatial Data. In *26th International Parallel and Distributed Processing Symposium Workshops & PhD*

Forum (IPDPSW), IEEE , pages 2085–2094, 2012.

[37] Himanshu Gupta, Bhupesh Chawda, Sumit Negi, Tanweer A. Faruquie, and L.V. Subramanium. Processing multi-way spatial joins on map-reduce. In *Proceedings of the16th International Conference on Extending Database Technology(EDBT)*, pages 113–124, 2013.

[38] Kai Wang, Jizhong Han, Bibo Tu, Jiao Dai, Wei Zhou, and Xuan Song. Accelerating Spatial Data Processing with MapReduce. In *16th International Conference on Parallel and Distributed Systems (ICPADS), IEEE*, pages 229–236, 2010.

[39] Shubin Zhang, Jizhong Han, Zhiyong Liu, Kai Hwang, and Zhiyong Xu. SJMR: Parallelizing Spatial Join with MapReduce on Clusters. In *IEEE International Conference on Cluster Computing and Workshops*, pages 1–8, 2009.

[40] Qiang Ma, Bin Yang, Weining Qian, and Aoying Zhou. Query processing of massive trajectory data based on mapreduce. In *SIGSPATIAL*, pages 9–16, 2009.

[41] Priority R-Tree, http://www.cs.umd.edu/class/spring2005/cmsc828s/slides/prtree.pdf.

[42] Matt Haynos. Perspective on grid: Grid computing next-generation distributed computing from http://www-106.ibm.com/developerworks/library/gr-heritage.

[43] Anshu Chauhan, D.K. Gupta, and Manoj Kumar Sah. Detection of Packet Dropping Nodes in MANET using DSR Routing Protocol. *International Journal of Computer Applications*, 123(7):10–16, 2015.

[44] Manoj Kumar Sah, D.K. Gupta, and Pooja Rani. Energy Efficient Routing Protocol for Wireless Sensor Networks with Multiple Sinks. In *2015 Second International Conference on Advances in Computing and Communication Engineering (ICACCE)*, page doi:10.1109/ICACCE.2015.127, 2015.

[45] I. Foster. Globus Toolkit Version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21:513–520, 2006.

[46] Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal. Globus Grids and Software Toolkits: A Study of Four Middleware Technologies, https://arxiv.org/ftp/cs/papers/0407/0407001.pdf.

[47] Md. Ahsan Arefin et. al. Alchemi Vs Globus: A Performance Comparison. In *4th International Conference on Electrical and Computer Engineering, ICECE 2006*, pages 3–13, 2006, dhaka, Bangladesh.

[48] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco Morgan Kaufmann Publishers, 1999.

[49] Hailong Sun, Wantao Liu, Tianyu Wo, and Chunming Hu. CROWN Node Server An Enhanced Grid Service Container Based on GT4 WSRF Core. In *Fifth International Conference on Grid and Cooperative Computing Workshops*, volume 4, pages 510–517, 2006.

[50] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Kari Czajkowski, and et. al. Modeling Stateful Resources with Web Services version 1.1, https://www.ibm.com/.../library/ws-resource/ws-modelingresources.pdf, 2004.

[51] Eduardo Huedo, Ruben S. Montero, and Ignacio M. Liorente. A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services. *Future Generation Computer Systems*, 23:252–261, 2007.

[52] Karl Czajkowski and et al. The WS-Resource Framework Version 1.0, Tech. Rep. Available from http://www.globus.org/wsrf/specs/ws-wsrf.pdf, 2004.

[53] Ian Foster and et al. Modeling and managing state in distributed systems The role of OGSI and WSRF. In *Proceedings of the IEEE 93 (3)*, pages 604–612, 2005.

[54] Marty Humphrey and et al. State and events for web services A comparison of five WS-Resource framework and WSNotification implementations. In *In Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing, HPDC-14*, pages 3–13, 2005.

[55] OASIS, Web Services Resource Framework (WSRF) V1.2, http://ws.apache.og/wsrf/wsrf.html, Dec. 2004.

