DISTRIBUTED COMPUTATION ON SPATIAL DATA ON HADOOP CLUSTER

MTP STAGE 1

REPORT

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology

By

Abhishek Sagar
Roll No: 10305042

Under the Guidance of

Prof. Umesh Bellur

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai-400076

Acknowledgement

I would like to express my deep gratitude to ***Prof. Umesh Bellur****, who has always been making things simple to understand. Without his deep insight into this domain and his valuable time for this project, it would not have been possible for me to move ahead properly. He has been remarkable in his attempt to keep me motivated in this project and has always tried to improve me with proper feedback.*

Abhishek Sagar
(10305042)
Mtech-II

Abstract

MapReduce is a widely used parallel programming model and computing platform. With MapReduce, it is very easy to develop scalable parallel programs to process data-intensive applications on clusters of commodity machines. Spatial Databases such as postgreSQL , Oracle have been extensively in use to perform spatial SQL Query manipulation on large datasets. But spatial database on a single machine has its limitations with respect to the size of the datasets it can process. For instance, Spatial Queries Performing Spatial joins between two large tables take unreasonably large time to output the entire set of results, Or in many cases the Query breach the Maximum available memory limit of the system resulting in System Or Server Crash.

In this effort, we have brought the power of distributed concurrent computation to spatial data analysis of large data sets. We evaluated the performance and efficiency of spatial operation in Hadoop environment with the real world data set with proper justification of why we need cluster to perform that particular spatial operation . It demonstrates the applicability of cloud computing technology in computing-intensive spatial applications and compare the performance with that of single spatial databases. We also address the architectural differences between Hadoop framework and Distributed database systems , and list the pros and cons of each. The report content has been organized as follows : Chapter 1 prepares the GIS back ground and introduce Hadoop and Map-Reduce. Chapter 2 outlines the literature survey and problem formulation for MTP Stage 1. Chapter 3 discusses the implementation and performance aspects of spatial operations on Hadoop platform in detail. Chapter 4 concludes the work and outlines the future work that we intend to do in MTP Stage II. Chapter 5 presents 'References'.

Contents

1. Introduction

Hadoop[1][2] provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce programming model. An important characteristic of Hadoop is the partitioning of data and computation across potentially thousands of hosts, and executing application computations in parallel close to their data. A Hadoop cluster scales computation capacity, storage capacity and IO bandwidth by simply adding commodity servers. Hadoop clusters at Yahoo! span 25,000 servers, and store 25 petabytes of application data, with the largest cluster being 3500 servers. One hundred other organizations worldwide report using Hadoop.

Keeping in mind the above Benefits that Hadoop offers , It is a potential platform to perform the computation on the Spatial Data. GIS is now playing an important role in many areas of modern city. Space information has become the basic infrastructure of modem digital city and is the integral part of information construction. Generally, GIS functions such as spatial analysis is involved in a lot of vector data (points, lines or polygons) and raster data (satellite or aerial images). This type of data is periodically generated via special sensors, satellites or GPS devices. Due to the large size of the spatial data repositories and the computationally intensive nature of analysis operations, it makes such analysis a perfect candidate for parallelization.

Cloud computing[3] is a new term for a long-held dream of computing as a utility, which has recently emerged as a commercial reality. Cloud computing provide service over the

Internet to users based on large scalable computing resources. Cloud computing can improve system performance, computing and storage capability greatly, and reduce software and hardware cost effectively. In this effort, we import cloud computing technology including Hadoop platform and MapReduce parallel computing model into the domain of geospatial data Analysis. We have studied those key technology problems including spatial data storage, spatial data index and spatial operation in the application of GIS. Aiming at the characteristics of spatial operators, we have designed the process flow of spatial data. We evaluate their performance using real spatial data set based on the actual implementations of these spatial algorithms on Hadoop. The experiment results show that MapReduce is applicable for computing-intensive spatial applications.

1.1 GIS Background - Spatial data types & Operations

1.1.1 Spatial Data Types

The real world can only be depicted in a GIS through the use of models that define phenomena in a manner that computer systems can interpret, as well perform meaningful analysis . The data model determines how GIS data are structured, stored and processed. Spatial Data Models is primarily classified into two types :

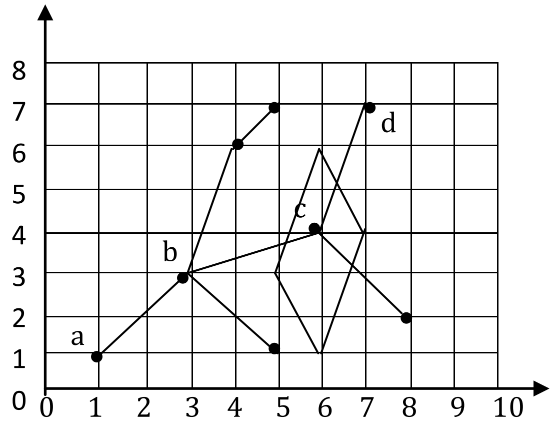
1. VECTOR DATA MODEL
2. RASTER DATA MODEL

VECTOR DATA MODEL: The Vector Data Model uses points and their x-,y- coordinates to construct a spatial feature such as point , line , area , region etc . A point may represent a well , a benchmark or a gravel pit whereas a line may represent a road, a stream or a administrative boundary . The basic vector data types comprises of the **POINTS**, **LINES, POLYLINES, POLYGONS**, and **REGIONS**.

- **POINT DATA:** The Point data type is the simplest data type. It is stored as a pair of X ,Y Coordinates . It represents the entity whose size is negligible as compared to the scale of the MAP such as it may represent a city or a town on the map of a country, or may represent a shopping complex on the map scaled to represent the city.

- **LINES AND POLYLINES:** A LINE is formed by joining two POINTS in a end to end manner. A Line is represented as a pair of POINT data types. POLYLINES comprises of a series of lines connected in an end to end manner. In Spatial Databases, POLYLINE is represented as a sequence of lines such that end point of a line is the starting point of next line in series. POLYLINES are used to represent spatial features such as Roads, Rivers, Streets, Highways, Routes Or any one Dimensional spatial feature.

- **POLYGONS:** POLYGON is one of the most widely used spatial data type. They capture two Dimensional spatial features such as Cities, States, Countries etc. In Spatial Databases, polygons are represented by the ordered sequence of Coordinates of its Vertices, first and the last Coordinates being the same. In the same figure , one POLYGON is also shown. In Spatial Databases, this shall be represented as **POLYGON((6 1,7 4,6 6,5 3,6 1))**. Not that first and last coordinates has to be same.



VECTOR DATA REPRESENTATION

Fig 1.1

Fig 1.1, one of the LINESTRING is represented by connecting the points a, b, c & d. In Spatial Database , this Shall be represented as **LINESTRINGS(1 1,3 3,6 4,7 7).**

- **REGIONS**: Regions is another significant spatial data model. A Region is a collection of overlapping, non-overlapping or disjoint polygons. Regions are used to represent the spatial feature such as: area burned in 1917 forest fire and area burned in 1930 fire or the State of Hawaii, including several islands (polygons). A region can also have a void area contained within it, called a hole.

<u>**RASTER DATA MODEL**</u>: The Vector data model uses the geometric objects of point, line , and area to construct discrete features of well defined locations and shapes , the vector data model does not work well with spatial phenomenon that vary continuously over the space such as precipitation, elevation and soil erosion . A better approach for representing continuous phenomena is the Raster data model.

The Raster Data model uses a Regular Grid to cover the space and the value in each grid cell to correspond to the characteristics of a spatial phenomenon at the cell location. Conceptually, the variation of the spatial phenomenon is reflected by the changes in the cell value. A wide variety of used in GIS are encoded in raster format. They include digital elevation data, satellite images, scanned maps and graphics files. For Example, Grid cab be used to represent the height of every point on the land surface above the Sea Level. This type of information which smoothly varies from point to point on spatial Surface is difficult to model as Vector data. In this example each grid cell stores the value of height of a point it represent on the surface from sea level.

1.1.2 Common Spatial Operations

This section discusses simple operations that take one or many themes as input and give as a result a theme. The geospatial information corresponding to a particular topic is gathered in a theme. A theme is similar to a relation as defined in the relational model. It has a schema and instances. A theme is hence a set of homogeneous geographic objects (i.e., objects having the same structure or type). A theme has two set of Attributes viz **Descriptive attributes and Spatial attributes**.

PostgreSQL[4] has a support to integrate the SQL queries with Geo-spatial operations. We shall explain the effect of Geo-spatial operations when applied on a theme representing a piece of land such as a state or a country . Below is the List of the operations each of which is briefly discussed next . PostgreSQL has the feature to augment SQL Queries with these operations , and thus has a support to manipulate Geo-Spatial Data in manner that doesn't differ much from the way the SQL Queries are fired on RDBMS.

1. **THEME UNION:**

Two themes having the same Descriptive attribute as well as spatial attribute types are combined together to form a larger theme of larger geometry.

2. **THEME OVERLAY(Intersection):**

The Descriptive attributes of input themes are unioned whereas Geometry is intersectioned .

Eg: let **theme1** are the drought-hit towns in the state and let **theme2** are the towns with population over 50 million. Then performing **theme1 Overlay theme2** gives the theme **theme3** as the towns which are drought hit and have population over 50 million .

3. **THEME SELECTION:**

It is used to select only those tuples of the theme which satisfies the given condition. Here the Condition can be applied on Descriptive attribute Or spatial attribute of a theme or both. The Query such as *Name and population of countries of 50 million inhabitants or more* are usually addressed by this operation.

4. **THEME MERGER:**

The merger operation performs the geometric union of the spatial part of *n* geographic objects that belong to the same theme, under a condition supplied by the end user. Observe the difference with the theme union, which takes as arguments two themes and gathers them into a single theme. Merger relies on the concept of object aggregation.

