5-1-2017

Finding Top-k Dominance on Incomplete Big Data Using Map-Reduce Framework

Payam Ezatpoor
University of Nevada, Las Vegas

FINDING TOP-K DOMINANCE ON INCOMPLETE BIG DATA USING MAP-REDUCE FRAMEWORK

by

Payam Ezatpoor

Bachelor of Science (B.Sc.) in Information Technology
Sanandaj Azad University
2014

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2017

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

March 31, 2017

This thesis prepared by

Payam Ezatpoor

entitled

Finding Top-k Dominance on Incomplete Big Data Using Map-Reduce Framework

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Justin Zhan, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Fatma Nasoz, Ph.D.
Examination Committee Member

Kazem Taghva, Ph.D.
Examination Committee Member

Ge Lin Kan, Ph.D.
Graduate College Faculty Representative

Abstract

Incomplete data is one major kind of multi-dimensional dataset that has random-distributed missing nodes in its dimensions. It is very difficult to retrieve information from this type of dataset when it becomes huge. Finding top-k dominant values in this type of dataset is a challenging procedure. Some algorithms are present to enhance this process but are mostly efficient only when dealing with a small-size incomplete data. One of the algorithms that make the application of TKD query possible is the Bitmap Index Guided (BIG) algorithm. This algorithm strongly improves the performance for incomplete data, but it is not originally capable of finding top-k dominant values in incomplete big data, nor is it designed to do so. Several other algorithms have been proposed to find the TKD query, such as Skyband Based and Upper Bound Based algorithms, but their performance is also questionable. Algorithms developed previously were among the first attempts to apply TKD query on incomplete data; however, all these had weak performances or were not compatible with the incomplete data. This thesis proposes MapReduced Enhanced Bitmap Index Guided Algorithm (MRBIG) for dealing with the aforementioned issues. MRBIG uses the MapReduce framework to enhance the performance of applying top-k dominance queries on huge incomplete datasets. The proposed approach uses the MapReduce parallel computing approach using multiple computing nodes. The framework separates the tasks between several computing nodes that independently and simultaneously work to find the result. This method has achieved up to two times faster processing time in finding the TKD query result in comparison to previously presented algorithms.

Acknowledgements

"Immeasurable appreciation and deepest gratitude for the help and support to the following persons who made this scientific research possible.

Dr. Justin Zhan: academic advisor for this masters thesis. For his support, guidance and helps toward this research. For sharing his knowledge and provide ample opportunities and equipment to make this thesis possible.

Dr. Kazem Taghva, **Dr. Fatma Nasoz** and **Dr. Ge Lin Kan**: for accepting to serve on my advisory committee and their help and guidance toward making this paper as better as before.

Mrs. Janine Barrett: for her motivations and support and help me stand on my feet strongly and follow my dreams.

And to my parents:

Fateh Ezatpoor and **Mahin Kalami**: who helped me to strive for the best and taught me to follow my goals with confidence. "

Payam Ezatpoor

University of Nevada, Las Vegas

May 2017

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In a given dataset R with multiple dimensions d, an in-depth analysis may be required to find the most powerful or influential values throughout the dataset. The most influential values can also be referred to as the dominant objects over the other objects present in the data. Based on a specific pre-defined definition, referred to as dominance definition. A value can be evaluated as a dominant value based on the dominance definition. Discovering the dominant values in a dataset helps to fulfill different data mining purposes. Suppose the dataset R has n objects (= items) from d dimensions that can be imagined as a two-dimensional array with a range of objects and dimensions. Each item in R, accommodates the corresponding value of *(n,d)*. In the real-life application, a database of movies with different movie ratings from a range of users is a reasonable sample of a multi-dimensional dataset. The values are the ratings for each movie m from user *0* to *n-1* represented for every object m as:

$$\sum_{i=1}^{m-1} i = (\sum_{p=0}^{n-1} p, \sum_{q=0}^{d-1} q) \tag{1.1}$$

Top-k dominance query finds the most powerful values which dominate other values in the same dimension by using a pre-defined scoring function. Top-k is one of the main uncertain queries that returns the top-k objects with the highest scores according to a scoring function [WLLW13]. The dominant values are distinguished using the dominance definition which defines when and how a value can be dominant over the other.

