Thesis for the Degree of Master of Engineering

Enhanced R-Tree Bulk Loading Scheme Using Map-Reduce Framework

by

Huynh Cong Viet Ngu

Department of IT Convergence and Application Engineering

The Graduate School

Pukyong National University

February 2017

Enhanced R-Tree Bulk Loading Scheme Using Map-Reduce Framework

맵리듀스 프레임워크를 이용한 향상된 R-트리 벌크로딩 기법

Advisor: Prof. Ha-Joo Song

by

Huynh Cong Viet Ngu

A thesis submitted in partial fulfillment of the requirements

for the degree of

Master of Engineering

in Department of IT Convergence and Application Engineering,

The Graduate School,

Pukyong National University

February 2017

Enhanced R-Tree Bulk Loading Scheme Using Map-Reduce Framework

February 2017

주 심	공학박사	신 봉 기	(인)	[illegible]
위 원	공학박사	권 오 흠	(인)	[illegible]
위 원	공학박사	송 하 주	(인)	[illegible]

Contents

List of Figures

Enhanced R-tree Bulk Loading Scheme

Using Map-Reduce Framework

Huynh Cong Viet Ngu

Department of IT Convergence and Application Engineering,

The Graduate School, Pukyong National University

Abstract

An R-tree is an index structure that enables fast accesses to multi-dimensional data. Constructing an R-tree for a given data set yields an efficient R-tree structure than incrementally building one as data are inserted. However it usually takes a lot of time constructing an R-tree for a huge volume of data. In this paper, we propose a parallel R-Tree construction scheme based on a Hadoop framework. The proposed scheme divides the data into partitions, builds local R-trees in parallel, and merges them to construct an R-tree that covers whole data set. While generating the partitions, it considers the data distribution so that each partitions have nearly equal amount of data. Therefore the proposed scheme gives an efficient index structure while reducing the construction time

I. Introduction

Nowadays, aside from the fact large amounts of traditional data are still increasing significantly, there is an explosion in the amount of spatial data that is being produced from many devices such as satellites or smart phones. In order to handle this amount of spatial data efficiently, the R-tree is considered as an optimal index mechanism that will help retrieve data quickly according to its spatial locations. An R-tree [2] is a data structure in which each node contains a certain number of index entries, each of which consists of a Minimum Bounding Rectangle (MBR) and the pointer to an object or its child node if it is not a leaf node. Both objects and non-leaf nodes are always represented by MBR. A layout of MBRs of objects is shown in Figure 1 and Figure 2 show the R-tree structure with four nodes.

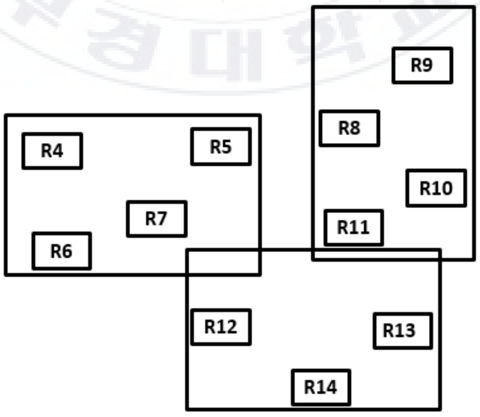


Figure 1. A layout of MBRs

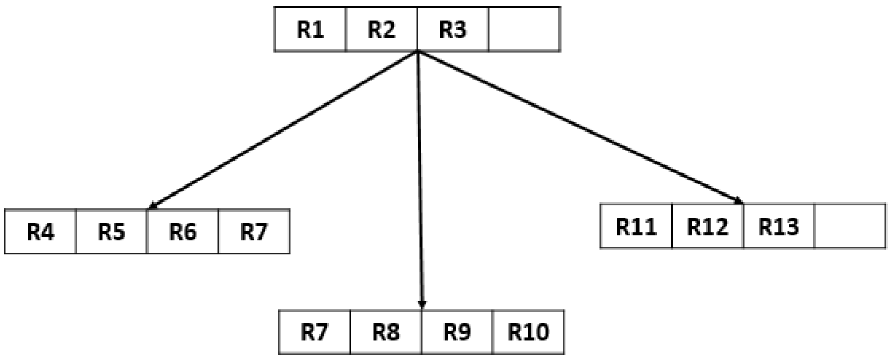


Figure 2. The Corresponding R-tree with Four Nodes

An R-tree, is built by inserting new items iteratively as they arrive [2]. But with this method, when an object is inserted into an R-tree node, in some cases of the node splitting operation, it requires the locking of the R-tree, therefore the concurrent insertions are prevented.

In fact, in some applications where all the items is available as with the Geometry Information System (GIS) problems, if an R-tree is built using the packing technique in a parallel way, it’s much faster than the traditional method as mentioned previously.

Since its release in April 2005, Hadoop [3] was adopted as an optimal solution for scalable processing of huge datasets in many applications, e.g., machine learning, image processing, web crawling or text processing, and so on. Hadoop employs MapReduce [4], a simplified programming paradigm for distributed processing, to build an efficient large-scale data processing framework.

An R-tree is an index structure that enables fast accesses to multi-dimensional data. Constructing an R-tree for a given data set yields an efficient R-tree structure that builds itself incrementally as data are inserted. However, it usually takes a lot of time to construct an R-tree for a huge volume of data. In this paper, I propose a parallel R-Tree construction scheme based on a Hadoop framework. The proposed scheme divides the data into partitions, builds local R-trees in parallel, and merges them to construct an R-tree that covers a whole data set. While generating the partitions, it considers the data distribution so that each partition has nearly equal amounts of data. Therefore the proposed scheme gives an efficient index structure while reducing the construction time.

The remainder of the thesis is organized as follows. Chapter II give a brief description about Hadoop, MapReduce framework and related algorithms for parallel R-tree construction. Chapter III describes my method to build parallel R-tree on Hadoop environment with MapReduce model in detail. Chapter IV gives the experimental results. And the last chapter, chapter V is the conclusion of the thesis.

II. Related Works

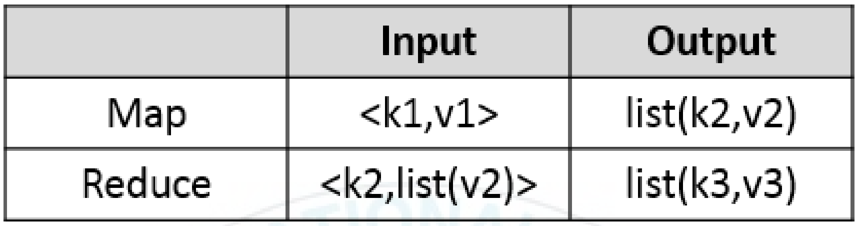
2.1 Big Data

2.1.1 Hadoop - MapReduce

A few years ago, to store or process data, most enterprises had a super computer to perform this task. Here data can be stored in an RDBMS such as Oracle Database, MS SQL Server or DB2. After that, the software can be written to interact with the database, then send to user for analysis purpose.

But with this approach, when it has to handle huge amounts of data, it faces many difficulties in processing such data through a traditional database server. Facing those difficulties, in 2005, an Open Source Project called Hadoop was released.

In order to handle a huge amounts of data, Hadoop runs all applications using the MapReduce algorithm, where the data is processed in the parallel way on different nodes. MapReduce is a programming model suited for parallel computation, it handles parallelism, fault tolerance and other level issues. Furthermore, MapReduce consists of both a map and reduce function which are user-defined. The input data format is specified by the user and the output is a set of <key,value> pairs. As shown in (Figure.3), the mapper applies user-defined logic on every input key/value pair (k1,v1) and transforms it into a list of intermediate key/value pairs(k2,v2). Then the reducer will apply user-defined logic to all intermediate values (v2) associated with the same k2 and produces a list of final output key/value pair (k3,v3). The data flow of the MapReduce framework is illustrated in Figure 4.



	Input	Output
Map	<k1,v1>	list(k2,v2)
Reduce	<k2,list(v2)>	list(k3,v3)

Figure 3. Input and Output in MapReduce

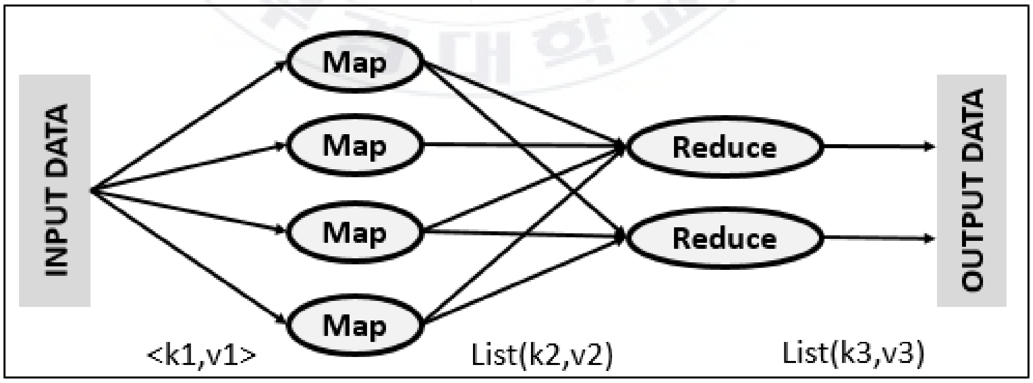


Figure 4. MapReduce Framework

2.2 Quality of R-tree

As I described in Chapter One, an R-tree is built by inserting new items iteratively as they arrive. Each insert operation of a polygonal object represented by its (MBR), first follows one path from root down to a leaf node, then the new item is added to the leaf node. If the leaf node does not have enough space for the new item, it will be split into two nodes.