5. THEME WINDOWING:

By windowing a theme, one obtains another theme, which includes only those objects of the input theme that overlap a given area or window, which is usually rectangular.

6. THEME CLIPPING:

Clipping extracts the portion of a theme located within a given area. As opposed to windowing, the geometry of an object in the result corresponds exactly to the intersection of the geometry of the geographic objects and the geometry of the area.

1.2 Hadoop and its Architecture

The **Hadoop Distributed File System (HDFS)** is a distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject.

1.2.1 Assumptions & Goals

1. **Hardware Failure:** Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

2. **Streaming Data Access:** Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access.

3. **Large Data Sets:** Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

4. **Simple Coherency Model:** HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

5. "Moving Computation is Cheaper than Moving Data" : A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

6. Portability Across Heterogeneous Hardware and Software Platforms : HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

1.2.2 HDFS Architecture

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

Data Replication: HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are **write-once** and have strictly one writer at any time. The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Block report from each of the DataNodes in the cluster. A Heartbeat sensed by the NameNode implies that the DataNode is functioning properly. A Block report contains a list of all blocks on a DataNode.

HDFS ARCHITECTURE

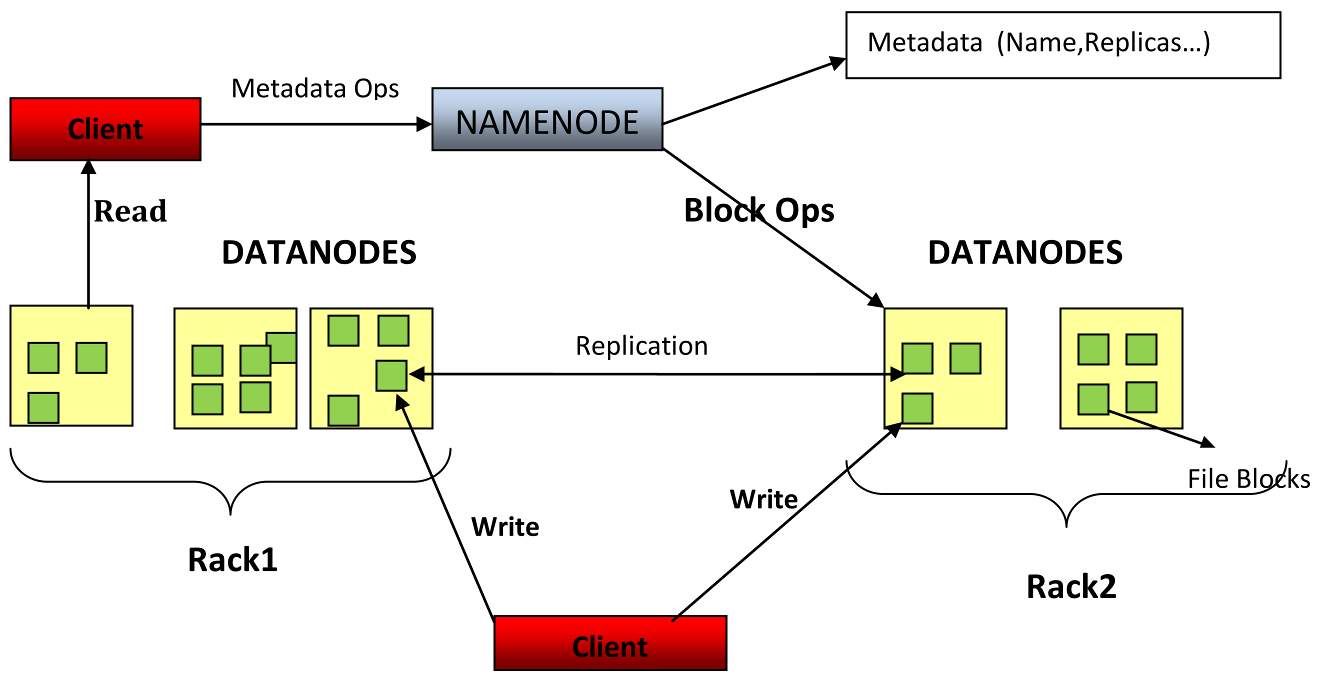


Fig 1.2 HDFS Architecture

Data Disk Failure, Heartbeats and Re-Replication : Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

1.2.3 Map-Reduce Programming Model

Map-Reduce[5],[6] is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the pro-gram's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers with parallel and distributed systems to easily utilize the resources of a large distributed system.

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: **Map and Reduce.**

MAP Function: Map Function, Written by the User, takes an input pair and produces a set of Intermediate Key/Value Pairs. The Map-Reduce library groups together all intermediate values associated with the same intermediate key **'*I*'** and passes them to the Reduce function.

REDUCE Function: Reduce function, also written by the user, accepts an intermediate key **'*I*'** and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

The Whole Map-Reduce Strategy is depicted in the diagram below:

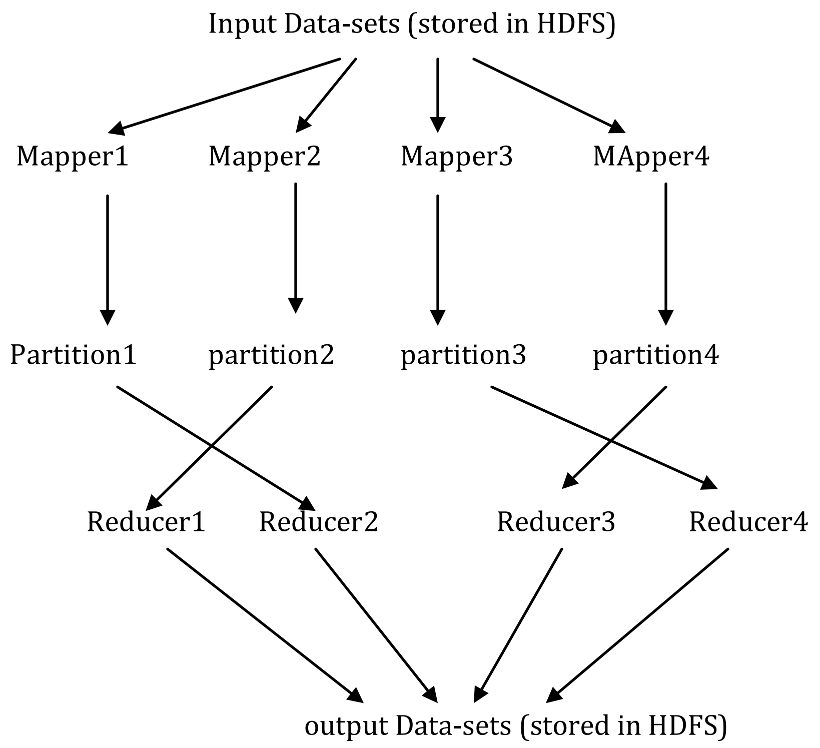


Fig 1.3 Map-Reduce programming Model

To Illustrate the concept of Map-Reduce with the help of example , the diagram below shows the simple 'word count' example. Given an input file , it is required to compute the frequency of each word in the file using Map-Reduce strategy.

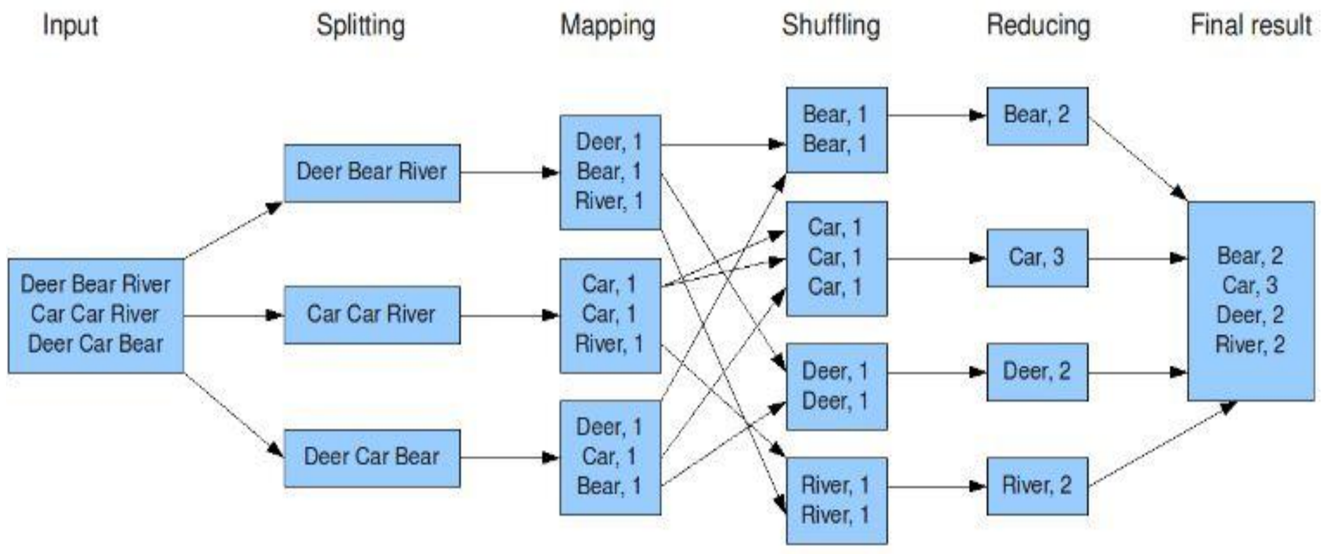


Fig 1.4 Map-Reduce solution to word count problem

Explanation: In the above Example , the input file contains the textual data , the chunks of which are distributed across multiple nodes in HDFS(splitting-phase). The Mappers run in parallel on slave nodes and process the chunk of input file that is local on the machine on which mappers are executing. Mappers output Key-value pairs corresponding to each word processed (Mapping-phase). Here value of each word is one. In shuffling phase , all values which are associated with the same key are grouped together and are made available to the reducer process. Shuffling phase is provided by the framework, user do not have to explicitly code for it. Reducing process then process all values corresponding to the same key. Since Key here is the word itself , therefore , Reducer process simply adds the 'value' of a key to determine the frequency of this word. The reducer processes runs in parallel on slave machines, this computing the frequency of each word in parallel.