Finding the answer for TKD queries can be accomplished by using different methods and algorithms. While thinking about finding dominant values in a dataset, the first and easy approach that might come to mind is to compare each two items in the dataset individually. This

naive approach implies the pair-wise comparison between the values of different objects in the same dimension. In this approach, a comparison is required for every two values existing in the dataset. The observed dominant value in each comparison can be used to find the final top-k dominant values later. Indicating the top-k dominant values can be challenging when facing big data, processing time can reach to infinite even if the computational cost be ignored, therefore, this approach may not be the best way of solving this problem.

Several algorithms have been proposed to apply the TKD queries which their performance are more acceptable and efficient than the naive approach. Based on [KML08],[PFS05], applying the top-k dominance query is possible in the incomplete dataset using Skyline query processing. Skyline algorithm separates the uniform values into different buckets. The buckets groups the dataset by dividing the dataset into chunks that have same missing-value dimension. The buckets are easier and faster to process. By having separate top-k dominant values for each bucket, and finally combine them together the top-k values of the whole dataset would be possible. Upper Bound Based algorithm is another method which works by finding top scores using the bit-wise comparison between values that will be covered later in the paper [MGZ$^+$16].

The Bitmap Index Guided algorithm is another approach proposed by [MGZ$^+$16] that can greatly enhance the performance of finding the TKD query. Although, our analysis shows that this difference is not significant in comparison to other algorithms like k-Skyband Based or Upper Bound Based. As the size of data increases over time and Big Data emerging more commonly in this field, it is useful to consider applying the same logic and algorithms more realistically and prepare them to face real-world problems with exponentially larger datasets. Based on the results indicated in [MGZ$^+$16] the performance of the BIG algorithm for finding TKD results has been improved by using tiny subsets of an incomplete multi-dimensional data. Although, the performance of the BIG approach is not clear in the real datasets which are exponentially larger in size.

In all of the aforementioned algorithms, incomplete data is the base dataset for applying the TKD queries. The incompleteness in the Incomplete Data is independent, means that both present and missing values in this type of dataset are not related to each other and there is no way to find values based on others using any probabilistic approaches. For incomplete data, neither prior knowledge nor calculation of data is required. Derivation of values is assured and not based on probabilities, but in the uncertain data, the missing values can be found based on experience or prior information. Also, the probabilistic concept of TKD approach has been reviewed by [LC13] for working on missing data.

The results of top-k dominance algorithms help us to enhance our ability to obtain information and knowledge from unprocessed raw data that contain numerous missing values. The incomplete dataset will soon be the ever-present data in every system and finding the top-k dominant values throughout a dataset helps us to design smart and intelligent systems such as movie recommenders that have strong, accurate, efficient, and real-time recommendations.

This paper explains attempts to enhance the performance of the Bitmap Index Guided algorithm while dealing with large datasets by getting help from not only one machine, but having multiple processing machines working simultaneously to find the TKD query result in a fast and accurate way. Using single computing nodes, even with powerful computing components, is still not enough for processing the large real-time data, and the process duration makes those systems completely unresponsive. The machine power resources are not always able to accommodate the algorithm meta-data and temporary files. In those cases, limited processing power and memory capabilities become a significant difficulty. Ideally, the MapReduce framework is one productive method this paper tries to focus on implementing a new enhanced algorithm that can efficiently apply TKD queries in a faster way and by using multiple processing machines working simultaneously to find the TKD query results.

To our knowledge, this thesis is the first work that considers big data in the area of finding the top-k dominance values. Applying the MapReduce framework on this subject is an innovative approach which guarantees enhanced and acceptable performance. This work also has several different aspects of innovation in comparison to previous works which are focused on uncertain data or complete data. Efforts in this context are not only focused on incomplete data but also the massive size incomplete Big Data. Our work provides new ways of thinking about a specific usage area that has not been considered thoroughly so far.

Chapter 2

Structure and Related Work

In this section, related works with regard to finding top-k dominance, incomplete data, and Bitmap Indexing were reviewed. First, an overview of the previous works for applying top-k dominating (TKD) queries will be provided. Then we proceed by explaining related works about incomplete data. Finally, we consider the bitmap indexing related works, which help us to distinguish different approaches and compare to the method provided in this thesis.