[56] Jinpeng Huai, Hailong Sun, Chunming Hu, Yanmin Zhu, Yunhao Liu, and Jianxin Li. ROST Remote and hot service deployment with trustworthiness in CROWN Grid. *Future Generation Computer Systems*, 23:252–261, 2007.

[57] James Frey, Todd Tannenbaum Miron Livny, Ian Foster, and Steven Tuecke. Condor-G: A computation Management Agent for multi-Institutional Grids. *Cluster Computing*, 5:237–246, 2002.

[58] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. 6:37–46, 2002.

[59] The Globus Project Argonne National Laboratory USC Information Sciences Institute: Grid Architecture. Available from:http://www.globus.org.

[60] German Molto, Hernandez Vicente, and Jose M. Alonso. A service-oriented WSRF-based architecture for metascheduling on computational Grids. *Future Generations Computer Systems*, 24(4):317–328, 2008.

[61] David J. Maguire, Michael F. Goodchild, , David W. Rhind, and Paul A. Longley. *Geographical information systems : principles and applications*. Longman Scientific & Technical; New York: Wiley, Harlow, England.

[62] Peter A. Burrough, Rachael A. McDonnell, and Christopher D. Lloyd. *Principles of geographical information systems*. Oxford University Press, Oxford.

[63] Joe Mambretti and Gigi KarmousEdwards. *Grid Networks Enabling Grids with Advanced Communication Technology Franco Travostino*. John Wiley & Sons, Ltd, 2006.

[64] Nicholas R. Chrisman. What does 'GIS' mean?" Transactions in GIS. *Transactions in GIS*, 3:175–186, 2004.

[65] Stan Aronoff. *Geographic Information Systems: A Management Perspective*. WDL Publications, Ottawa, Canada., 1989.

[66] Bruce Ellsworth Davis. *GIS: a visual approach*. Delmar Thomson Learning, Albany., 2001.

[67] Varsha Bhat Kukkala, Jaspal Singh Saini, and S.R.S Iyenger. Secure Multiparty Construction of a Distributed Social Network. In *Proceedings of the 18th International Conference on Distributed Computing and Networking* , volume 12, page doi:10.1145/3007748.3007783, 2017.

[68] Varsha Bhat Kukkala and S.R.S Iyengar. MPC meets SNA: A Privacy Preserving Analysis of Distributed Sensitive Social Networks, doi:arXiv:1705.06929, 2017.

[69] Varsha Bhat Kukkala, Jaspal Singh Saini, and S.R.S Iyengar. Privacy Preserving Network Analysis of Distributed Social Networks. In *International Conference on Information Systems Security (ICISS)*, volume LNCS, Springer, pages 336–355, 2016.

[70] Paul Palathingal, Thomas E. Potok, and Robert M. Patton. Agent Based Approach for Searching, Mining, and Managing Enormous Amounts of Spatial Image Data, Available online at www.aser.ornl.gov/Publications/FLAIRS05.pdf).

[71] Open Geospatial Consortium (OGC), http://www.opengeospatial.org/about, 2005.

[72] Aijun Chen, Liping Di, Yaxing Wei, Yang Liu, Yuqi Bai, Chaumin Hu, and Piyush Mehrotra. Grid Computing enabled geospatial catalogue web service. In *American Society for Photogrammetry and Remote Sensing*, volume 4, 2005, Baltimore, USA.

[73] Jiehai Cheng and Wei Li. Research of the Application of Grid Computing on Geographical Information System. In *Proceedings of IC-NIDC, IEEE*, pages 1070–1075, 2009.

[74] Wang Yaqin, Gao Hua, Sun Cuiya, and Shen Weixing. Research on Multi-source Heterogeneous Spatial Data Exchange Model Based on Ontology and GML. In *The Eight International Conference on Electronic Measurement and Instruments, IEEE*, volume 3, pages 931–934, 2007.

[75] Teerayut Horanont, Nitin Kumar Tripathi, and Venkatesh Raghvan. A Comparative Assessment of Internet GIS Server Systems from www.gisdevelopment.net, Accessed on September 2010.