1.2.4 Hadoop Vs Distributed Databases

We Studied the Hadoop Map-Reduce Concept of parallelizing the computation (spatial operations) across multiple machines each running Hadoop client. The Obvious Question that reader might raise is Why Hadoop when Distributed/parallel Database systems are already in place! What is the criteria that one should choose to use the Hadoop for carrying out computation on large data sets over parallel databases and vice Versa ? What type of applications are suited for one over the other ? We try to answer these Questions by outlining the comparative study of the two technologies with the help of the table below.

we contrast the difference between Hadoop and Parallel DBMS[17],[18], We briefly give the explanation of some of the distinguishing factor listed in the table 4.1

Scalability: Parallel database systems scale really well into the tens and even low hundreds of machines. Until recently, this was sufficient for the vast majority of analytical database applications. Even the 'enormous eBay 6.5 petabyte database' (the biggest data warehouse that 've existed) was implemented on a (only) 96-node Greenplum DBMS[16]. Unfortunately, parallel database systems, as they are implemented today, unlike Hadoop, they do not scale well into the realm of many thousands of nodes. Parallel DBMS have been designed by keeping the assumption in mind that node failure is a 'Rare' event. The probability of a node failure increases with the increase in the size of the Cluster.

Fault Tolerance: Fault tolerance is the ability of the system to cope up with node/task failures. A fault tolerant analytical DBMS is simply one that does not have to restart a query if one of the nodes involved in query processing fails.

Fault tolerance capability of the parallel DBMS is much inferior to that of Hadoop. Hadoop has been especially designed to exhibit excellent Scalability property and Fault tolerance capability. The amount of work that is lost when one of the node in the cluster fails is more in dist/parallel DBMS than in case of Hadoop. In parallel DBMS, the intermediate results of query are pipelined to next query operator or another sub-query without having written to disk**.** Now if any Sub-Query fails, the intermediate results processed so far are lost and entire query have to be restarted again. However, in Hadoop, the intermediate results of the mappers (Or Reducers) are always written to the disk before they are fetched by the Reducers (Or mappers of the next Map-Reduce stage). Thus, instead of pipelining the intermediate results/data to subsequent processes, Hadoop processes themselves are pipelined to operate on target data. In case of a task/node failure, the same task is restarted on another node to operate on the target intermediate data which still exist on the disk.

Performance: Parallel DBMS have been designed to work in **real time system** and therefore what is important is performance, whereas Hadoop has been designed for **batch processing.** Hadoop gives up some performance in other areas where there are no tradeoffs for scalability. Hadoop was not originally designed for structured data analysis, and thus is significantly outperformed by parallel database systems on structured data analysis tasks. Another contributing factor for Hadoop's slower performance is that the default configuration of Hadoop stores data in the accompanying distributed file system (HDFS), in the same textual format in which the data was generated. Consequently, this default storage method places the burden of parsing the fields of each record on user code. This parsing task requires each Map and Reduce task repeatedly parse and convert string fields into the appropriate type.

Data Loading : Once a DBMS is up and running properly, programmers must still write a schema for their data (if one does not already exist), then load the data set into the system. This process takes considerably longer in a DBMS than in an MR system, because the

DBMS must parse and verify each datum in the tuples. In contrast, the default (therefore most common) way for MR programmers to load their data is to just copy it into the MR system's underlying distributed block-based storage system.

To Summarize , Table below shows highlights the major differences between two technologies.

	HADOOP	**Dist/parallel DBMS**
Scalability	Very flexible , high , up to thousands of nodes	Low, from few dozen to at most a hundred no of nodes.
Fault tolerance capability	Very high , esp. designed for this	Low , lot of work is lost when node failure occurs
Performance	Lower , because it process unstructured data and write intermediate results to disk	Good , process structured data , and avoid writing intermediate results onto disk
Applications type	Write once , Read many , No data updating – Batch Processing	Read many + frequent updating – User Interactive
Data type	Unstructured , Ad-hoc data sets	Structured , Schema based data , Metadata , indexed data
Data loading	Faster , Textual data in the form as it was originally collected	Slow , Schemas design and refinement has to be done before hand
Cost	Low , free open Source , require no specialized hardware	No good open Source Parallel DBMS available
Age	Launched in 2004 by Google, young piece of software , much scope of improvement & optimization in future releases	Exist for over two decades, much of the optimization work has already been done
Queries Representation	Map-Reduce programs , user has to write atleast 60-70 lines of code even for simplest logic	SQL Queries , some times difficult to transform problem into SQL , eg: Graph based problems , Compact , just one line required to represent a problem

Table 1.1 A comparison between Hadoop and Dist/parallel DBMS

Thus , Hadoop and DDMS differ significantly and each has their own advantages and drawbacks. Hadoop is good when we need to process large unstructured data sets , of the order of thousands of terabytes or petabytes, and you have large number commodity machines available (in thousands) , but Hadoop lacks in the performance aspects. On the other hand, DDBMS process structured data , with much better performance but have poor scalability. Also , fault tolerance capability of Hadoop is much better than DDBMS.

2. Literature Survey

2.1 Related Work

To begin With , [7] Discusses the Implementation of Common Spatial Operators such as **Geometry Intersection** on Hadoop-Platform by transforming it into Map-Reduce Paradigm. It throws the light on what Input/output Key Value pair a Map-Reduce programmer should choose for both Mapper & Reducers to effectively partition the data on several slave machines on a cluster and carrying out the spatial Computation in parallel on slave machines. This paper also presents a performance evaluation of spatial operation comparing Spatial Database with Hadoop platform. The results demonstrate the feasibility and efficiency of the MapReduce model and show that cloud computing technology has the potential to be applicable to more complex spatial problems.

[8] Discusses the Implementation of Spatial Queries into Map-Reduce Paradigm which particularly involves **the Spatial join** between two or more heterogeneous spatial data sets. The strategies discussed include strip-based plane sweeping algorithm, tile-based spatial partitioning function and duplication avoidance technology. Paper experimentally demonstrates the performance of SJMR (Spatial Join with Map-Reduce) algorithm in various situations with the real world data sets. It demonstrates the applicability of computing-intensive spatial applications with MapReduce on small scale clusters.

[9] is the optimization work done on [8] . It discusses effective strategies to partition the Spatial data in the Mapper Phase so that Most Reducers running on the slave machines get the fair share of data to processed i.e. it should not happen that some reducer gets very less data for processing while other reducers are just overwhelmed with test data to be processed. Paper shows Experimental statistics & results that shows the improvement in Overall Cluster Utilization , Memory Utilization & Run time of the Hadoop Job.

[10] Discusses the Strategies to transform the various Graph manipulating Algorithms such as Dijkastra's Single Source Shortest Path Algorithm , Bipartite Matching , approximate vertex and edge covers and minimum cuts etc into Map-Reduce form. Generally the Map-Reduce Algorithms to manipulate graphs are iterative in Nature i.e. the problem is solved by processing the graph through a Map-Reduce pipeline, each iteration being Converging towards a Solution. The **key challenge** in Transforming the Graph based problems into map-reduce form is the Partitioning of the graph . It has to be done carefully as the slave machine processing one part of the graph has no information whatsoever about the rest of the Graph. The partitioning of the graph among slave machines must be done so that there is no need for the slave machine to be aware of the rest of the portion of the graph and it can perform computation independently on its own share of Graph.

[11] Discusses the three Stage Map-Reduce Solution to Spatial Problem ANN (All nearest Neighbor)[12]. Since Mapper phase partition the spatial objects and group all those together which lies close to each other within a Rectangular 2D space called **partition**. The Algorithm requires just

one Map-Reduce Stage in case if every object's nearest neighbor is also present with in the same partition as the object is. But what if the nearest neighbor of the object belongs to the adjacent partition. Such objects whose NN is not guaranteed to exist within its own partition are called Pending elements. They make use of the Intermediate Data Structure called **Pending Files** in which they write the pending element and the potential partition which could contain the NN of this pending element. Input to the Mapper Phase of the next Map-Reduce Stage is the Pending Files + Original Data Source . Output of Second Map-Reduce stage produces the final results in which every element is guaranteed to find its Nearest Neighbor. Through this approach , the Map-reduce programming model overcome the communication barrier between slave nodes. Hadoop Platform doesn't allow slave nodes to share information while Map-reduce task is executing. The Data partition that a particular slave node executed is made available to another slave node in the Next Map-Reduce stage.

[13] Discusses the Map-Reduce Strategy to Perform build the Index of large Spatial Data sets. It discusses the Map-Reduce Paradigm to Construct the R-tree[14] in parallel on Slave Machines of Cluster. The Maximum Size of RTree that can be Constructed is Limited by the total size of Main Memory of all Slave Machines of the Cluster.