Similar to all the other data structures, the goal is how to build them so that search performance is the best. In general, there are two primary goals for R-tree construction, a main consideration is how to minimize the area of the MBRs of the non-leaf nodes that are not covered by MBRs at the leaves these enclose, which is also called the “Dead Area”, this goal can improve search performance since decisions on which paths have to be traversed can be taken on higher levels. A second consideration is how to minimize the overlap between MBRs, this goal is also designed to decrease the number of paths to be traversed.

Figure 5 illustrates how to split R-tree node into new two nodes, in which, although in “Bad Split” case, there is no overlapping area, but the total area of the covering rectangles in the “Good Split” case is much less than in the “Bad Split” case.

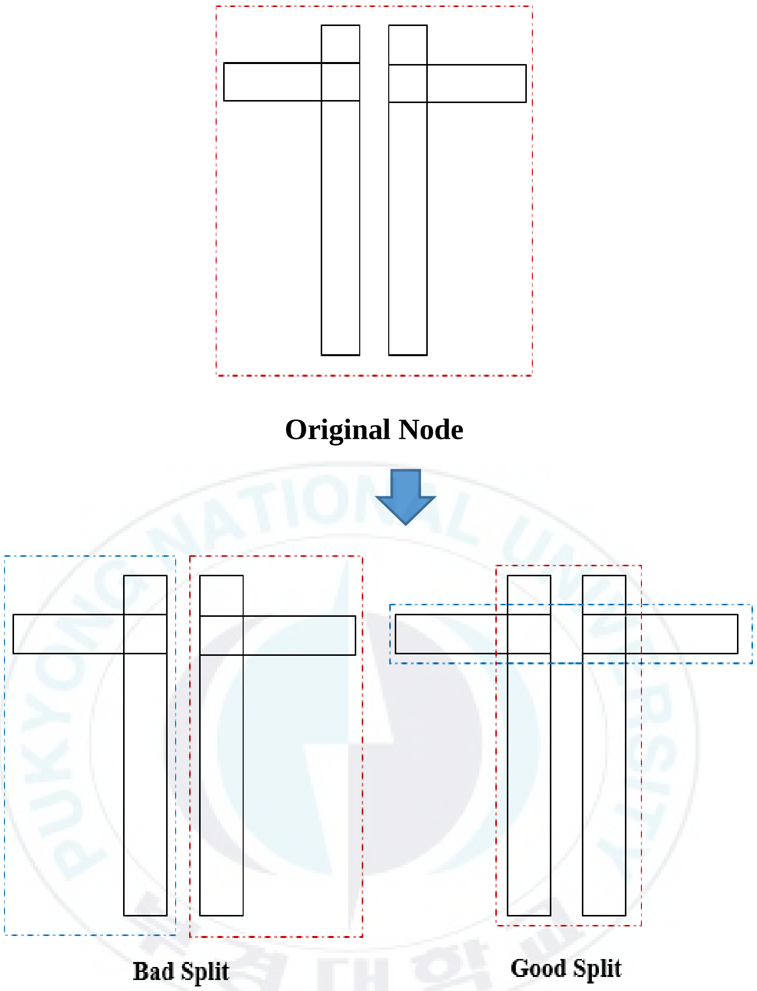


Figure 5. R-tree Splitting Operation

But with this approach, in case of some applications in which all the items are available such as the GIS problem, it will take a lot of time for R-tree construction. With applying the Bulk-loading methods such as Z-order curve or Hilbert curve for R-tree construction build fast an R-tree with maximum node occupancy (thus, the R-tree's height is minimal), besides that, the area of Minimum Bounding

Rectangles (MBRs) that cover the non-leaf node and the overlapping area between them also is minimal.

2.3 Parallel R-tree Construction using MapReduce

For parallel R-tree construction, in [5], author have proposed a method for parallel R-tree construction using MapReduce model in Hadoop environment, it is performed in a bottom-up fashion and has three phases, in which the first two phases are executed in MapReduce framework, while the last phase is executed outside cluster because it does not require the high computational, as shown in figure 6.

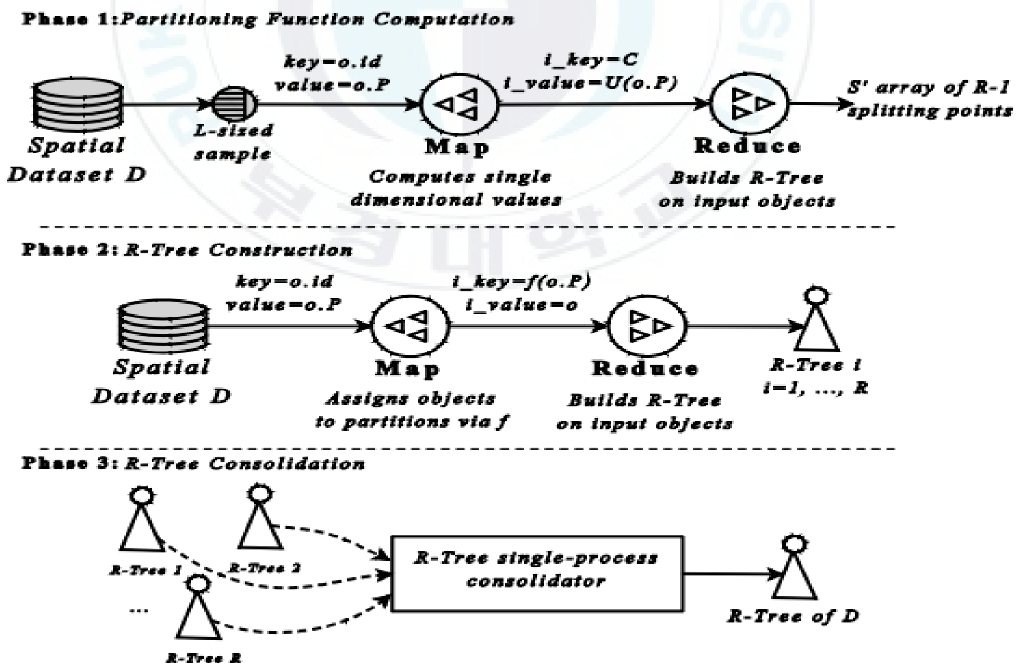

Phase 1: Partitioning Function Computation
key=o.id
value=o.P
i_key=C
i_value=U(o.P)
S' array of R-1 splitting points
Spatial Dataset D
L-sized sample
Map
Computes single dimensional values
Reduce
Builds R-Tree on input objects
Phase 2: R-Tree Construction
key=o.id
value=o.P
i_key=f(o.P)
i_value=o
Spatial Dataset D
Map
Assigns objects to partitions via f
Reduce
Builds R-Tree on input objects
R-Tree i
i=1, ..., R
Phase 3: R-Tree Consolidation
R-Tree 1
R-Tree 2
...
R-Tree R
R-Tree single-process consolidator
R-Tree of D

Figure 6. R-tree Construction Scheme is Proposed by A.Cary (6).

Purpose of the first phase, which is also the most important phase of this scheme is to assign the objects to a pre-defined number of R partitions. In order to assign the objects so that the “Dead Area” in resulting of Minimum Bound Rectangles (MBRs) and overlap between these MBRs can be decreased, Z-order curve is used as Geo-packing technique to grouping spatially neighboring objects, it is used for mapping multi-dimensional spaces into an ordered sequence of one-dimensional values via space filling curves.

The spatial objects are mapped onto Z-order curve by compute Morton number for each of the objects, then those Morton numbers are sorted into a list and the splitting points that split the list into **R** equal-sized partitions will be determined.

In second phase, all objects are divided into R partitions, then **R** independent “small” R-trees are built concurrently on their data. Output of this phase is a set of “small” R-trees.

In the last phase, **R** individual “small” R-trees will be combined under a single root node to create the final R-tree.

For short, in the rest of thesis, I call this method is Z-curve method.

2.4 R-tree Packing Algorithm

There are many kinds of packing techniques for R-trees are proposed before, in this section, two primary packing algorithms for R-tree construction will be described, including Z-order curve and Sort-Tile-Recursive algorithms. Instead of using an area of rectangle, all of these algorithms only use the center point of rectangle for grouping. The only difference is how the rectangles are ordered at each level.

2.4.1 Z-order Curve

The Z-order curve algorithm [6] orders the rectangle using space filling curve. The Z-value (Morton number) of a point is calculated by interleaving the binary representations of its coordinate values.

For each rectangle that cover an object, Z-value is calculated based on its center point, then the data rectangles are sorted into a list on ascending Z-values, this determines the order in which the rectangles are placed into the partitions. Figure 7 show the Z-order curves of order 1, 2 and 3 and how to partition objects.