[76] Ron Lake. Introduction to GML: Geography Markup Language from http://www.w3.org/Mobile/posdep/GMLIntroduction.html, Accessed on October 2010.

[77] David A. Patterson. Technical perspective: the data center is the computer.

Communications of the ACM, 51(1):105–105, 2008.

[78] Tom White. *Hadoop: The Definitive Guide, 4th Edition*. OReilly Media, Inc., 2015.

[79] Feng Li, Beng Chin Ooi, M. Tamer Ozsu, and Sai Wu. Distributed Data Management Using MapReduce. *ACM Computing Surveys*, 46(3-31), 2014.

[80] Owen OMalley and Arun C. Murthy. Yahoo! Winning a 60 Second Dash with a Yellow Elephant. http://sortbenchmark.org/Yahoo2009.pdf (April 2009).

[81] Eric Anderson and Joseph Tucek. Efficiency matters! *ACM SIGPOS Operating Systems Review*, 44(1):40–45, 2010.

[82] Foto N. Aftari and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology, EDBT*, pages 99–110, 2010.

[83] Dawei Jiang, Beng Chin, Lei Shi, and Sai Wu. The Performance of MapReduce: An In-depth Study. . In *In Proceedings of the VLDB Endowment*, volume 3(1-2), pages 472–483, 2010.

[84] Rajashekhar M. Arasanal and Daanish U. Rumani. Improving MapReduce Performance through Complexity and Performance Based Data Placement in Heterogeneous Hadoop Clusters. *Distributed Computing and Internet Technology, LNCS*, 7753:115–125, 2013.

[85] Burhan UI. Islam Khan, Rashidah F. Olanrewaju, Hunain Altaf, and Asadullah Shah. Critical insight for MapReduce optimization in Hadoop. *International Journal of Computer Science and Control Engineering*, 2(1):1–7, 2014.

[86] Matei Zaharia, Andy Konwinski, Anthony D. Joseph Randy Katz, and Ion Stoica. Improving MapReduce Performance in Heterogeneous Environments . In *In Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 29–42, USENIX Association Berkeley, CA, USA 2008.

[87] Jeffrey Dean and Sanjay Ghemawat. SimplifiedDataProcessing on Large Clusters. *Magazine Communications of the ACM - 50th anniversary issue: 1958 2008*, 51(1):107–113, 2008.

[88] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, ISBN 1558605703, 1999.

[89] Steffen Heinz and Justin Zobel. Efficient single-pass index construction for text databases. *JASIST*, 54(8):713–729, 2003.

[90] Anthony Tomasic and Hector Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems . In *In Proceedings of PDIS*, pages 8–17, 1993.

[91] Mike Cafarella and Doug Cutting. Building nutch: Open source search. *ACM*

Queue, 2(2):54–61, 2004.

[92] Richard M.C. McCreadie, Craig Macdonald, and Ladh Ounis. Comparing Distributed Indexing: To MapReduce or Not?. LSDS-IR Workshop, July 2009, Boston, USA, http://ceur-ws.org/Vol-480/paper5.pdf.

[93] Christos Doulkeridis and Kjetil Norvag. A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal*, pages 1–27, 2013.

[94] Applications powered by Hadoop: http://wiki.apache.org/Hadoop/poweredby.

[95] Hadoop. In http://Hadoop.apache.org.

[96] Yahoo! Launches Worlds Largest Hadoop Production Application, http:/tinyurl.com/2hgzv7.

[97] Performance Measurement of a Hadoop Cluster, http://www.acma.com/acma/pdfs/AMAX Emulex Hadoop Whitepaper.pdf.

[98] Understanding Hadoop Clusters and the Network, http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/.

[99] Dominique A. Heger. Hadoop Design, Architecture & MapReduce, Performance. http://www.datanubes.com/mediac/HadoopArchPerfDHT.pdf .

[100] Grid Computing with Oracle, An Oracle Technical White Paper, March 2005.

[101] T. Bestull. *Parallel paradigms and practices for spatial data, Ph.D. dissertations, CS-TR-2897, Computer Science Dept., Univ. Of Maryland.* 1992.