SUMMARIZATION:

MAP-REDUCE STRATEGY	SPATIAL OPERATION	CLUSTER MEMORY BOUND (Y/N)	# Map-Reduce ITERATIONS	Short-Comings
SJMR-V1	Spatial join with Intersection	Y	1	Unbalanced Partitioning & Duplicate outputs results
SJMR-V2	Spatial join with Intersection	N	1	Unbalanced Partitioning & Duplicate output results
Geo-Spatial Indexing	Quad tree, R-Tree Construction	Y	1	Duplicate Entries
Dispersed MR	ANN problem	N	3	-
	All Pair Distance	N	(n-1)/2 , where n – No of partitions	
Iterative MR	Graph Algorithms: SSSP , APSP, BFS , DFS	N	As many as Diameter of Graph	most of the computation done is discarded

Table2.1 A comparative study of various Map-Reduce Strategies

2.2 PROBLEM FORMULATION

Spatial queries include spatial selection query, spatial join query, nearest neighbor query, etc. Most of spatial queries are computing intensive and individual query evaluation may take minutes or even hours. Parallelization seems a good solution for such problems. However, parallel programs must communicate efficiently, balance work across all nodes, and address problems such as failed nodes. We describe Map-Reduce and show how different ***spatial operations/Graph algorithms*** *can be expressed in this model, without explicitly addressing any of the details of parallelization. We present performance evaluations for Map-Reduce form of spatial operations on a small scale cluster and establish that MapReduce is appropriate for small scale clusters and computing intensive applications in spatial domain.*

3. Experiments and Map-Reduce Algorithms

3.1 Queries with Spatial Join

In this Section we execute the Computation-intensive Spatial Query involving the Spatial Join between two heterogeneous datasets (heterogeneous in the sense that one data set can have POLYGON geometry representing the cities/states etc , whereas other data set could be LINESTRING geometry representing the rivers & roads etc) . We deliberately have chosen such Spatial Query as it involves the each tuple of one data set S to compute against every tuple of other data set , say R.

We shall transform the following Spatial Query into Map-Reduce form:

Temp.csv = roads.csv UNION hydrography.csv

Select p.OID , r.OID , st_astext (st_Intersection (p.the_geom , r.the_geom))
from poygons.csv as p , Temp.csv as r
Where st_Intersects(p.the_geom , r.the_geom) ;

Here we perform the UNION of **roads.csv** and **hydrography.csv** and lets store the final result in **Temp.csv.** The Above Query groups all roads and hydrography together according to the county it intersects with.

3.1.1 Map-Reduce Algorithm

MAP-PHASE : The objective of the MAP-PHASE is the Partition of Data tuples into partitions . Partition is the Rectangle in 2D space. All the data tuples either in S or R data sets , the geometry of which intersects anywhere in partition p shall be categorized to exist in partition p . Thus if geometry of a polygon intersects with 2 or more partitions , then it belongs to all partitions it interests with.

Input output Key Value Pair of Mapper phase :

Input : <Key:Value> = <byte Offset , Geometry of object>

Output : <Key:value_list> = <Partition Number , Geometry.MBR + Geometry of object >

Thus each partition that will be created contains the set of geometry objects , their Minimum Bounding Rectange and OID(Object Identifier) . Each Partition shall contain the spatial objects that lies partially or completely within its boundary.

REDUCER PHASE: The partition , which is the output of the Mapper phase , that is created is processed by Hadoop Slave node. The Hadoop Daemon that process the partitioned data sets is called Reducers which run in parallel on HADOOP Slave machines. In this phase we store all POLYGON object geometries in one ArrayList and LINESTRING Object geometries in another array list . We process the two array List to output all Hydrologies and roads that intersects a polygon. Note that we build the data structures (Two Array Lists) in this phase which stores all data objects of a partition the reducer is processing.

3.1.2 Experiment Set up

Experiment Data : Real World Tiger Line Data of California

Source : http://www.census.gov/geo/www/tiger/

For Simplicity we refine the files to contain only the Object Identifier OID , and the Geometry . **Polygons.csv** are the counties of state of California.

FILE NAME	# OF TUPLES	FILE SIZE
Polygons.csv	62,096	42.7 MB
Roads.csv	20,62,872	210.7 MB
Hydrography.csv	3,44,570	83.2 MB
Total =	**24,69,538**	**336.7 MB**

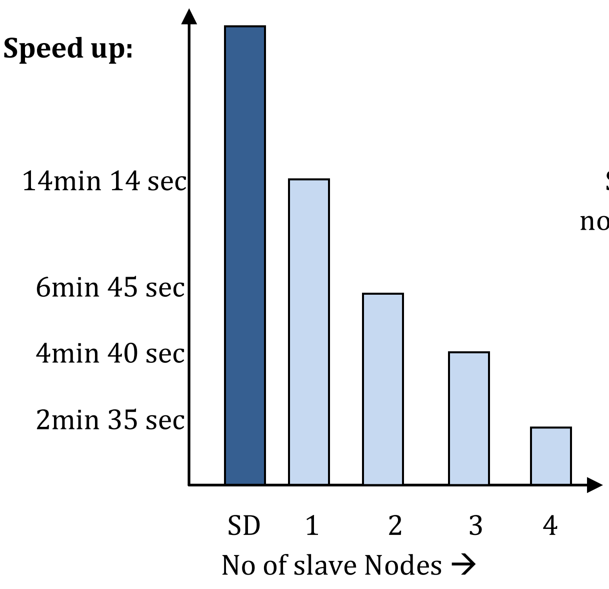
Slave Nodes Configuration is As follows:

Node No	# OF CORES	RAM CAPACITY
Node1	2	1GB
Node2	4	2GB
Node3	4	2GB
Node4	4	2GB

3.1.3 Speed up & CPU Utilization

We execute the Equivalent Map-Reduce Program of the target Spatial Query on the Hadoop Cluster with 1,2,3 & 4 Slave nodes . We monitor the speed up and CPU-Utilization of slave nodes of a cluster.

Speed up:



Speed up Obtained as we increase the No of slave nodes.

SD* - Query Executed on Spatial Databases.

Fig 3.1 Speed up Vs Cluster Size

Observation & Explanation : The above graph shows the amount of time taken by the Hadoop Cluster to output all results of the Query. Each time we add one more slave node to the cluster to demonstrate Speed up. However the Same Query When executed on the postgreSQL Database, it did not return any output for as long as up to 25 minutes of wait . Also, we noticed that 3 out of 5 times the Query causes System Crash. Reason is due to extremely large no of tuples present in **polygons.csv** (62,000)& **Temp.csv**(2.3 million)**.** When Query performed the Spatial join on these two datasets , in the backend the database might be constructing the temporary dataset containing 62,000x2.3million tuples which breached the Maximum Main memory Size of the system which was **1.5GB** resulting the System Crash. However Hadoop Cluster with just one slave node is able to compute the results in **14min 14 sec** time. Reason being that in the Map-Reduce program we implement the Memory efficient **SweepLine Algorithm** [3] to find all Roads & hydrography against each county. Like database, We never build any natural-joined dataset .

CPU-Utilization:

We monitor the CPU-Utilization of Node '3' which is 4 core machine with 2 GB RAM in the experiment , when No of slave nodes in the cluster were 3. We found the Average CPU-Utilization of the Node is **approx 35% .**

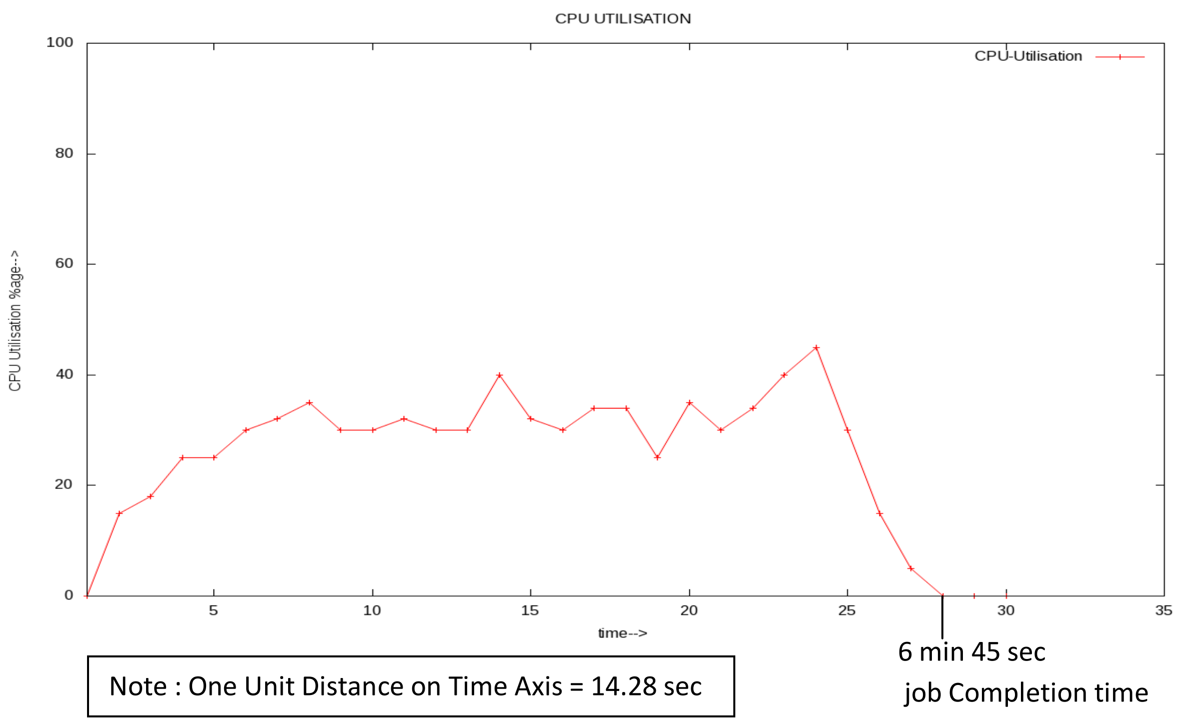


Fig 3.2 graph showing CPU Utilization of node '3' of the cluster

Observation & Explanation: To reason about low CPU utilization We do the Analysis of the amount of Mapper output data each reducer got for processing. We found the some Reducers got the large data tuples from Mapping phase whereas some were moderately loaded. The reason for this varying distribution of Mapper output among reducers is due to the non-uniform mapping of the input tuples to partitions during Mapping Phase. This problem is called **Spatial partition Skew**.