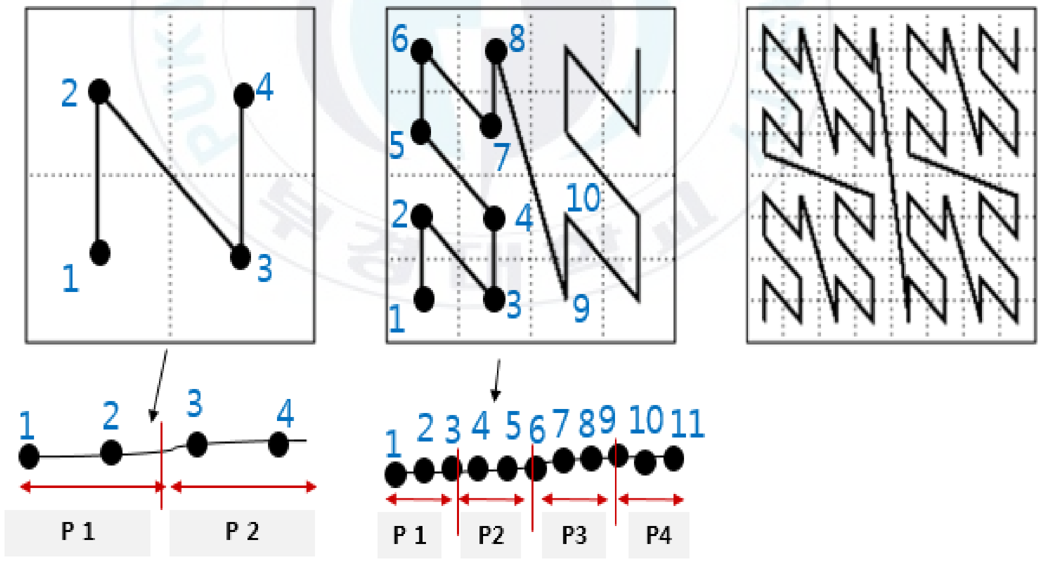


Figure 7. Z-order Curve of Order 1, 2 and 3

With Z-order curve packing technique, it still has some restrictions when in fact, almost the location of the objects are represented by the fraction number, thus, the accuracy in data partitioning is not high.

2.4.2 Sort-Tile-Recursive (STR)

If the previous algorithm groups rectangles by mapping multi-dimensional space to one-dimensional values and the accuracy in data partitioning is not high in case of the location of the objects are represented by the fraction number, in this section, I'm going to describe the Sort-Tile-Recursive algorithm [7], which is considered as one of the techniques not only a simplicity of implementation but also has a good query performance.

The basic idea of this algorithm is split the data space using vertical slices so that each slice contains nearly-equal rectangles. First, the rectangles are sorted by x-coordinate and partition the objects into pre-defined vertical slices. Each slice consists of the same rectangles from the sorted list, note that the last slice may contains the number of rectangles less than others. After the rectangles are sorted into the partitions, the rectangles in each slice are sorted by y-coordinate, then pack into nodes (example the first M rectangles are packed into first node, the next M into second node, and so on). Example of this algorithm are shown in figure 8.

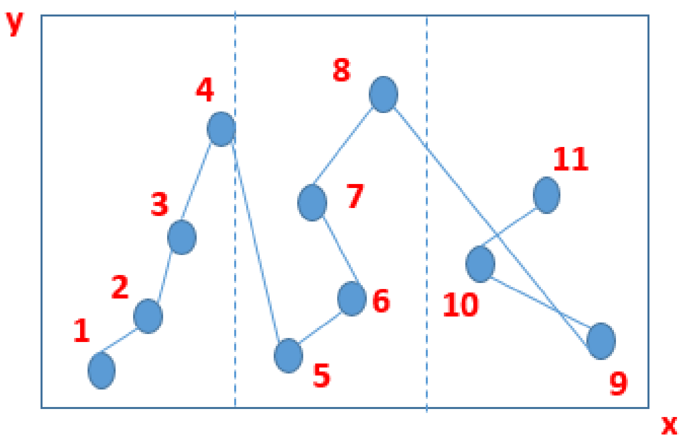


Figure 8. Sort-Tile-Recursive Algorithm

III. Parallel R-tree Construction using Hadoop

3.1 Overview

Before I describe in detail about our scheme, there are some notations that I will use in the rest of the paper are as follows: M is maximum node occupancy, r is the number of total spatial objects, and we use the coordinate of center point of objects for representing objects' location. For simplicity, in this thesis, I only concentrate on two-dimensional objects.

When building an R-tree, the quality of resulting R-tree is our main consideration. As discussed in the previous section, for a good R-tree, it should be built by grouping spatially neighboring MBRs, so the "Dead Area" and overlap between these MBRs can be decreased. Our parallel R-tree construction is performed in a bottom-up fashion and has four phases, and three of them are implemented in Hadoop environment with MapReduce model:

- Partitioning phase:
    - In this phase, instead of using Z-order curve that has been proposed by A.cary, I propose the new method that is inspired by the Sort-Tile-Recursive (STR) algorithm. However, instead of partition data by x-coordinate, firstly, we determine the "Longest coordinate" that has the two most distant centers between the objects, then partition data based on the "Longest coordinate". I named for my method is ISTR.(Improving of STR)
- R-tree Construction phase:
    - In this phase, independent small R-trees are built simultaneously.

- R-tree consolidation phase:
    - In this phase, merge small R-trees into the final R-tree.

Firstly, let us start our description by defining the problem. The data set that I am using is a CVS file where each line represents one object, it contains <o.id, o.P> where o.id is the object's unique identifier and o.P is the location of an object is represented by a list of coordinates.

The proposed scheme consists of three phases executed in sequence, as can be seen in Figure 9. First, I find out the "longest" coordinate that has the two most distant centers of the rectangles in the coordinate, then the number of partition and the partitions boundary will be determined. In the second phase, data is partitioned into the corresponding partition and create small R-Trees. Finally, the small R-Trees are merged into the final R-Tree. The first two phases are executed in MapReduce, while the last phase does not require high computational, so it is executed sequentially outside of the cluster.

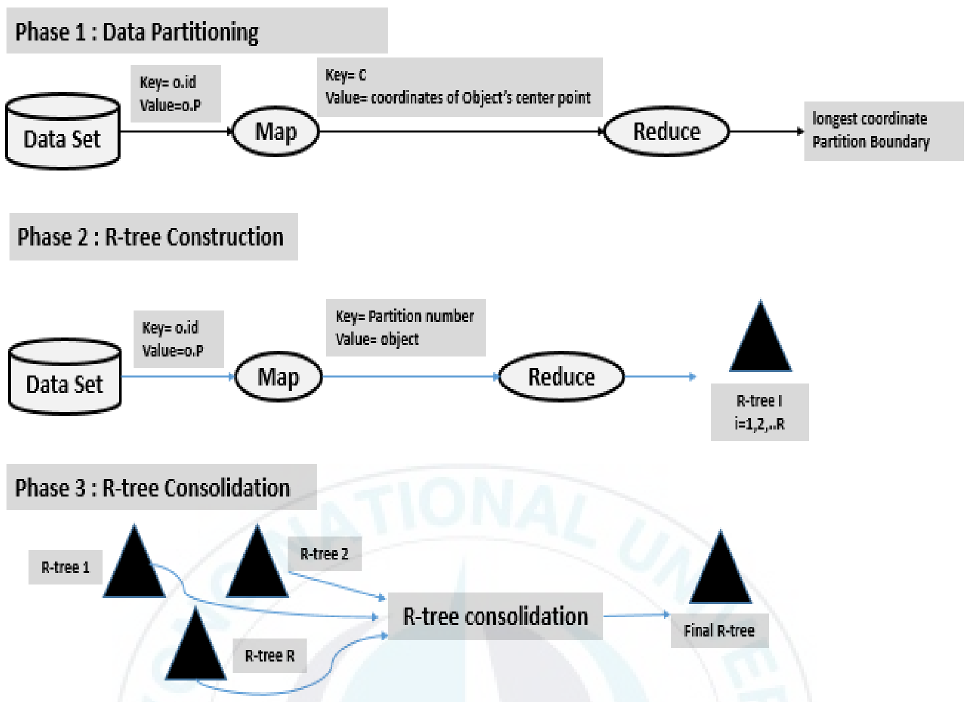


Figure 9. Our scheme for parallel R-tree construction in MapReduce framework

3.2 Data Partitioning

3.2.1 Description

As I discussed before, in this phase, instead of using Z-curve, I propose the method that is inspired by the Sort-Tile-Recursive (STR) algorithm. However, instead of partition data by x-coordinate, firstly, I determine the "Longest coordinate" that has the two most distant centers between the objects, then partition data based on the “Longest coordinate”. I named for our method is ISTR (Improving of STR).

First, we assume all objects lie in the plane, each object’ location is represented by a center point with its coordinate. To find out the longest coordinate that has the two most distant centers of the hyper- rectangle in the coordinate, our idea is to read random objects from the input file via data sampling with a default ratio of input data.

3.2.2 Proposed MapReduce Algorithm

The MapReduce algorithm runs M Mappers that take sample objects from the input file, then in each Mapper, it calculates the coordinates of center point of each object.

Then a single Reducer, firstly, it will calculate the distance between those objects and determine the longest coordinate as shown in Figure 10. In order to determine the “Longest coordinate”, I use two arrays, one to store x-coordinate of a center point of all objects that are sorted in ascending and one to store y-coordinates of a center point of all objects that are also sorted in ascending, the coordinate of a center points are implemented in Mapper. In order to determine the “Longest coordinate”, we calculate the distance between the start point and the end point in each of those arrays, then I compare and choose the larger one as the “Longest coordinate”, then data is partitioned based on the “Longest coordinate” array. After that, a new list K of R-1 partition boundary that split the longest coordinate of sample into R partitions so that each partition has nearly equal amounts of data is determined.

In Figure 10a, since distance between A and B is largest, x is determined to be a longest coordinates, and Figure 10b for the opposite case.