[102] Pinaki Mitra, Asish Mukhopadhyay, and S. V. Rao. Computing the Closest Point to a Circle. In *Canadian Conference on Computational Geometry, Halifax, Nova Scotia* , pages 1–4, 2003.

[103] Pinaki Mitra and Asish Mukhopadhyay. Computing a closest point to a query hyperplane in three and higher dimensions. In *International Conference on Computational Science and Its Applications* , volume LNCS, Springer, pages 787–796, 2003.

[104] Hanan Samet. *The Design and Analysis of Spatial Data Structures, Addison-Wasley, Reading, MA*. 1990.

[105] Jurg Nievergelt, Hans Hinterberger, and Kenneth Clem Sevcik. The grid file: An adaptable, symmetric multikey file structure. In *In Proceedings of the Third ECI Conference, Duijvestijn and Lockemann, Eds., LNCS 123, Springer-Verlag*, volume 2, pages 236–251, 2010.

[106] Randal C. Nelson and Hanan Samet. A consistent hierarchical representation for vector data. *Computer Graphics*, pages 197–206, 1986.

[107] Census 2000 tiger/line data. In www.esri.com/data/download/census2000-tigerline or www.census.gov/geo/tiger.

[108] Pinaki Mitra, Girish Sundaram, and Sreedish PS. Just In Time Indexing. In *In Proceedings of NETs2012 International Conference on Internet Studies* , pages 1–13, 2012.

[109] Ibrahim Kamel and Christos Faloutsos. Parallel R-trees. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, volume 21(2), pages 195–204, 1994.

[110] Yannis Theodoridis and Timon Sellis. Optimization issues in R-tree construction. In *In Proceedings of the International Workshop on Geographic Information Systems*, pages 270–273, 1994.

[111] Yu Sun and Li Guoqing. Interoperability Research of Heterogeneous GIS Based on Spatial Information Grid. In *International Conference on Computer Science and Software Engineering, IEEE Computer Society*, pages 41–44, 2008.

[112] Zhanfeng Shen and et. al. Architecture design of grid GIS and its applications on image processing based on LAN. *Information Sciences*, 66:1–17, 2004.

[113] Yanfeng Shu, Jack Fan Zhang, and Xiaofang Zhou. A Grid-Enabled Architecture for Geospatial Data Sharing. In *Proceedings of the IEEE Asia-Pacific Conference on Services and Computing*, pages 369–375, 2006.

[114] Gobe Hobona, David Fairbairn, and Philip James. Workflow Enactment of Grid-Enabled Geospatial Web Services. In *Proceedings of the 2007 UK e-Science All Hands Meeting*, 2007.

[115] Zhou Huang, Yu Fang, Bin Chen, and Xi Wu. Development of a Grid GIS Prototype for Geospatial Data Integration. In *GCC08*, pages 628–631, 2008.

[116] Qingfeng Guan. Grid-enabled Urban-CA GIS. https://www.researchgate.net/publication/240721200_Grid-enabled_Urban-CA_GIS.

[117] OGSA-DAI. http://www.ogsadai.org.uk/ .

[118] Xiaosheng Liu, Xiaobin Huang, and Zhiyong Zhao. Research on Spatial Databases based on Grid Computing. In *International Conference on Communications and Mobile Computing, IEEE Computer Society*, 2009.

[119] Jinsong Gao, Wen Zhang, and Lingkui Meng. A Method for Heterogeneous Spatial Data Integration with Storage Agent in Grid. In *IEEE Computer Society*, pages 1230–1233, 2005.

[120] Zhen-Chun Huang and Guo-Qing Li. Building data grid for spatial information applications by meta-data adapters. In *International Conference on Computer Science and Software Engineering, IEEE Computer Society*, 2008.

[121] Qinghui Sun, Tianhe Chi, Xiaoli Wang, , and Dawei Zhong. Design of Middleware Based Grid GIS. In *IEEE Computer Society*, pages 854–857, 2005.

[122] Jian Tan and et al. Research on Geographical Information Synchronizing in GRID-GIS. In *International Workshop on Earth Observation and Remote Sensing Applications, IEEE*, 2008.