Table below shows the distribution of Input data tuples from **polygon.csv** & **Temp.csv** among 6 partitions :

Partition 0 # of tuples **7,37,593**	**Partition 1** # of tuples **2,53,531**	**Partition 2** # of tuples **08**
Partition 3 # of tuples **94,333**	**Partition 4** # of tuples **9,20,742**	**Partition 5** # of tuples **4,65,759**

We noticed out of 5 reducers running on 3 slave nodes, two processed **1203352 records**(part 0 and 5) and **920742 records**(part 4) where rest of the three reducers processed **94333 , 8, 253531 records**(part 3,2,1 respectively) respectively. Thus, the reducer which processed the smaller partitions finished off quickly and slave machines sits idle whereas heavily loaded reducers hold the Map-Reduce job for a long time. This leads to low-Cluster Utilization and larger execution time of the job. The Experiment statistics we performed are shown below in fig 3.3.

Reducing process R1, R4 are heavily loaded, R5 is moderately loaded whereas R2 & R3 processes very less no of records , 94,333 and 8 only.

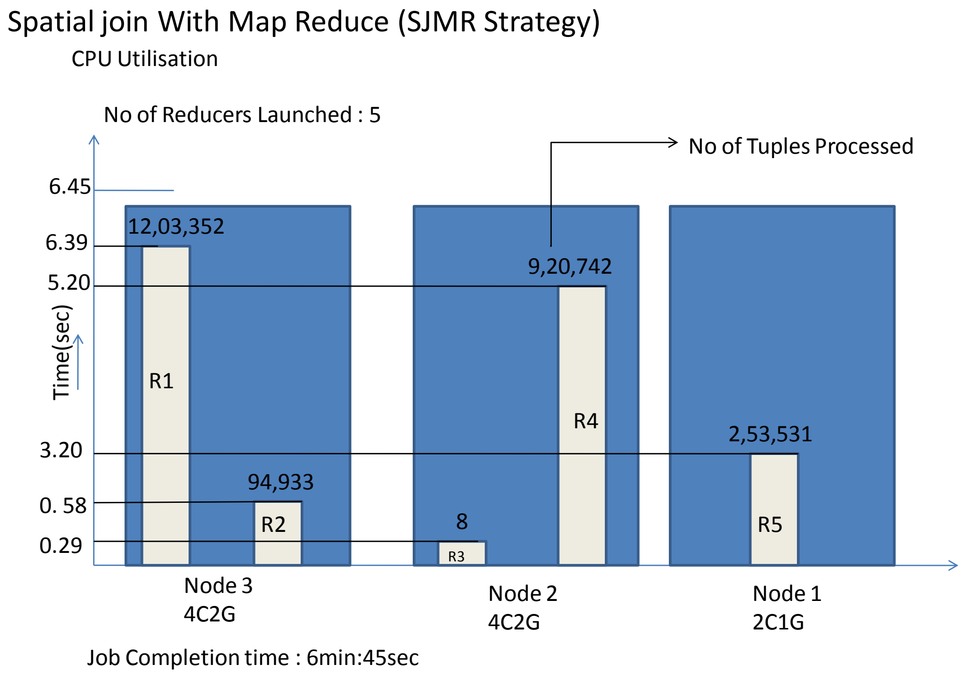


Fig 3.3 graph showing no. of records processed per slave node

3.1.4 Optimization

To equally distribute the objects among all partitions, we use **Tile Based Partitioning Scheme**[3]. In this Scheme , each partition in turn is splitted up into smaller rectangles called Tiles. Each tile has a mapping to exactly one partition. The MBR of the object if overlaps with the Tile **x** (which maps to partition , say, y) , is copied to partition **y**. When we Re-Distribute the tuples among partitions using Tile Based Partitioning method by using 16 tiles , we obtained the following distribution of tuples among six partitions:

Partition 0 # of tuples **4,54,712**	**Partition 1** # of tuples **4,70,709**	**Partition 2** # of tuples **4,93,492**
Partition 3 # of tuples **4,50,812**	**Partition 4** # of tuples **3,97,545**	**Partition 5** # of tuples **2,02,268**

With this uniform distribution of tuples among Reducers, We monitor the cpu Utilization of the Node 3 of the cluster. Figure 3.4 below Shows the higher utilization of the machine resources. Same is true for all other nodes of the cluster as well.

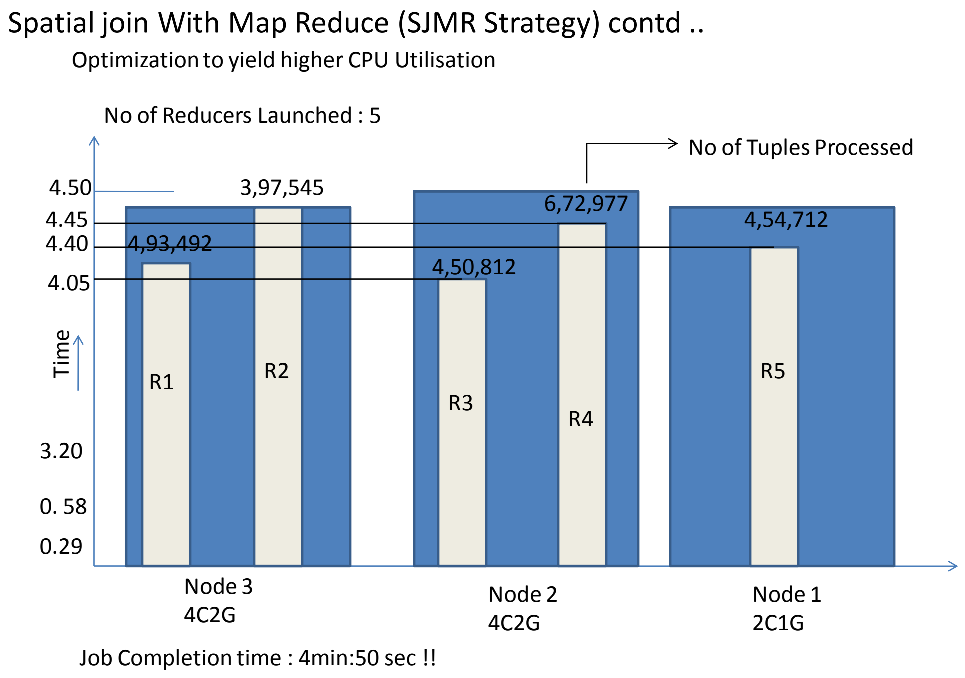


Fig 3.4 graph showing no. of records processed per slave node after optimization

In the above figure, Reducers executed more or less for the same amount of time , and No of tuples processed by each of them doesn't differ much. By equally loading each reducer process, job gets completed **in 4 min:50 sec time while it was 6min:50 sec in the previous case.**

3.1.5 Results and Conclusions

In this Experiment, we describe how spatial join can be effectively implemented with MapReduce on clusters. SJMR algorithm splits the inputs into disjoint partitions with a spatial partitioning function at the Map stage, and merges every partition with a strip-based plane sweeping algorithm.

The strategies of SJMR can also be used in other parallel environments, especially where neither of the inputs has spatial index. The results demonstrate the feasibility and efficiency of the SJMR algorithm, and show that MapReduce is applicable in computation-intensive spatial applications and small scale clusters.

We also demonstrated that Spatial join Query's might result in eating up all the Main Memory of the system if the candidate data sets contains significantly large no of tuples resulting in System Crash or long waiting time.

A Hadoop Cluster of given size (No of slave nodes) has the upper bound on amount of data it can perform spatial join on. It is because the reducer while receiving the Mapper output , first stores the data tuples in the array Lists before carrying out any processing, and we cannot create the Array List more than the main memory size of a slave node. Had the reducer was computing the data in flight without building any data structure to store the entire Mapper output , the Maximum amount of data a cluster(of any size) can process is limited by the Disk Capacity then.

3.2 GeoSpatial Indexing – Rtree Construction

What BTrees are to RDBMS, RTrees are to SDBMS. In this section we discuss the Construction of Small chunks of a large Rtree (large enough to fit in the main memory of a single slave Machine) on slave Machines in Map-reduce Environment.

Motivation : Generally, GIS functions such as spatial analysis is involved in a lot of vector data (points, lines or polygons) and raster data (satellite or aerial images). This type of data is periodically generated via special sensors, satellites or GPS devices. Due to the large size of spatial repositories and the complicacy of the geospatial models, it is essential to keep such a huge data well Indexed so that it can be used later for speedy Manipulation & Computations.

3.2.1 Map-Reduce Algorithm

Map-Reduce Implementation of Rtree Construction is Simple & Straight forward. We describe the Map & the Reduce phase as described below:

MAP PHASE: Exactly same as the one discussed in section 2.1.1 . We split the spatial data sets into partitions such that all data sets that overlaps partially or completely with the particular partition are said to belong to that partitions.

REDUCE PHASE: Rtree Implementation is done in this phase. Rtree is constructed of the Minimum Bounding Rectangle of all data tuples that are processed by this Reducer.

3.2.2 Experimental Setup

Experiment Data : Real World Tiger Line Data of California.

Source : http://www.census.gov/geo/www/tiger/

10 X

FILE NAME	# OF TUPLES	FILE SIZE
Polygons.csv	62,096	42.7 MB
Roads.csv	20,62,872	210.7 MB
Hydrography.csv	3,44,570	83.2 MB
Total =	**24,69,5380**	**3367 MB**

Since , the objective is to Construct the Rtree on slave nodes. As Rtrees only stores the MBRs of the geometry of the object , to construct the Rtree of reasonable size , we copied 10 replicas of our real world data set in HDFS. So total data set that we will index is 3.6GB.