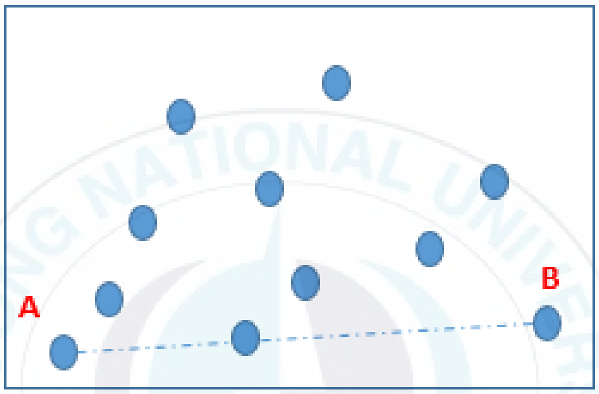


Figure 10a. X-coordinates will be used for data partitioning

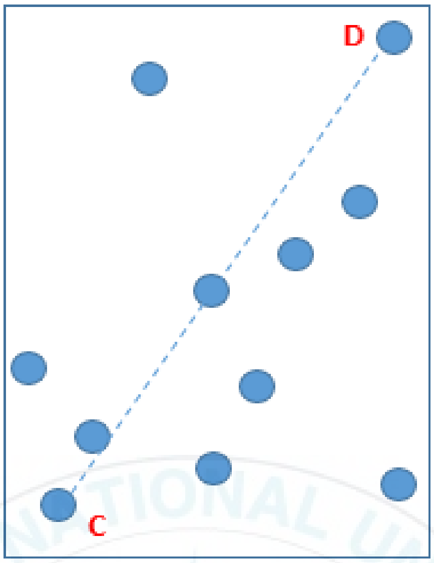


Figure 10b. Y-coordinates will be used for data partitioning

Figure 10. Longest Coordinate Determination

The specific MapReduce key/value input pairs are presented in Figure 11. Mappers read the default ratio of data from input file and calculates the coordinates of center point of the objects. The intermediate key is a constant that helps to send all the Mappers' outputs to a single Reducer. Then Reducer receives all center point with theirs coordinates from Mappers, firstly find out the longest coordinate by calculate the two most distant centers of the hyper- rectangle in the coordinate, then determine the list splitting point K base on the longest coordinate as shown in Figure 12.

Function	Input	Output(Key,Value)
Map	(o.id, o.P)	(C,center point)
Reduce	(C, list(center point))	K

Figure 11. Inputs/Outputs for Data Partitioning

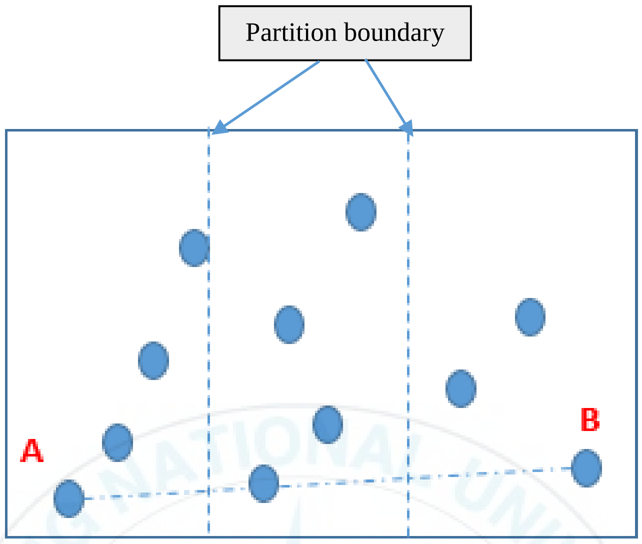


Figure 12a Data Grouping on X-coordinate

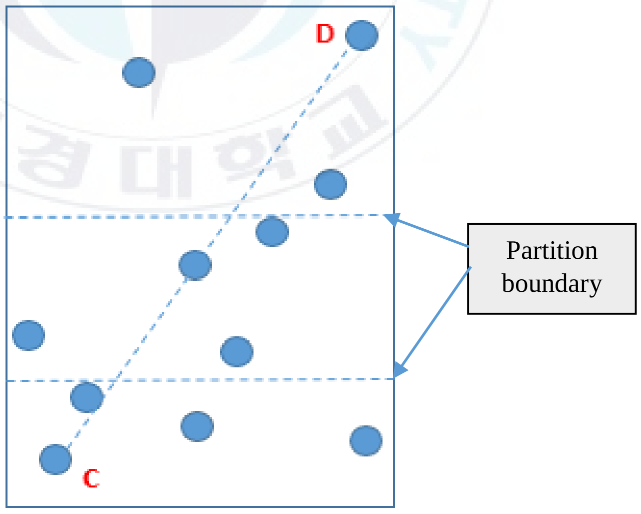


Figure 12b Data Grouping on Y-coordinate

Figure 12. Data Grouping into Partition with Maximum Objects equal 4

3.3 R-tree Construction

3.3.1 Description

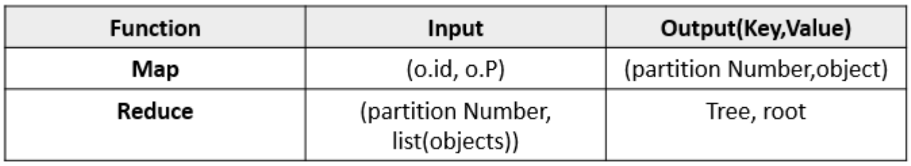
In this phase, independent “small” R-trees are built simultaneously.

3.3.2 Proposed MapReduce Algorithm

Mappers calculate the coordinates of center point of each object, then partition the objects into R groups, then every partition is executed by a different Reducer.

In each Reducer, a “small” R-tree is built independently using STR (Sort-Tile-Recursive) packing technique.

The output of every Reducer is a root node of their constructed R-Tree, as shown in Figure 13.



Function	Input	Output(Key,Value)
Map	(o.id, o.P)	(partition Number,object)
Reduce	(partition Number, list(objects))	Tree, root

Figure 13. Inputs/Outputs for R-tree Construction

3.4 R-tree Consolidation

In this phase, I am going to combine the R individual R-tree, built in the second phase, under a single root. Because it’s not computationally intensive and the logic to run this phase is fairly simple, it is executed outside the cluster as shown in Figure 14.

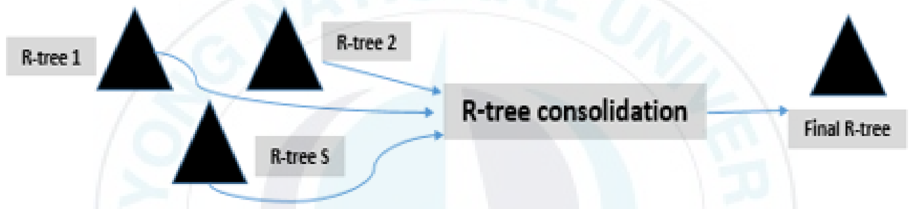


Figure 14. R-tree Consolidation

IV. Experimental Result

4.1 Experimental Environment

4.1.1 Hadoop Cluster

To implement the experiments, our cluster consists of eight machines, in which, each machine (Slave node) acted as a Tasktracker and Datanode, and one server (Master node) acted as Jobtracker and Namenode as shown in Figure 15.

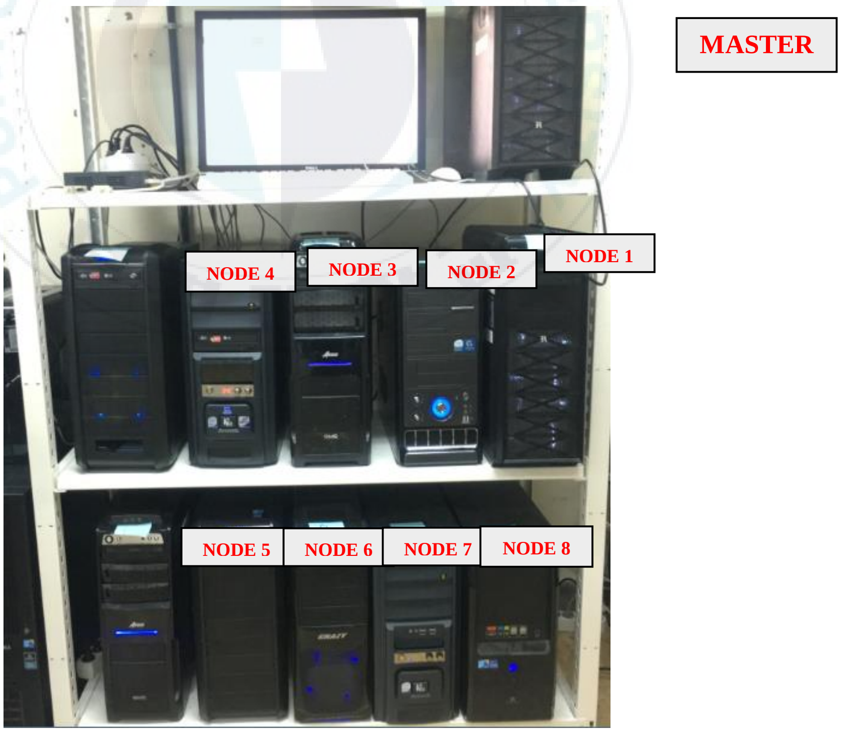


Figure 15 Hadoop Cluster

All machines were installed Centos 7.0 Operating system. All our experiments were performed with the latest Hadoop version (2.7.2) running on all machines in the cluster.

Besides that, to achieve the better performance, instead of keeping all default parameters, on each node we changed the number of tasks (Map or Reduce) can be run simultaneously, and adjust the memory size so that it suits the volume of data.