[123] Qidong Yin, Qi Li, , and Xi Wu. Research on Spatial data interoperability in Spatial Data Grid Environment. In *Third International Joint Conference on Computational Science and Optimization, IEEE Computer Society*, 2010.

[124] Bin Chen Menglong Yan Yong Zhao, Yu Fang and Yixian Sun. A New Grid GIS Prototype for Vector Geospatial Data. In *Eighth International Conference on Grid and Cooperative Computing, IEEE Computer Society*, pages 367–372, 2009.

[125] Zhanfeng Shen, Jiancheng Luo, Guangyu Huang, Dongping Ming, Weifeng Ma, and Hao Sheng. Distributed computing model for processing remotely sensed images based on grid computing. *Information Sciences*, 177:504–518, 2007.

[126] Jinajun Cui, Baoyin Zhang, Xi Wu, Zhou Huang, Yu Fang, and Bin Chen. Design and Implementation of WSRF based GIS Service in Spatial Data Grid. In *Eighth International Conference on Frid and Cooperative Computing, IEEE Computer Society*, 2009.

[127] Lin Wan, Zhong Xie, Liang wu, and Jiayuan Lin. Research on the Key Technologies of Geospatial Information Grid Service Workflow System. In *18th International Conference on Geoinformatics*, 2010.

[128] Min Chen, Jihong Guan, and Xiaobin Huang. Research on Distributed GIS Based on Mobile Agent. *International Archives of Photogrammetry and Remote Sensing*, XXXIII:67–73, 2000.

[129] G. Ji-hong, Z. Shui-geng, B. Fu-ling, and M. Ling-kui. Building Distributed Web GIS: A Mobile Agent Based Approach. *Wuhan University journal of Natural Sciences*, 6(1-2):474–481, 2001.

[130] Tian Gen. A New Method of Distributed Mobile GIS Grid Model Based on Mobile Agent. In *International Conference on Computer Application and System Modeling*, pages 664–668, 2010.

[131] N. Shahriari and C.V. Tao. GIS Applications Using Agent Technology. In *Symposium on Geospatial Theory, Processing and Applications*, 2002.

[132] Jihong Guan, Shuigeng Zhou, and Fuling Bian. A Mobile-Agent and GML Based Frmework for Integrating Distributed GIS. In *IAPRS*, volume XXXIV(2), 2002.

[133] G. Ali, N.A. Saikh, and Z.A. Shaikh. Integration of Grid and Agent System to Perform Parallel Computations in a Heterogeneous and Distributed Environment. *Australian Journal of Basic and Applied Sciences*, 3(4):3857–3863, 2009.

[134] Gen Tian, Jian Li, and Bin Chen. A New Mobile Police Spatial Information Service Grid Computing Model Based on Mobile Agent. In *18th International Conference*

on Geoinformatics, 2010.

[135] Jian Ma, Qiang Liu, Boyan Cheng, and Yuancheng Sun. A Mobile Agent Based Spatial Data Grid. In *IEEE*, pages 3254–3257, 2006.

[136] Qiang Tong and Zhidong Shen. Research of Mobile Agent Technology for GIS Grid. In *Ninth International Conference on Hybrid Intelligent Systems, IEEE Computer Society*, pages 190–194, 2009.

[137] Salvatore Venticinque, Beniamino Di Martino, Rocco Aversa, Richard Olejnik, Iyad Alshabani, and Bernard Toursel. Integrating Distributed Component and Mobile Agent Programming Models in Grid Computing. In *Sixth International Symposium on Parallel and Distributed Computing, IEEE Computer Society*, 2007.

[138] Hadoop-Grid/Big Data & Big Compute with Grid Engine and Hadoop GridEngine.htm.

[139] Hadoop-Grid/Difference between computing with hadoop and grid or cloud computing-Quora.htm.

[140] Bing Tang, Haiwu He, and Gilles Fedak. Hybrid-MR: A new approach for hybrid-MapReduce combining desktop grid and cloud infrastructures. *Concurrency and computation: Practice and experience*, doi: 10.1002/cpe:1–16, 2014.