Memory Utilization: This Experiment is Memory Intensive as Rtree is build in Main Memory and when Rtree is fully constructed then only we write it to the disk . so , while Constructing the Rtree on Cluster, Administrator should make sure the no Rtree chunk exceeds the Main Memory size of a slave node otherwise the slave node may crash resulting in job failure.

We perform the memory Analysis of the cluster with three slave nodes as shown below :

SLAVE NODE	**MAIN MEMORY SIZE**	**For Rtree construction C – 200 - 400 Memory Av (MB)**
Node1 : 2C2G	2048	1400
Node2 : 4C2G	2048	1400
Node3 : 4C2G	2048	1400

XCYG represent a machine of X cores and Y GB RAM.

200 : We Subtract 200 MB of memory from the total for Rtree Construction because we saw that even in IDLE State , around 200 MB of memory was being Consumed by System processes .

400 : We Subtract 400 MB of memory from the total for Rtree Construction because Along with the Reducers there are other hadoop processes called **Datanode & tasktracker** which we reserved 200 MB of memory for each .

3.2.3 Observation and Results

Table 3.1 below shows the Complete Memory Consumption on slave Nodes. Column 2 shows the Reducer IDs which ran on the machine listed in column 1 against it. Column 3 is the amount Maximum main memory allotted to a reducer process . Column 5 list the size of Rtree constructed in the main memory by each reducer on slave machines where as column 4 is the no of spatial objects indexed into the R-tree.

Node #	**Reducers ID**	**Memory Allotted(MB)**	**# of Entries**	**Size of Rtree Constructed(MB)**	**Time Taken**
Node3 4C2G	R1	700	49,34,920	71	3min:02 sec
	R2	700	39,75,450	54	2min:42 sec
Node2 4C2G	R1	700	67,29,770	88	3min:16 sec
	R2	700	45,08,120	67	2min:31 sec
Node1 2C2G	R1	1400	45,47,120	63	3min:07sec
	Total	**4200**	**24695380**	**343**	**3 min:21sec**

Table 3.1 Memory Consumption on Slave Nodes

Time it took to build 343 MB of Rtree : **3min:21 sec**

Time to build the Rtree with the same set of data on single machine : **19min : 45 sec !!**

What is the Maximum Data set that can be indexed by this hadoop Cluster of 3 slave nodes ?

Since , 3367 MB of data is indexed with 343 MB of Rtree .

Total RAM size = 4200 MB which is actually the largest possible cumulative size of Rtree that can be kept in Main memory of all slaves nodes together.

Thus , 4200 MB of Rtree would Index

= (3367*4200) / 343 MB of data ≈ **41 GB of Spatial data .**

Assumptions : This Approximate Calculation Valid only if Average size of tuple of 41 GB of data must be more or less equal to one Used to perform Experiment.

**If a Reducer (Or Mapper) attempts to consume the memory more than allotted to it , Master node Kills the task and attempt to run it on machine with larger memory . After several Trials , if the task still fails due to lack of required memory size , the job is Terminated by Hadoop Framework giving 'GC Overhead Limit Exceed' error.*

3.2.4 Shortcomings and Issues

Like SJMR (section 3.1) , Constructing an Rtree might also suffer from the same two shortcomings as SJMR did. Both MR algorithms have exactly similar Map-Phase to distribute the data objects among partitions. First, those objects whose MBRs overlap with two or more partitions or tiles , get copied to multiple partitions and therefore its multiple entries co-exist in two or more Rtree chunks created on different slave machines. This causes redundancy in the Rtree. Secondly, to ensure uniform distribution of objects among partitions, we perform tile based partitioning of 2D space which splitted up the partition area into potentially smaller partitions(tiles). This increases the chance of geometry of data object to overlap across two or more tiles, and hence object get copied to multiple partitions, whereas it might be the case that it was overlapping with just one partition before tile based partitioning was performed. Hence, performing Tile Based partitioning to obtain uniform distribution of objects among partitions, also increases the Number of Duplicate entries the Rtree that will be formed would have.

To address these issues , we can keep the track on the no of objects that has been mapped to each partition in the Mapper phase. Every partition is associated a **count** with it which denotes the no of data objects that has been copies to that partitions at that point of time. Now if the case arise that some data objects need to get copied two or more partitions, we always choose only one partition with the **least count.** This would ensure that the object get mapped to exactly one partition and , smooth distribution of objects among partitions is also maintained. However , this approach won't rectify the shortcomings for SJMR.

3.3 Dispersed Spatial Operation

This is the class of Spatial operation which when implemented in Map-Reduce form cannot be solve in one Map-Reduce Stage. Therefore Map-Reduce Implementations of these operations require to have more than one Map-Reduce Stages in which the output of Reduce phase serves as the input to the Mapper phase of subsequent Map-Reduce iteration.

We shall discuss the Map-Reduce Implementation of **ANN(ALL nearest neighbor problem)** which belongs to this class of spatial operation. ANN problem requires three Map-Reduce Phases to solve the problem.

3.3.1 ANN Problem – Map-Reduce Three Stage Algorithm

Given Two Data sets, **say R & S** , We need to find nearest neighbor of each object in R , in S . Two Map-Reduce Stages are required to Solve this problem.

Challenge:

In this Figure 3.5 , the nearest neighbor of **R1** in its own Partition is **S3**, called **tentative NN** which is easy to compute in map-reduce as both the are elements on same partition(hence available to same Reducers).

But Actual NN of **R1** is **S4** which lies in another Partition . Clearly , Reducer doing Computation to find NN in partition 1 do not have the access to any object belonging to Another partition .

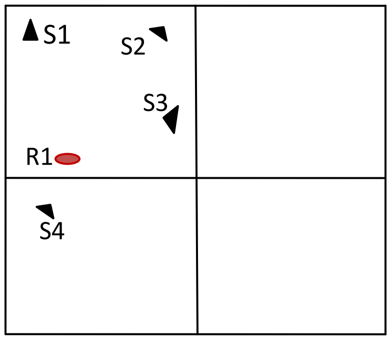


Fig 3.5

Map & Reduce processes executes in Isolation with respect to each other . There is no shared memory/variable Or network Communication amongst reducers (or Mappers Or between Mappers & Reducers) while **they are in the state of execution (**reducers fetches the mappers output prior to execution**).** So the challenge is how would the Reducer process processing the partition elements carrying R1(in the fig above) know that there exist another object element of **S ,** S4 , which lies in some other adjacent partition (which is being processed by another reducer process) which is closer to R1 than S3 is in terms of 2D Euclidian distance. To Solve this problem , we use intermediate data structure called **Pending Files.**

Key points:

- We Use two-directional Plane Sweep Line Algorithm to determine the nearest neighbor of object **'r'** **,** the element of dataset R, within its own partition space .
- When the tentative Nearest neighbor of object r is found within its own partition, then we compare the Distance between r and NN(r) with other three walls of the partition.
- We maintain another data Structure called Pending files which store pending Objects**.** Pending objects are elements of dataset R whose NN is not guaranteed to exist within its own partition after first Map-Reduce pass.

In the fig 3.6, let we come to find that **s** is the NN of r whose distance from r is **d.** Now if 'd' is less than each of **x, y & z** which is the distance of r from walls of the partition0, then s is the true NN of r.

However, if distance of any of the walls of partition0 is less than d, say **y < d** for example, than **s** become the tentative NN of r because there can potentially exist another object of dataset **S** in partition1 whose Euclidian distance from r is lesser than d.

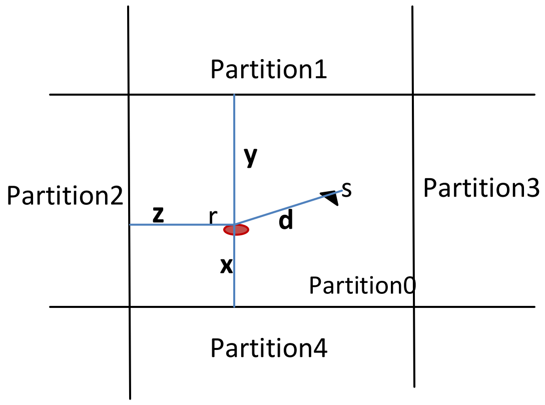


Fig 3.6

Here is the **pseudo code** that capture the above scenario:

```
If (d < y & d < z & d < x) then
        s is the NN(r) and return
Else
   Add the following information in pending files
        1. mbr of object r (pending object) and its OID
        2. Distance of r from its NN found (d)
        3. Pending partition no.(partition1)
```

We now describe input/output key value pairs of Map-phase & Reduce-Phase of the Algorithm:

STAGE 1 : : Implement two Direction Sweep Line Algorithm to compute **newNN** of every pending object 'r' . Add pending objects into pending files.

Map-Phase : As usual , in map phase we devide the spatial data R & S in isolated partitions.

INPUT : <key , value> = <Immaterial> , <Object tuple >
OUTPUT : <key , value> = <Partition no> , <Objects MBR +Object_OID >

Reduce-Phase : Implement two Direction SweepLine Algorithm to compute NN of every r .

INPUT : <key , value> = <Partition no> , LIST <Objects MBR +Object_OID >
OUTPUT : <key , value> = <r.OID> , <NN(r).OID >

In case if r becomes the pending object , then reducer must write pending information to the Pending File . Do not output for pending 'r'.

output of this stage : True NN of most objects are found , while whose NN wasn't guaranteed are written to pending files.

STAGE 2 :

Map-Phase : The input to the Map-Phase are the tuples from pending files which was generated In previous reduce –phase and data set S.

INPUT **:** <key , value> = <Immaterial> , <tuple p>

If p is from S , distribute it among partitions as usual ('d' would have no meaning in this case). If p is from pending file , place 'r' in the pending partition only.