2.1 Top-k Dominance

TKD query finds the best values in a dataset based on a pre-defined goodness criterion in every single use case. Dominancy shows that at least one attribute or value makes an object better than the object it dominates. The dominance admittance is based on a pre-defined definition or relationship which reviews the best dominance relationship in terms of handling constraints of the system. Following the [GLMC13] approach, budget constrained optimization query help to increase the profitability of products. Furthermore, L. Yiu and N. Mamoulis [YM09] reviewed the TKD queries on multi-dimensional datasets. They propose ITD an enhanced algorithm applied to indexed multi-dimensional datasets without using Skyline-based algorithms. Furthermore, they also proposed the LCG algorithm that computes upper bound scores using the tree structure, giving a relaxed version of the top-k dominating query. [MCYC06] suggests a new algorithm that refines the object accesses throughout top-k processing. X. Lian and L. Chen [LC09] consider Probabilistic Top-k Dominating (PTD) query in uncertain data. [HPZL08] works on uncertain data by applying probabilistic top-k queries by assigning a probability threshold. Their approach reduces the PTD search space by pruning the improbable values in the uncertain dataset. It finds the PTD query

for values that dynamically dominate all possible values in the dataset. X. Han et al. [HLG15] work on TKD queries on massive data, accomplished by sorting and listing values and making the process faster than other methods. Their proposed TDEP algorithm sorts and prunes the data with specifically selected objects, which helps to make multi-criteria decisions for the data.

Multi-dimensional databases are primarily used in applying the top-k dominating queries. [TV14] study processing top-k dominating queries over dynamic attribute vectors where finding the distances depends on the defined metrics between objects. Their algorithms benefit from the applied metric space to solve the TKD query. Results of their work show that out of several reviewed approaches such as SBA and ABA, the pruning-based algorithms show the best performance. [SC14] and [MGZ$^+$16], follow the combination of top-k and Skyline queries that led to top-k dominating query that [SC14] process the more complex situations. [LEB13] provides the work on Skyline in Crowd-Enabled Databases with consideration of incomplete datasets. Their proposed query, continuous top-k dominating query (cTKDQ), can continuously generate the answers to the query after any changes in the data. [HLLG16] applies the top-k query on massive data without considering the Skyline queries which distinguishes it from top-k dominance queries but makes it related because of the TKAP model which uses adaptive pruning processes on massive datasets to enhance performance. [MBP06] also follows the same structure as [HLLG16] but more for monitoring purposes for tracking top-k queries. Also, [ZZY13] addresses the concurrency problems while applying TKD queries based on a service selection scheme to apply TKDs based on specific requests.

Papadias et al. [PFS05] uses Skyline Computation to find dominated values. First, they introduce the branch-and-bound skyline (BBS) algorithm that accesses the skyline points using an R-tree and Nearest-Neighbor search and then implements the TKD queries. Furthermore, top-k dominance has been further considered in [NC16], in which the top-k Dominance Range Query (TkDR) operator helps to find the most interesting objects in any uncertain datasets by taking advantage of the probabilistic skyline queries. Their approaches focused on probabilistic skyline queries, which try to enhance the TkDR performance of uncertain datasets.

Further details and definitions have been provided in [WLLW13] to give a better understanding of top-k and Skyline queries based on uncertain data. The presented concepts in [WLLW13] are helpful for grasping the underlying concepts of this paper. The mentioned efforts in this subsection provide different approaches for dealing with top-k dominating queries, and each may have different implementation processes. They consider the complete, incomplete, probabilistic and uncertain data and query types, and several performance outcomes. MRBIG approach

adds a new improved view for the top-k dominating queries on a massive scale by improving the performance.

2.2 Incomplete Data

The main characteristic of incomplete data is based on having missing values in its dimensions. Khalefa et al. [KML08] propose TKD query processing using the ISkyline algorithm that is especially suitable for incomplete data. Haghani et al. [HMA09] study incomplete unsynchronized data streams and try to address the issue of continuously monitoring top-k queries through their efficient pruning approach. Soliman et al. [SIBD10] provides a probabilistic model and express types of ranking queries to apply the TKD query.

S. Razniewski and W. Nutt [RN11] assure the data integrity of query answers while dealing with incomplete data. [BCZW14] considers the clustering of the incomplete datasets in high-dimensional big data and improves performance for clustering. This approach also reduces the dimensions of the dataset using a hierarchical clustering structure. [LEB13] uses incomplete datasets and proposes an approach to overcome current defects while applying skyline queries, and enhances the quality in crowd-enabled databases.