[141] Bing Tang, Mircea Moca, Stephane Chevalier, Haiwu He, and Gilles Fedak. Towards MapReduce for Desktop Grid computing. In *International conference on P2P, Parallel, Grid, Cloud and Internet computing, IEEE Computer Society*, pages 193–200, 2010.

[142] Gilles Fedak, Haiwu He, and Franck Cappello. BitDew: A data management and distribution service with multi-protocol file transfer and metadata abstraction. *Journal of network and computer applications*, 32(5):961–975, 2009.

[143] Gabriel Antoniu and et. al. Scalable data management for MapReduce based data intensive applications: a view for cloud and hybrid infrastructure. In *IJCC*, volume 2(2-3), pages 150–170, 2010.

[144] Hui Jin, Xi Yang, Xian-He Sun, and Ioan Raicu. Adapt: availability aware MapReduce data placement for non-dedicated distributed computing. In *ICDCS, IEEE*, volume 2(2-3), pages 516–525, 2012.

[145] Kyungyong Lee and Renata Figueiredo. MapReduce on opportunistic resources leveraging resource availability. In *Cloudcom*, pages 435–442, 2012.

[146] Heshan Lin, Xiaosong Ma, and Wu-Chun Feng. Reliable MapReduce computing on opportunistic resources. *Cluster computing*, 15(2):145–161, 2012.

[147] Yonggang Wang et al. Research and implementation on spatial data storage and operation based on Hadoop platform. In *Second IITA International Conference on Geoscience and Remote Sensing (IITA-GRS), IEEE* , volume 2, pages 275–278,

2010.

[148] Apostolos Papadopoulos and Yannis Manolopoulos. Parallel bulk-loading of spatial data. *High performance computing with geographical data*, 29(10):1419–1444, 2003.

[149] Hanan Samet Eric G. Hoel. Data-parallel primitives for spatial operations using PM Quadtrees. In *Proceedings of International Conference on Computer Architectures for Machine Perception*, 1995.

[150] T. Bestul. *Parallel paradigms and practices for spatial data*. Ph.D. dissertations, CS-TR-2897, Computer Science Dept., Uni. Of Maryland, 1992.

[151] Eric G. Hoel and Hanan Samet. Performance of Data-Parallel Spatial Operations. In *Proceedings of VLDB Conference*, pages 156–167, 1994.

[152] Eric G. Hoel and Hanan Samet. Data-parallel polygonization. *Parallel Computing, Special Issue: High Performance Computing with Geographical Data*, 29(10):1381–1401, 2003.

[153] Eric G. Hoel and Hanan Samet. Data-Parallel R-Tree Algorithms. In *Proceedings of the 22nd International Conference on Parallel Processing, St. Charles, 1L*, 1994.

[154] Eric G. Hoel and Hanan Samet. A qualitative comparison study of data structures for large line segment databases. In *Proceedings of the SIGMOD Conference, San Diego*, 1992.

[155] Eric G. Hoel and Hanan Samet. Data-parallel spatial join algorithms. In *Proceedings of the 22nd International Conference on Parallel Processing, St. Charles, 1L*, 1994.

[156] Eric G. Hoel and Hanan Samet. Benchmarking spatial join operations with spatial output. In *Proc. 21st International Conference on Very Large Databases (VLDB), Zurich, Switzerland*, pages 606–618, 1995.

[157] Kian-Lee Tan, Beng Chin Ooi, and David J. Abel. Exploiting Spatial Indexes for Semijoin-Based Join Processing in Distributed Spatial Database. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):920–937, 2000.

[158] Ming-Ling Lo and Chinya V. Ravishankar. Spatial Hash-Joins. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, volume 25(2), pages 247–258, 1996.

[159] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient Processing of Spatial Joins Using R-trees. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, volume 22(2), pages 237–246, 1996.

[160] Oliver Gunther. Efficient Computation of Spatial Joins. In *Proceedings of IEEE International Conference on Data Engineering*, pages 50–59, 1993.

[161] Ming-Ling Lo and Chinya V. Ravishankar. Spatial Joins Using Seeded Trees. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 209–220, 1994.