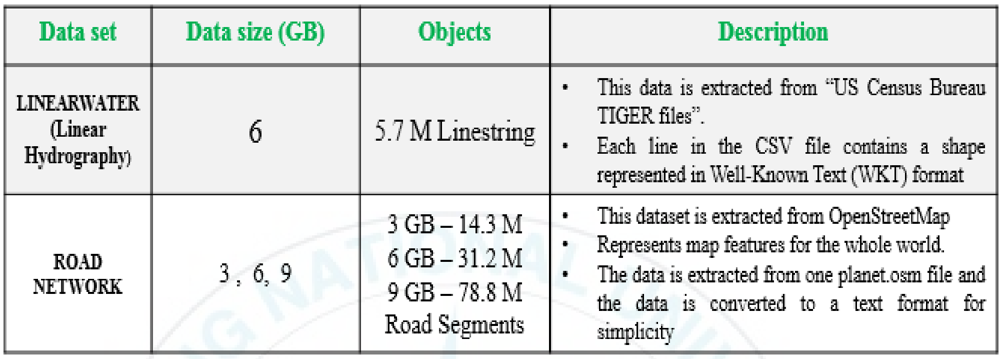
4.1.2 Data Set

All experiments are executed on two real spatial data sets. The spatial objects in the data sets are angular coordinates (CSV) in (latitude, longitude) format, and each data set is in tabular format where each line represents an object.

The first data set is “LINEARWATER” (Linear Hydrography), it is extracted from “US Census Bureau TIGER files”, each line in this CSV file contains a “Line String” represented in Well-Known Text (WKT) format. The size of this data set roughly six Gigabyte and the number of objects in this data set roughly five million seven hundred thousand objects, as shown in Figure 16. The distribution of this data is shown in Figure 17.

The second data is used in our experiments is “ROAD NETWORK”, this data set is extracted from OpenStreetMap. OpenStreetMap is a map of the world, it is created by many people and free to use under an open license. For simplicity, this data set is converted to a text format (TXT), as shown in Figure 16.

With the second data set, I use three different size are 3GB, 6GB, and 9GB with corresponding the number of objects are roughly 14 million, 31 million, and 78 million. The distribution of this data is shown in Figure 18.



Data set	Data size (GB)	Objects	Description
LINEARWATER (Linear Hydrography)	6	5.7 M Linestring	• This data is extracted from "US Census Bureau TIGER files". • Each line in the CSV file contains a shape represented in Well-Known Text (WKT) format
ROAD NETWORK	3 , 6, 9	3 GB – 14.3 M 6 GB – 31.2 M 9 GB – 78.8 M Road Segments	• This dataset is extracted from OpenStreetMap • Represents map features for the whole world. • The data is extracted from one planet.osm file and the data is converted to a text format for simplicity

Figure 16. Spatial Data Sets used in the Experiments

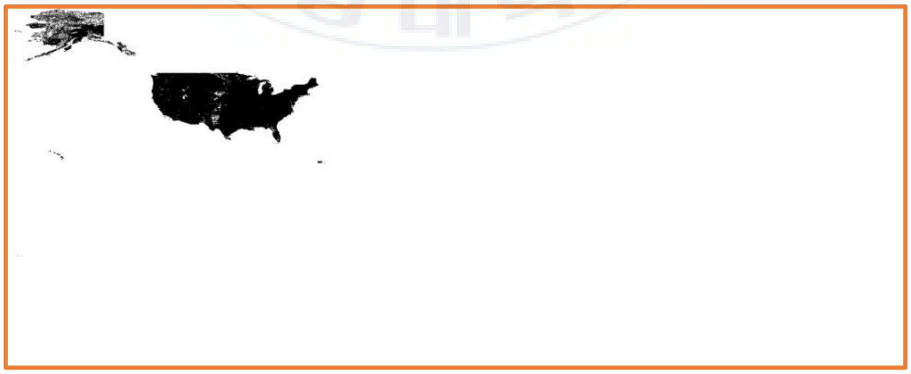


Figure 17. Distribution of LINEARWATER Data Set

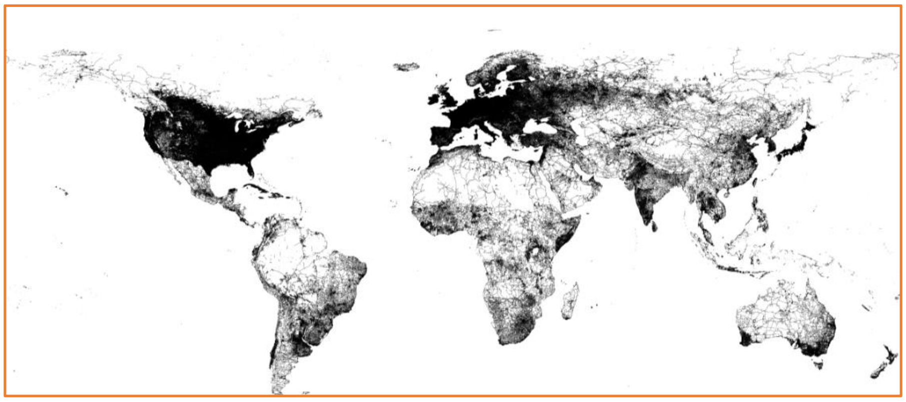


Figure 18. Distribution of ROAD NETWORK Data Set

4.2 Experimental Result

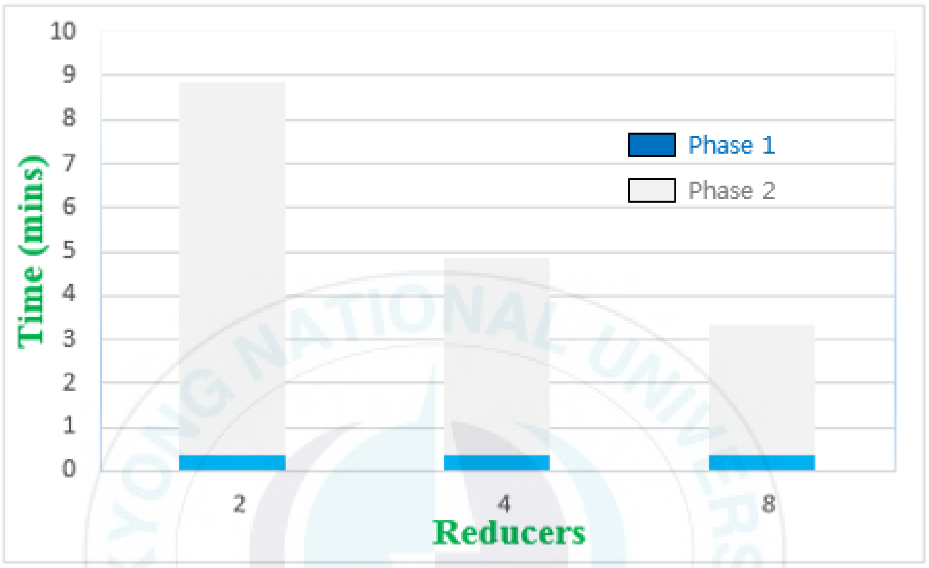
4.2.1 Time Performance

4.2.1.1 Our Approach

This experiment show the time for R-tree construction with 10% ratio of input file in the first phase and the different parameter R in the second phase. That is the number of concurrent “small” R-trees, from 2 up to 8. The data set that we use in this experiment is **LINEARWATER** data set.

The most computationally intensive part is performed in the second phase by Reducers where the “small” R-trees will be constructed. With the fewer the number of Reducers, since each Reducer receives a large number of objects, the

R-tree construction time takes longer. When number of Reducers is increased, the number of objects in each Reducer will be decreased, thus the time of this phase will take shorter as shown in Figure 19.



LINEARWATER Data set – Sampling ratio 10%

Figure 19. Time Performance of Our Approach

4.2.1.2 Comparison with Z-curve Method

To evaluate the effectiveness of my proposed method, this section shows the comparison results between my proposed method with the Z-curve method that has been proposed by A.Cary that we discussed in Chapter Two about time performance. The data set that we use in this experiment is **ROAD NETWORK** data set.

4.2.1.2.1 Data Partitioning Phase

This section shows the comparison of time performance in the First phase (Data Partitioning) with different ratio of input data as shown in Figure 20 with ten percent ratio of input data and Figure 21 with one hundred percent ratio of input data. In both cases of input data ratio, the implementation time decreases as the number of Reducers is increased in same data size and the implementation time increases when the data size is increased in both methods (ISTR and Z-order curve). The only different between them is when the ratio of input size is increased from ten to one hundred percent, the implementation time is a little bit increasing.

In both cases of input data ratio, they also indicate that the implementation time of our proposed method is a little bit slower than previous proposed method but it is insignificant.

The reason is in previous proposed method, it only uses one array to store all the Morton numbers that are calculated by Z-order curve packing technique in Mapper, then in Reducer, a list of splitting points is determined from that array, but in our method, in Reducer, we use two arrays, one to store x-coordinate of a center point of all objects that are sorted in ascending and one to store y-coordinates of a center point of all objects that are also sorted in ascending, the coordinate of a center points are implemented in Mapper. In order to determine the “Longest coordinate”, we calculate the distance between the start point and the end point of both arrays, then we compare and choose the larger one as the “Longest coordinate”, then data is partitioned based on the “Longest coordinate” array.