OUTPUT **:** <key , value> = <Partition no> , <Objects_MBR +Object_OID +d >

Reduce-Phase : Implement two Direction Sweep Line Algorithm to compute **newNN** of every pending object r .

INPUT **:** < key , value> = < Partition no> , LIST < Objects_MBR + Object_OID + d >

```
If distance (r , newNN (r)) < d then
  OUTPUT: < key , value> = <r.OID> , <newNN(r).OID + new d>
Else
  OUTPUT: < key , value> = <r.OID> , <oldNN(r).OID + d>
```

output of this stage : pending objects too have found the set of candidates **(exactly one reported by one reducing process)** exactly one among which is guaranteed to be true neighbor.

STAGE 3: This stage is to find out the trueNN from among all candidates claiming to be trueNN of r as reported by different reducer process of Stage2. Obviously trueNN will be the one with the least distance d from 'r'.

Map-Phase : No processing in this phase , IDENTITY MAPPER, just read the data from HDFS and make it available for the reducer process after Aggregation.

INPUT**:** < key , value> = <r.OID> , <oldNN(r).OID/newNN(r).OID + d>
OUTPUT: SAME AS INPUT

Reduce-Phase: Out of the several potential candidates claiming as trueNN of object 'r' as reported by reducers of stage 2, find one which actually is . This will be the one with the least 'd' , the distance between itself and object r.

INPUT **:** < key , value> = <r.OID> , LIST <oldNN(r).OID/newNN(r).OID + d>
OUTPUT: <r.OID> , <oldNN(r).OID/newNN(r).OID>

3.3.2 Analysis & Conclusion

In this Solution approach , we use the Intermediate data Structure called **pending files.** Pending files were used to communicate the information that was generated in the current MR Stage to the following MR Stages. In this problem, pending files carries the list of object whose NN weren't guaranteed to have found in first MR Stage.

Generalizing the solution strategy, the need to have intermediate data structure such as pending files would arise if a Reducer needs to process the data tuple (with the data tuple of its own partition it is processing) that actually being processed by another Reducer. Since Reducer executes in isolation with respect to each other (i.e. there is no communication), Hadoop Platform provides no way to carry out any communication between Reducers (Or Mappers) while they are in the state of execution.

3.4 Massive Graph Computation

Graphs are ubiquitous in modern society: examples encountered by almost everyone on a daily basis include the hyperlink structure of the web (simply known as the web graph), social networks (manifest in the flow of email, phone call patterns, connections on social networking sites, etc.), . Transportation networks (roads, bus routes, flights, etc.) are often modelled as basic graph problems and addressed within the scope of Geographic Information System. This chapter focuses on graph algorithms in MapReduce.

3.4.1 Real world scenarios generating massive graphs

By Massive Graphs , we mean the graph consisting of millions of nodes and edges such that the size of the graph ranges to hundreds of Gigabytes that cannot be fit into Main Memory of a Single Machine. The Real World Scenarios where Massive Graphs are Generated are **World Wide Web Graphs** in which the node represents the web page , and its neighbors represent the web pages the former contains the hyperlink of. Another Example could be the **Facebook social network graph**. It has more than 500 million users(and growing) and an average user has 130 friends . This means that a graph data structure would need at least 500x106x130x8bytes = 520 GB space .

Social Networking sites such as facebook Process such large graphs to come up with useful statistics which help them to improve their Services. Its obvious such large graphs can't be processed on a single machine . We Shall Discuss the Map-Reduce Strategy to perform Graph manipulation such as SSSP , BFS , DFS . One Common property the WWW graphs and Social Network Graphs has is that they have upper bound on the Max degree of any node. The reason being that there are at most 50-60 hyperlinks that even a complex web-page might have. Similarly, Social networking sites put the upper limit on the no. of users one can add in his friend-list (for facebook , it is 5000). Such graphs are called large small-world network graphs [15].

3.4.2 Map-Reduce Pipelining -- The iterative Map-Reduce Strategy

Graph Representation: While manipulating graph with Map-Reduce, it is important to choose the most optimum representation of graph first since the Complexity of the Map-Reduce algorithm that would going to manipulate the graph largely depends on the manner the graph has been represented. In Map-Reduce paradigm, it is best to represent the graph in **Adjacency list form.**

Challenges: While developing the map-reduce algorithm to solve the graph problem, the key challenge is to come-up is to decide the proper mechanism to partition the graph among slave nodes so that they can process their dedicated graph chunk independently without the knowledge about the rest of the graph. Remember, Hadoop framework is based on shared-nothing Architecture.

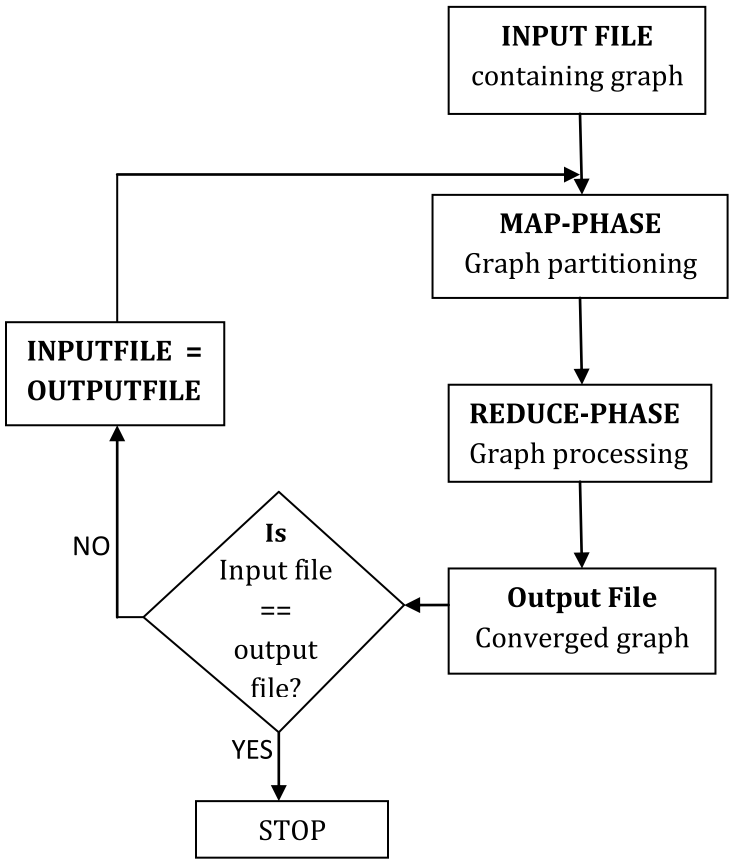


Fig 3.7 Flowchart showing the general strategy for Graph processing via Map-Reduce

Pipelining Nature: Generally the Map-Reduce Algorithms to manipulate graphs are iterative in Nature i.e. the problem is solved by processing the graph through a Map-Reduce pipeline, each iteration being Converging towards a Solution. Flowchart in fig 3.7 captures the basic idea of manipulating graphs via Map-Reduce. We continue to iterate as long as two successive Map-Reduce iterations outputs identical graphs.

3.4.3 Single Source Shortest path SSSP via MAP-REDUCE

In this Section we explain the Map-Reduce form of **Single Source Shortest Path** Algorithm. We shall Explain with the help of example. Consider the graph shown in the figure 3.8 below:

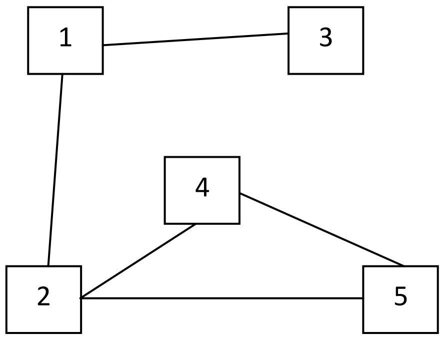


Fig 3.8

1 0 2:3: father -1

2 10000 1:4:5: father -1

3 10000 1: father -1

4 10000 2:5: father -1

5 10000 2:4: father -1

INPUT FILE FORMAT

The Graph is represented as Adjacency List . '1' is the Source Node . Second field represent the distance of a node (first field) from Source (i.e. node 1) . Initially all nodes are at infinite distance from source. '10000' is some large number we have chosen to represent infinity. Nodes separated by colons are the neighbors of node with which the line begins (see input file format).

For example last line represents that node 2 & 4 are neighbors of node 5 and node 5 is at infinite distance from Source (because of 10000). **-1** is the father of node 5 (Initialisation) in shortest path tree that will be formed. Last field will construct the shortest path tree.

ALGORITHM:

Map-Phase : For each node.neighbor , increment the second field and make the node (first field) as its father .

Example :

```
1          0          2:3:      father    -1
```
(Ist line of Input file)

Node.neighbor = {2,3} , where Node is 1 , output the flowing lines

```
2          1          father   1
3          1          father   1
```

This line represents that node 3 is at 1 unit distance away from Source node '1' , and its father is node '1' in the shortest path tree that has formed so far.

Now output two more lines corresponding to the line of input file just processed as :

1 0 father -1 (this will make the algorithm to work for disconnected graphs as well)

1 NODES 2:3:

Our objective is to replace the word "NODES" (which acts as a **flag**) with minimum distance of node '1' from Source (in this case 1 is itself a source.) This will be done in Reduce phase .