[162] Jignesh M. Patel and David J. DeWitt. Partition Based SpatialMerge Join. *ACM SIGMOD Record*, 25(2):259–270, 1996.

[163] Zhou Huang, Yu Fang, Xuetong Xie, and Mao Pan. Geobarn: A practical grid geospatial database system. *Advances in Electrical and Computer Engineering*, 9(3):7–11, 2003.

[164] Hu Cao, Ouri Wolfson, and Goce Trajcevski. Spatio-temporal data reduction with deterministic error bounds. In *The VLDB Journal*, volume 15(3), pages 211–228, 2006.

[165] Sherri K. Harms, Jitender Deogun, and Steve Goddard. Building knowledge discovery into a geo-spatial decision support system. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 445–449, 2003.

[166] John P. Hayes, Trevor N. Mudge, Quentiu F. Stout, Stephaen Colley, and John Palmer. Architecture of a Hypercube Supercomputer. In *International Conference on Parallel Processing*, 1986.

[167] Guy E. Blelloch and James J. Little. Parallel solutions to geometric problems on the scan model of computation. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume 3, pages 218–222, 1988.

[168] A.N. Yzelman. *R-Trees: An efficient structure for spatial data management.* Faculty of science theses, Master Thesis, Open access version via Utrecht University Repository, 2011.

[169] Houman Alborzi and Hanan Samet. Execution time analysis of a top-down R-Tree construction algorithm. *Information Processing Letters, Science Direct, Elsevier*, 101(6-12), 2007.

List of Publications

Papers published

1. Hari Singh and Seema Bawa, *"A MapReduce-based Scalable Discovery and Indexing of Structured Big Data"*, Future Generation Computer System The International Journal of Grid Computing and eScience from Elsevier, Vol. 73, pp. 32-43, 2017, ISSN: 0167-739X, (SCIE with 2016/17 IF 3.99).

2. Hari Singh and Seema Bawa *"A Survey of Traditional and MapReduce-Based Spatial Query Processing Approaches"*, SIGMOD RECORD from Association of Computing Machinery (ACM), Vol. 46, No. 2, 2017, ISSN:0163-5808, E-ISSN:1943-5835, (SCIE with 2016/17 IF 0.875).

3. Hari Singh and Seema Bawa, *"A MapReduce-Based Efficient H-bucket PMR Quadtree Spatial Index"*, Computer systems Science and Engineering from CRL Publishing Limited, Vol. 32, No. 5, 2017, ISSN: 0267-6192, (SCIE with 2016/17 IF 0.384).

4. Hari Singh and Seema Bawa, *"Spatial Data Analysis with ArcGIS and MapReduce"*, Proceedings of the 2016 International Conference on Computing, Communication, and Automation (ICCCA), pp. 45-49, IEEE Xplore Digital Library, doi:10.1109/CCAA.2016.7813687, (Scopus Index).

5. Hari Singh and Seema Bawa, *"IGSIM: An Architecture for High Performance Spatial Analysis"*, International Journal of Computer Science and Information Security (IJCSIS), Vol. 14, No. 11, pp. 302-309, November 2016, ISSN: 1947-5500, (ESCI).

6. Hari Singh and Seema Bawa, *"Scalability and Fault Tolerance of MapReduce for Spatial Data"*,Global Journal of Engineering Science and Research Management, 3(8), pp. 97-103, August 2016, ISSN:2349-4506, (Thomson Reuter Endnote).

7. Hari Singh and Seema Bawa, *"Evolution of Grid-GIS Systems"*, International Journal of Computer Science and Telecommunications (IJCST), ISSN 2047-3338, Vol. 3, Iss. 3, pp. 36-40, March 2012, (Peer-reviewed Non-SCIE).

Papers in communication

1. Hari Singh and Seema Bawa, *"An Improved Integrated Grid and MapReduce-Hadoop Architecture for Spatial Data: Hilbert TGS R-Tree-based IGSIM"*.

2. Hari Singh and Seema Bawa, *"Integrating Grid-GIS with a MapReduce-based Parallel Priority R-Tree index for Efficiently Handling Spatial Data and Queries"*.