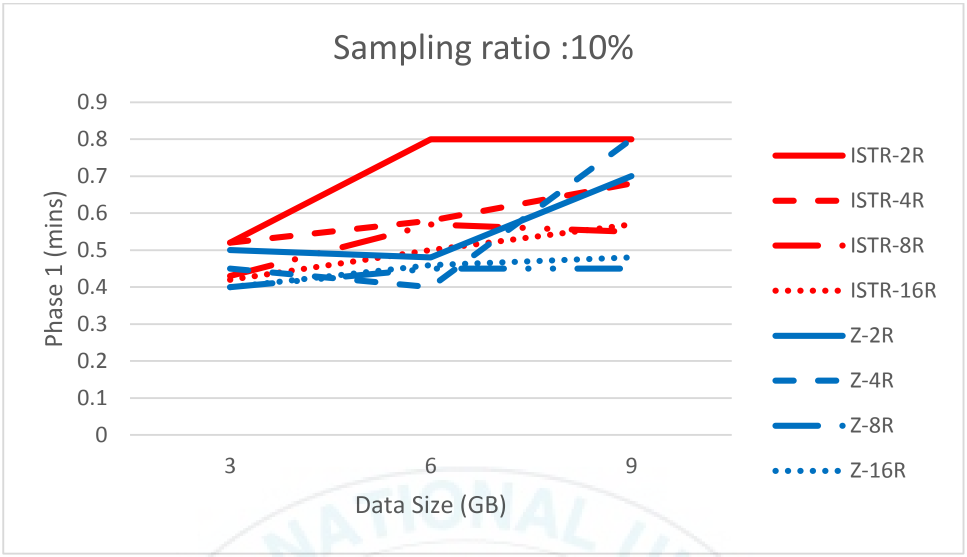


Figure 20. Time Performance of Phase One with 10% ratio of Input Data

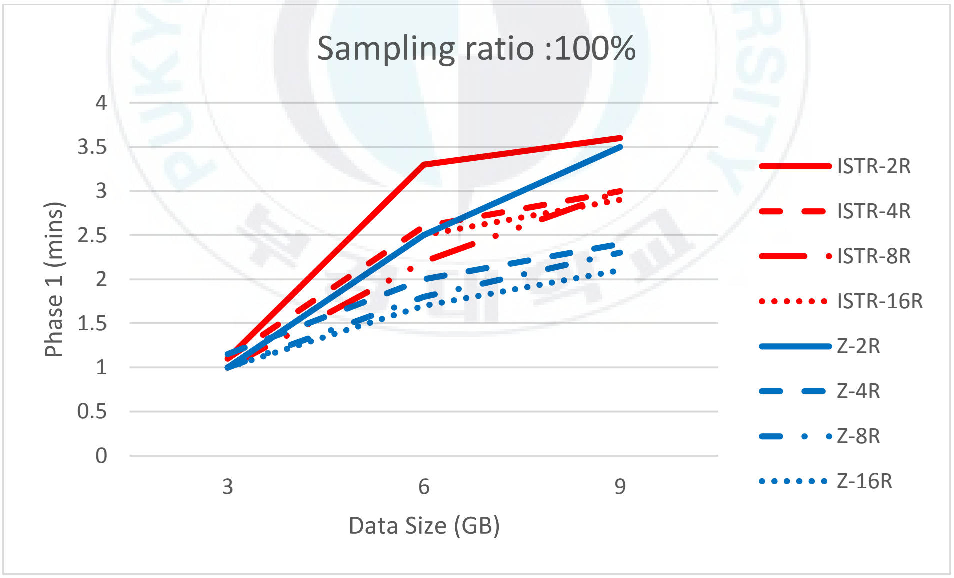


Figure 21. Time Performance of Phase One with 100% ratio of Input Data

4.2.1.2.2 R-tree Construction Phase

This section shows the comparison of time performance in the Second phase (R-tree construction) with different ratio of input data as shown in Figure 22 with ten percent ratio of input data and Figure 23 with one hundred percent ratio of input data.

As we discussed before, Reducers in second phase is the most computationally intensive part where all actual “small” R-trees construction occurs, so the implementation time for this phase take much longer than the first phase.

Similar to the first phase, the implementation time decreases as the number of Reducers is increased in same data size and the implementation time increases when the data size is increased in both methods (ISTR and Z-order curve).

Figure 24 and Figure 25 show total implementation time of phase one and phase two of both methods with ten and one hundred percent ratio of input data.

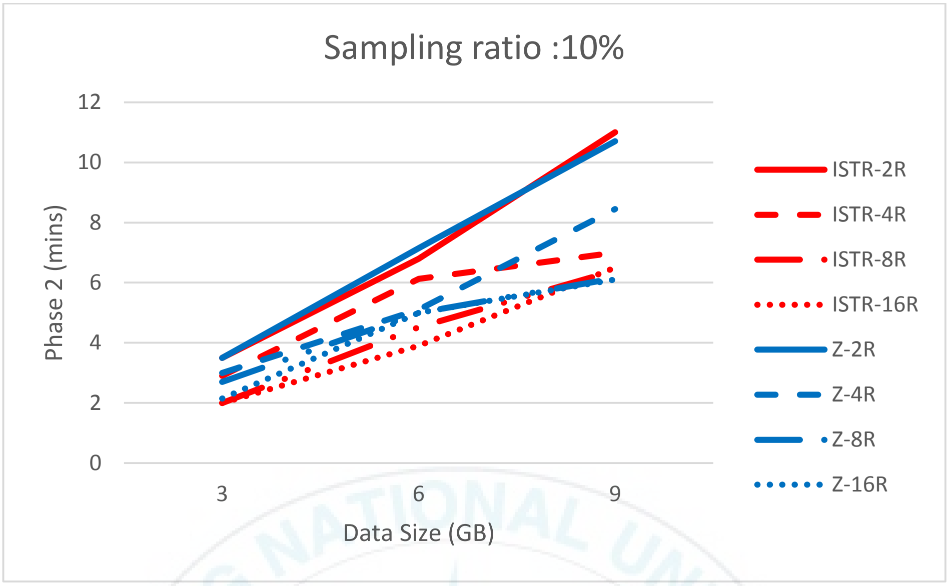


Figure 22. Time Performance of Phase Two with 10% ratio of Input Data

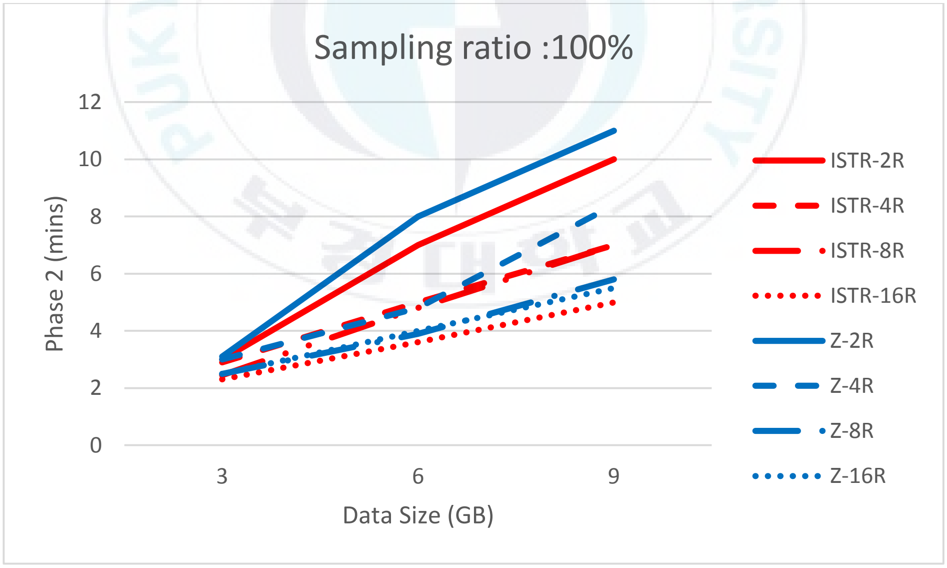


Figure 23. Time Performance of Phase Two with 100% ratio of Input Data

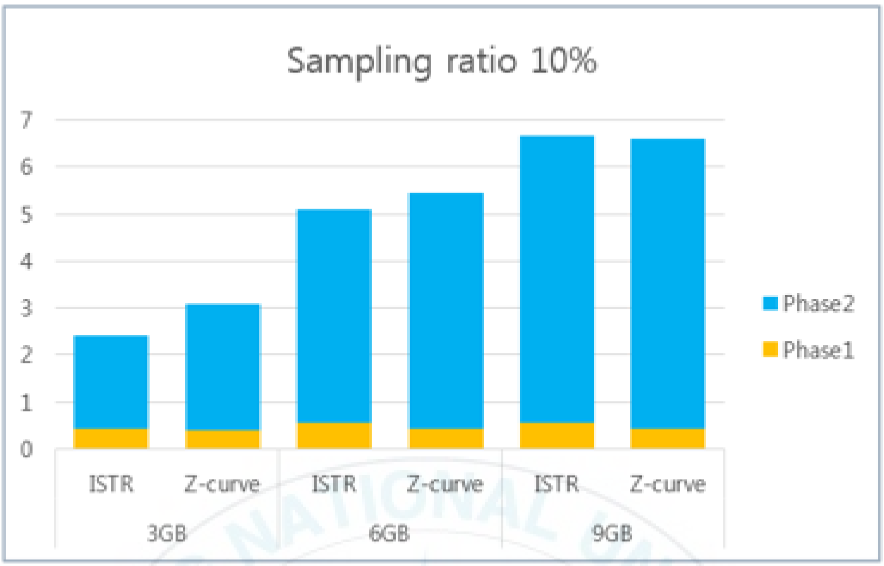


Figure 24. Total Implementation Time with 10% ratio of Input Data

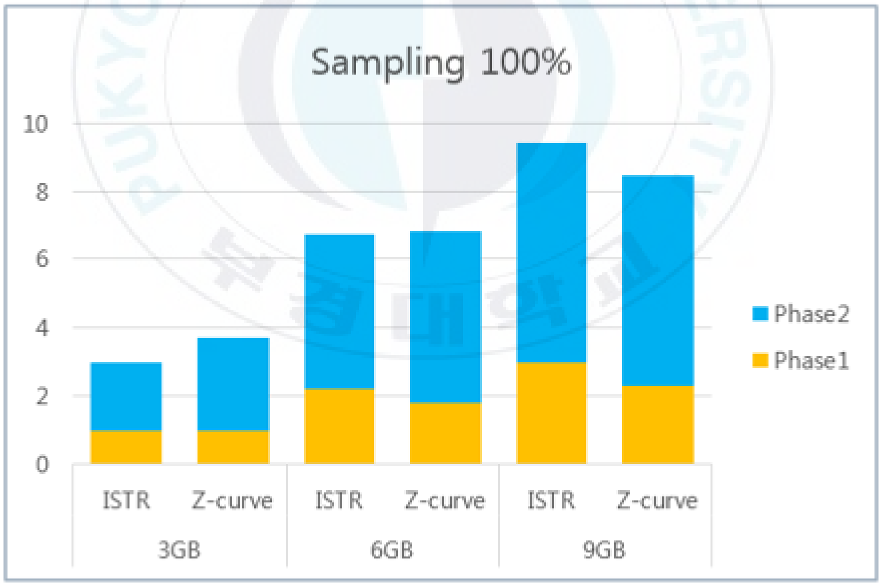


Figure 25. Total Implementation Time with 100% ratio of Input Data

4.2.2 Quality of Generated R-tree

This section show the comparison of the quality of generated R-tree between our proposed method with Z-curve method. We use two following equations to compute the area and overlap metrics respectively for given consolidated R-tree with root T as shown in Figure 26.