Thus, in Map phase, the Ist line of the input graph generate 4 lines put together below:

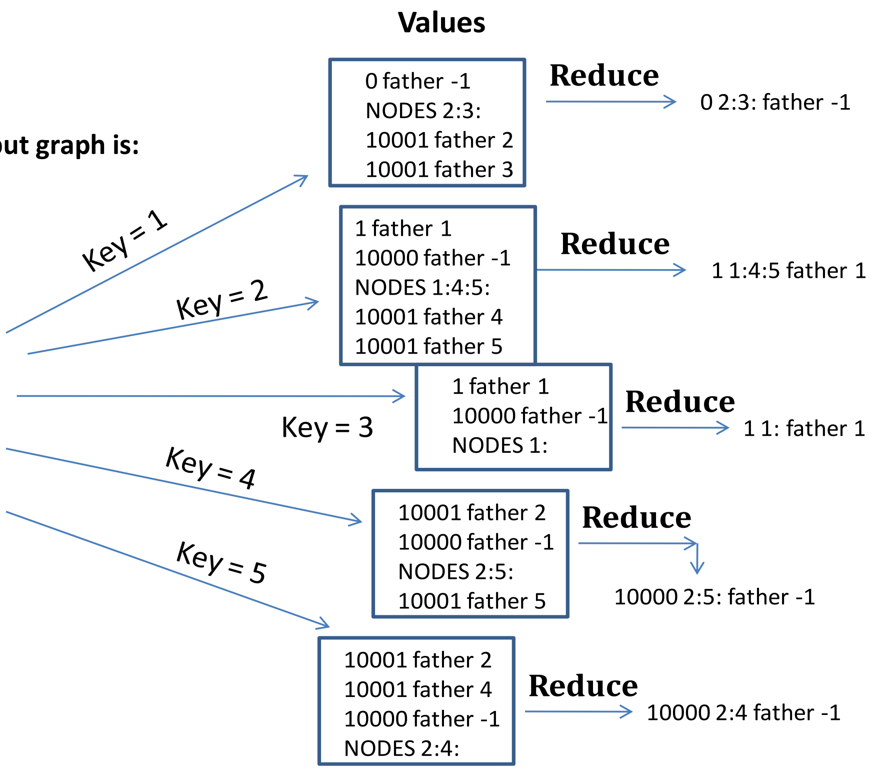
$$\boxed{1 \quad 0 \quad 2{:}3{:} \quad \text{father} \quad -1} \xrightarrow{\text{Mapper}}$$

```
2  1 father 1
3  1 father 1
1  0 father -1
1 NODES 2:3:
```

Input line **output lines generated**

Thus , mapout1 file for the input graph is:

```
2  1 father 1
3  1 father 1
1  0 father -1
1 NODES 2:3:
1  10001 father 2
4  10001 father 2
5  10001 father 2
2  10000 father -1
2 NODES 1:4:5:
1  10001 father 3
3  10000 father -1
3 NODES 1:
2  10001 father 4
5  10001 father 4
4  10000 father -1
4 NODES 2:5:
2  10001 father 5
4  10001 father 5
5  10000 father -1
5 NODES 2:4:
```



All the lines starting with node no ‘**x**’ in mapout1 , are taken by reducer process with key = x and processed. Since there are 5 nodes in all , 5 reducer processes are generated.

Reduce-phase: The Reducer process will collect all lines from mapper output corresponding to particular key value K, a determine the shortest distance of node K is from Source node so far.

Thus, output graph after first complete Map-Reduce iteration we obtain is :

1	0	2:3:	father	-1
2	1	1:4:5	father	1
3	1	1:	father	1
4	10000	2:5:	father	-1
5	10000	2:4	father	-1

Here , we see that the node 2 & 3 has Converged from Infinity to source (node 1), where as node 4 & node 5 hasn’t .

Node 4 & 5 will converge in Subsequent Iterations**.**

3.4.4 Analysis & Conclusion

The MR Strategy that we explain does not build any data structure whatsoever either in Mapper-phase or Reducer-phase. In both the phases, Computations are carried out on the input line of text as soon as it arrives. Thus, this algorithm enjoys the benefit of not being bound by the main memory size of the cluster. In fact, implementing above strategy (not necessarily in Hadoop environment, just a simple program that read- write to or from files) on a single machine can be used to process the graph of hundreds of gigabytes of size, containing millions or trillions of nodes and edges, provided that you have sufficient disk space and patience to read & write the entire graph per map-reduce iteration.

But the Question arises how many Iterations does it take to complete process the entire graph by this MR algorithm?

Answer is, it take as many iterations as the **Diameter** of the graph is. The **diameter** of a graph is the maximum eccentricity of any vertex in the graph. That is, it is the greatest distance between any pair of vertices. To find the diameter of a graph, first find the shortest path between each pair of vertices. The greatest length of any of these paths is the diameter of the graph.

Now if one process any arbitrary massive graph via this MR approach, the job may go on processing the graph through tens of thousands of iterations as diameter of the graph can be expected to be large. But thankfully, in real World Scenarios, such as WWW and Social Network Graphs, graphs have reasonably small diameter, therefore they are called **large small world Graphs** and thus can be processed via MR approach effectively with reasonable no of Map-Reduce Iterations.

4. Conclusion & Future Work

4.1 Conclusion

In our work, we present a performance evaluation & Map-Reduce implementation details of some complex spatial operations on Hadoop platform. The results demonstrate the feasibility and efficiency of the Map Reduce model and show that small scale cluster has the potential to be applicable to computationally intensive spatial data computations. We showed that executing Map-Reduce version of a spatial Query Involving Spatial join on a small scale cluster clearly outperforms that of single spatial databases in terms of time taken to output desired results. Real world Massive graphs, which can't be manipulated by traditional sequential algorithms on a single machine, can be efficiently processed on a small scale cluster because of their small world property. The Comparative study of Hadoop and Distributed database systems shows that the two technologies are rather complementary with each other and neither of the two is good at what other does.

4.2 Future Work

Problem Formulation for MTP STAGE 2:

Most of the architectural differences discussed in section 1.4 are the result of the different focuses of the two classes of system. Neither is good at what the other does well. Hence, the two technologies are complementary. Hadoop is scalable and exhibit excellent fault tolerance capability. DBMS, on the other hand, outperforms Hadoop in terms of performance of structured data analysis. We expect complex Spatial operations/problems would require the capabilities provided by both systems in the near future so that we could yield both, the performance while processing complex spatial SQL Queries (property of DBMSs) and could process thousands of terabytes of spatial data by scaling the DBMS cluster size to the extent of thousands of nodes while maintaining a good degree of fault tolerance (property of Map-Reduce). ***This requirement motivates the need to build the interfacing between MR systems and DBMSs that allow each system to do what it is good at****. We Expect the result is a much more efficient overall system than if one tries to run the entire spatial application in either system****. In Other Words, we shall try to bring the parallelization power of Map-Reduce to real spatial processing environments – the Spatial DBMSs.***

5. References

[1] T. White, Hadoop: The Definitive Guide. O'Reilly Media, Yahoo! Press, June 5, 2009.

[2] A. Bialecki, M. Cafarella, D. Cutting, and O. O'Malley, "Hadoop: a framework for running applications on large clusters built of commodity hardware," Wiki at http://lucene.apache.org/hadoop.

[3] Christian Vecchiola , Suraj Pandey , Rajkumar Buyya, High-Performance Cloud Computing: A View of Scientific Applications, Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks, p.4-16

[4] Bruce Momjian , postgreSQL: Introduction and concepts. Pearson Education Corporates Sales Division , 2001

[5] H. Karlff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In Proc. 20th SODA,2010

[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6. San Francisco, CA: USENIX Association, 2004, pp. 10-10.

[7] Yonggang Wang , Sheng Wang "Research and Implementation on Spatial Data Storage and Operation", 2010 , Second lITA International Conference on Geoscience and Remote Sensing, pp. 275-278

[8] Zhang, S., Han, J., Liu, Z., Wang, K., and Xu, Z. SJMR: Parallelizing spatial join with MapReduce on clusters. In Proceedings of CLUSTER. 2009, pp. 1-8.

[9] J.P. Dittrich and B. Seeger, "Data redundancy and duplicate detection in spatial join processing," in ICDE '00: Proceedings of the 16th International Conference on Data Engineering , 2000, pp. 535–546

[10] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, Sergei Vassilvitskii , "Filtering: a method for solving graph problems in MapReduce.", SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 85-94.

[11] K. Wang, J. Han, B. Tu, J. Dai, W. Zhou, and X. Song, "Accelerating Spatial Data Processing with MapReduce"*, in Proc. ICPADS, 2010, pp.229-236*

[12*] Jun Zhang, N. Mamoulis, D. Papadias, and Yufei Tao,"All-nearest-neighbors queries in spatial databases," June 2004, pp.297–306.*

[13] Haojun Liao, Jizhong Han, Jinyun Fang, "Multi-dimensional Index on Hadoop Distributed File System," Fifth International Conference on Networking, Architecture, and Storage, 2010 , pp.240-249

[14] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in Proceedings of the ACM SIGMOD . Boston, Massachusetts: ACM, 1984, pp. 47–57.

[15] Halim, F.; Yap, R.H.C.; Yongzheng Wu; , "A MapReduce-Based Maximum-Flow Algorithm for Large Small-World Network Graphs," Distributed Computing Systems (ICDCS), 2011 31st International Conference on , 20-24 June 2011, pp.192-202

[16] http://www.dbms2.com/2009/04/30/ebays-two-enormous-data-warehouses/

[17] *Pavlo, a., Paulson, e., rasin, a., abadi, d.J., deWitt, d.J., Madden, s.r., and stonebraker, M.A "comparison* of approaches to large-scale data analysis". In Proceedings of the 35th SIGMOD International Conference on Management of Data. aCM Press, new york, 2009, pp.165–178.

[18] *Abouzeid, a., bajda-Pawlikowski, K., abadi, d.J., silberschatz, a., and rasin, a. Hadoopdb: an architectural hybrid of Map-reduce and DBMS technologies for analytical workloads. In Proceedings of the Conference on Very Large Databases, 2009.*