$$Area(T) = \sum_{i=1}^{n} Area(T_i.MBR)$$

$$Overlap(T) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} Area(T_i.MBR \bigcap T_j.MBR)$$

- **n** : the number of smalls R-tree generated by Reducers of root T
- $\mathbf{T_i}$: i-th child node of T

Figure 26 Equations for Area and Overlap Computation of Generated R-tree

4.2.2.1 Area Comparison

4.2.2.1.1 Comparison with Single R-tree

This section shows the comparison of total area of the generated R-tree between our approach with the single R-tree. This experiment show that the total area of

R-tree slightly decreases as the number of Reducers is increased with our proposed method and this total area is much less than the single R-tree as shown in Figure 27. The data set that we use in this experiment is **LINEARWATER** data set.

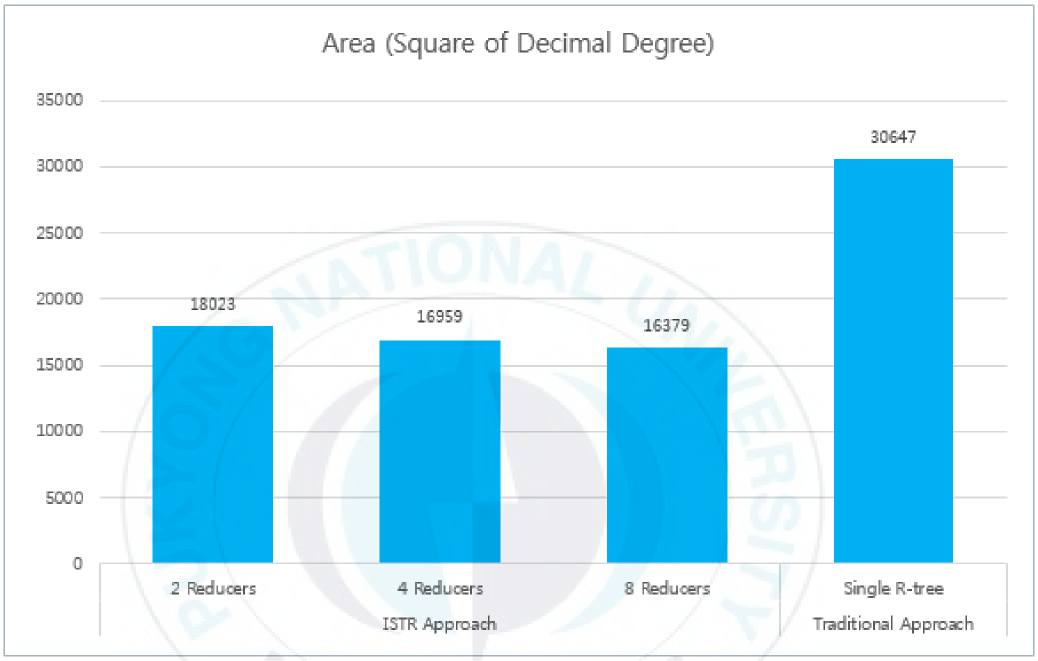


Figure 27. Comparison of Total Area between Our Method with The Single R-tree

4.2.2.1.2 Comparison with Z-curve Method

This section shows the comparison of total area of the generated R-tree between our method with a previous proposed method. This experiment is performed with different ratio of input data in phase 1 and on 8 Reducers in phase 2 with different data size. It shows that on each different data size, the total area of R-tree is built by our proposed method is much less than total area of R-tree is built by Z-curve method as shown in Figure 28 , Figure 29 and Figure 30 with the corresponding ratio of input data are 1%, 10% and 100% . The data set that we use in this experiment is **ROAD NETWORK** data set.

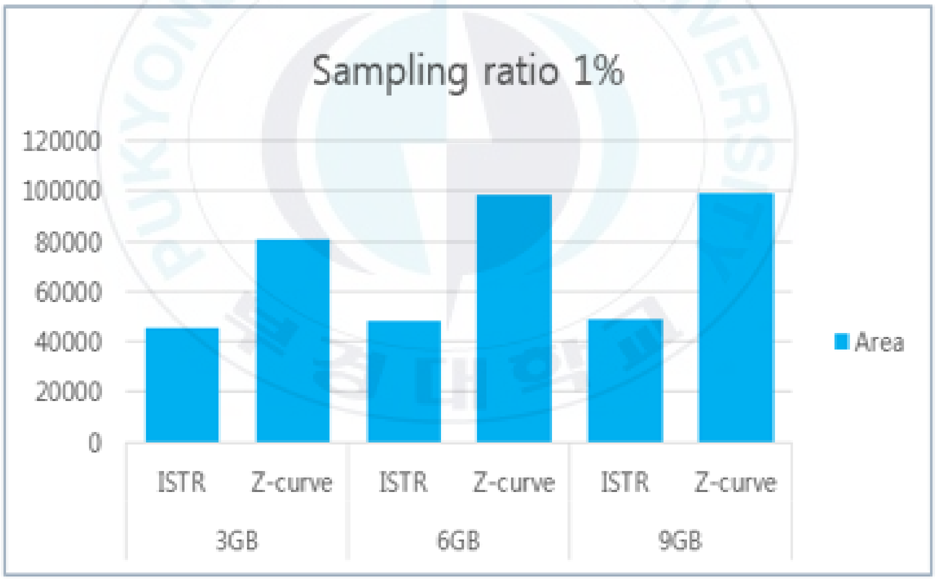


Figure 28. Comparison of Total Area of R-tree with 1% Ratio

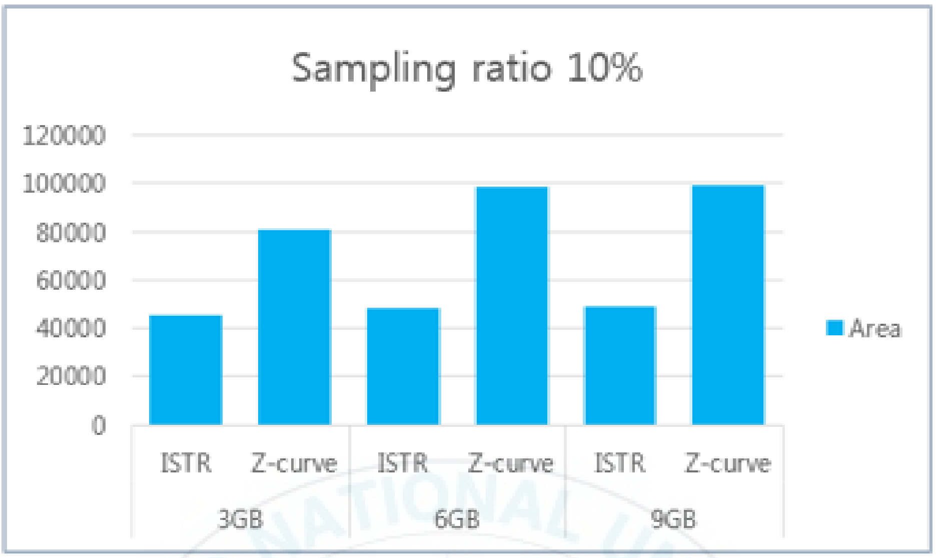


Figure 29. Comparison of Total Area of R-tree with 10% Ratio

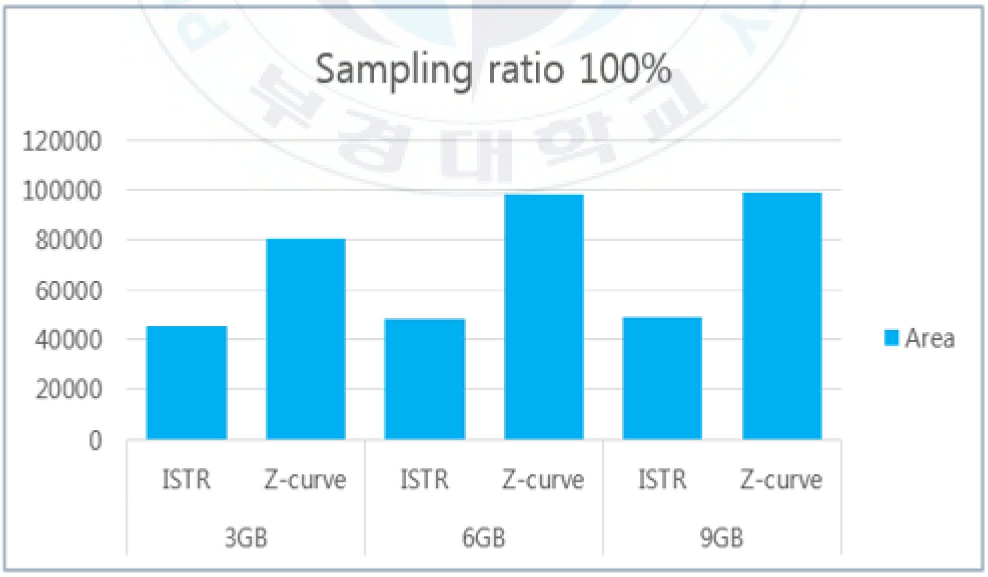


Figure 30. Comparison of Total Area of R-tree with 100% Ratio

4.2.2.2 Overlap Comparison

4.2.2.2.1 Comparison with Z-curve Method

This section shows the comparison of overlapping area between “small” R-trees between our proposed method with Z-curve method. This experiment is performed with 10% ratio of input data in phase 1 and different number of Reducers in phase 2 on different data size.

It shows that on each different data size, with our proposed method, the overlapping area slightly increases when the number of Reducer is increased. In case of Z-curve method, the overlapping area significant increases when the number of Reducer is increased as shown in Figure 31. Besides that, this experiment also indicates that the overlapping area between “small” R-trees are built by our proposed method is much less than overlapping area between “small” R-tree that are built by Z-curve method as shown in Figure 31.

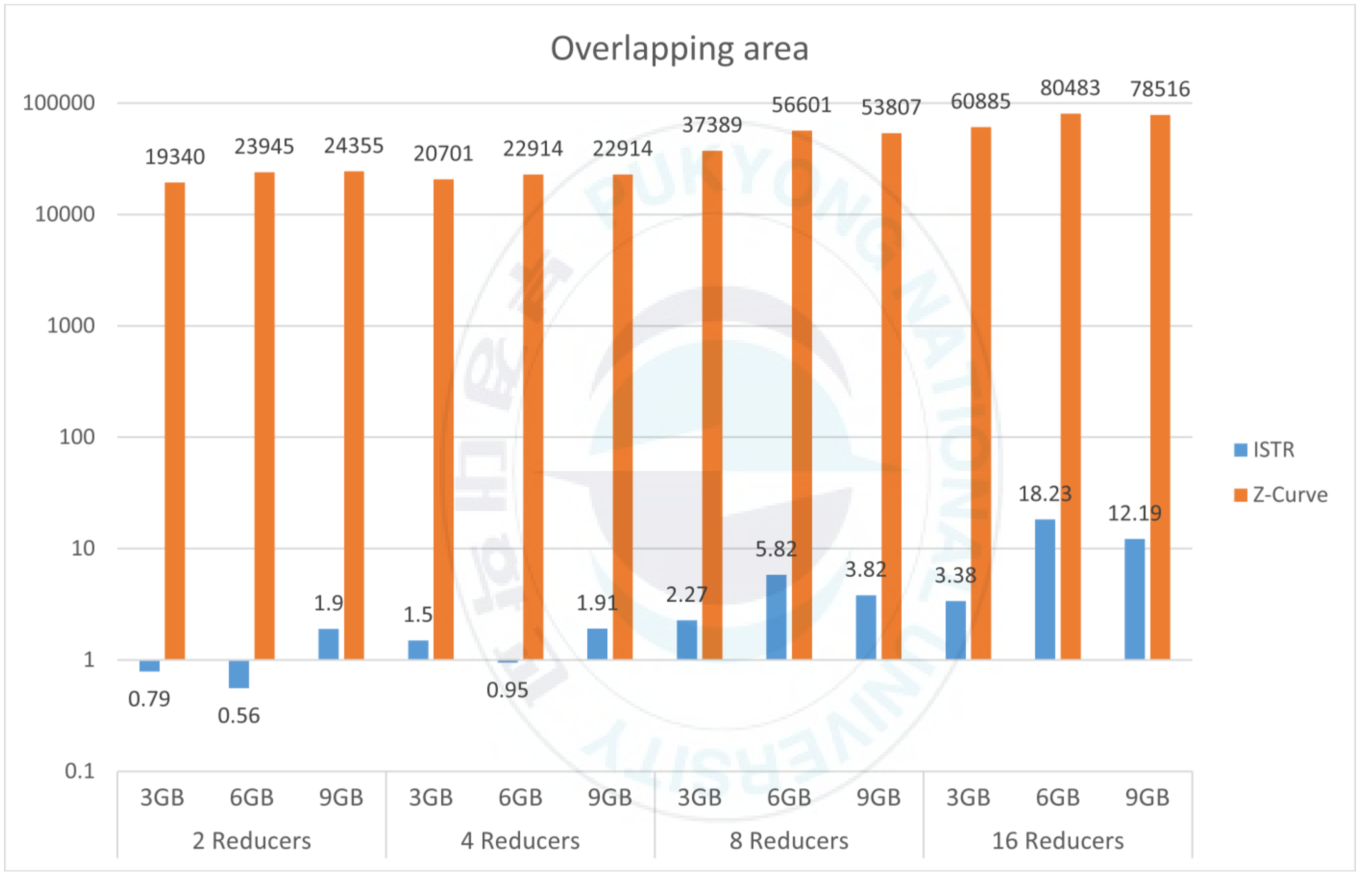


Figure 31. Comparison of Overlapping Area between Our Method and Z-curve Method

V. Conclusion

In this thesis, we proposed a scheme that has three phases for parallel R-tree construction, in which, the first two phases are executed in parallel with MapReduce model, while the last phase is executed outside the cluster because it does not require the high computational. In the first phase (Data Partitioning), to minimize the “Dead Area” in resulting Minimum Bounding Rectangles (MBRs) and overlap between these MBRs, instead of using Z-order curve as the geo-packing technique to grouping spatially neighboring objects, we propose a new method, called “Improving of STR” (ISTR). My proposed packing technique method was inspired by the Sort-Tile-Recursive (STR) algorithm. However, instead of partition data by x-coordinate, firstly, we determine the “Longest coordinate”, then data will be partitioned based on the “Longest coordinate”.

To evaluate the effectiveness of my proposed method, I make the comparison between my proposed method with the previous method that has been proposed by A.Cary that we discussed in Chapter Two. From the experiments, with my proposed method, although the implementation time for the first phase is slightly higher than the previous one but it is insignificantly. As we discussed before, the most important consideration is the quality of the generated R-tree, from the experiments, with my proposed method, the total area of the final R-tree is much less than the previous approach, and the overlapping area between “small” R-trees is also much less than the previous approach.

With my proposed method, I hope to contribute to improve the quality of R-tree construction and reduce the construction time for an available huge volume of data set. With our proposed schema, it can be used in many kind of applications in the field of databases in general and in particular, spatial data.

Nowadays, with the amount of spatial data is increasing significantly, with the availability of Big Data, commodity hardware, has opened many opportunities for analyzing astonishing data sets quickly and cost-effectively for the first time in history.

REFERENCES

[1] I Polato, R Ré, A Goldman, F Kon, A comprehensive view of Hadoop research—A systematic literature review, Journal of Network and Computer Applications, Elsevier, Vol.46, pp. 1-25, 2014.

[2] Antonin Guttman, R-trees- A Dynamic Index Structure for Spatial Searching, ACM SIGMOD, pp. 47-57, 1984.

[3] Apache Hadoop: http://hadoop.apache.org.

[4] J Dean, S Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, 50th anniversary issue, Communications of the ACM, pp.107-113, 2008.

[5] A.Cary, Z Sun, V Hristidis, N Rishe, Experiences on Processing Spatial Data with MapReduce, 2009 SSDBM, LNCS, Springer, Berlin Heidelberg, Vol. 5566, pp. 302–319, 2009.

[6] Z-order Curve, https://en.wikipedia.org/wiki/Z-order_curve

[7] S. T. Leutenegger, J. M. Edgington, M. A. Lopez, STR: A simple and efficient algorithm for R-tree packing, 13th International Conference on Data Engineering, pp. 497-506, 1997.

Acknowledgement

Time flies like an arrow. Finally, I have finished my master course at Pukyong National University (PKNU). Also, this thesis would not have been completed without guidance and supporting comments from people around me, who also braces me to keep studying.

First of all, I would like to express my most sincere gratitude to my advisor, **Prof. Ha-Joo Song**, who gave me the scholarship opportunity and fully support this research. He didn't only give me a lots of knowledge in specialized knowledge, but also an abundance of experiences in order to encourage me improving my soft and hard skills.

I would like to thank my Lab mate, also my brother, **PhD student: Mr. Sugarbayar Otgonchimeg,** who are always ready to help me in any condition and any problems, also I would like to thank all Korean students in my lab, who taught me about the good and bad things about life in Korea. I also would like to thank my Korean friend, also my brother, **Assistant Professor in Catholic University of Busan: Mr. Jun-Ho Huh,** who are always ready to share experience in scientific research.

I would like to thank my Vietnamese friend in PKNU, also my best Vietnamese younger brother, **Mr. Nguyen Viet Hoan,** who are ready to help me in any situations, both inside and outside of campus. Thanks to all Vietnamese students in PKNU who encourage and always give me a helping hand. Thanks for your kind support**.**

Most importantly, I would like to express my most heartfelt gratitude to **my Parents**, who was always at my side, always as my spiritual toehold whenever I

get stuck in life. I love you so much. And I also would like to thank my aunts: **Ms. Nguyen Thi Hoai My, Ms. Nguyen Thi Thu Trang**, my uncle: **Mr.Nguyen Khac Vu**, who had help, give me some money from the early days when I go to study abroad, and I would like to thank all my family members whose name I could not mention here was always ready to support and help me. I love you guys so much.