The incomplete data requires different processing methods than other types of data such as complete, probabilistic, or uncertain data. One must solve issues like the presence of missing values and how to manage the value absence without wasting available computing resources, and this requires innovative methods. The missing values in an incomplete structure require complex methods of computation which distinguish them from complete datasets [IL84], [CJS$^+$14], [AKO07]. This incompleteness can differentiate the methods from processing information in a database to conducting relational operations [IL84], or searching or mining process of the incomplete data [CJS$^+$14]. Some works provide languages to handle incompleteness and study algebraic methods to map them to completed datasets [AKO07] and how to map an incomplete data to a complete dataset [Lib14]. Finding top-k dominance and relating it to the incomplete data is a major point that can differentiate MRBIG approach from the mentioned works. This paper addresses further issues in this field by considering big data.

2.3 Bitmap Indexing

Bitmap Indexing is a way to ease the processing of non-uniform data. By using the bitmap indexing approach, the results discovery will be efficient and easier to handle in most of the cases. The bitwise operations in the Bitmap Indexing generate data patterns which are simpler for a machine to use and process, but in some cases might make the process over complicated. Big Data is one of the environments where bitmap indexing can be either useful or dangerous. Bitmap Indexing can become an additional workload for the system while dealing with multi-attribute data. Creating a bitmap index for a particular algorithm can be as complex as the algorithm itself in some cases. Therefore, compression of bitmap indexes has also emerged as a useful process, to make the bitmap indexes simpler and easier to navigate and process.

Bitmap indexing compression speeds up the processes and makes the algorithms that utilize this method more efficient, as can be seen in [WSS08] with the effects of compression on multi-component and multi-level compressed bitmap indexes. There are several methods for compressing bitmap indexes, such as BBC, CONCISE, and WAH methods that [CWZ+15] has considered thoroughly and reviewed about Big Data. Furthermore, to make the compressions more stable [WOS02] has proposed their word-aligned hybrid code (WAH) as a compressing method for bitmap indexes that leads to improved performance. Each method has different CPU and GPU runtimes as well as varying Segmentation, Chunking, etc., configuration characteristics that are utilized for various use cases [CWZ+15].

X. Miao et al. [MGZ+16], review the TKD query on incomplete data as one of the first attempts to solve this issue using Bitmap Index Guided (BIG) algorithm. They use the Bitmap Indexing as the infrastructure for their algorithm and to perform TKD queries on incomplete datasets. After conducting compression on the bitmap indexing, they propose the IBIG algorithm, which has the same structure of BIG but uses the compressed bitmap indexing method. There is no significant difference regarding the performance and runtime of the algorithms. This paper uses uncompressed bitmap indexing as the base of work.

Chapter 3

TKD Query on Incomplete Data

In this section, we review the procedure to apply the TKD query to incomplete data as well as the problem statement for finding the top-k dominant values. Various algorithms have been proposed to handle Top-k dominance. Some of these algorithms are handling incomplete data, which an overview is provided later. Later in the context, the structure and functionality of the Bitmap Index Guided algorithm (BIG) will be considered.

3.1 Problem Statement

In this section, we address the issue and illustrate the details of implementation of the MRBIG algorithm. TKD query on incomplete data starts with an incomplete dataset R with n dimensions and m items. All n dimensions of an item are depicted as follows:

$$\sum_{i=1}^{n} d_i \tag{3.1}$$

By using the pairwise comparison method or any other related algorithm, the objective is to find the items that are dominant over the other values in the dataset. If item m_1 dominates over m_4, it showed as $m_1 \succ m_4$. Each item contains all the ratings from users as well as the missing values in some dimensions. Suppose the item m_1 having nine dimensions and showing the dimensions depiction:

$$m_1 = (d_1, d_2, d_3, d_4) = (2, -, 1, 0)$$

Each missing values would be represented as a dash (-). We define a dominance definition which lets us decide which value(s) can be dominant based on that definition described in dominance

definition. For instance, suppose the following theorem which defines a dominance definition. From now on, Dominance Definition remains the primary dominance definition for this whole paper.

Dominance Definition. *Given the two items m_1 and m_2, m_1 dominates over m_2 if the values in the dimensions of m_1 are larger than the values in the dimensions of m_2 excluding the missing values for all dimensions. In other words:*

$$\forall\ m_1[\forall d_i] \text{ and } m_2[\forall d_j] : m_1[\forall d_i] > m_2[\forall d_j] \tag{3.2}$$

The domination of m_1 over m_2 is denoted as $m_1 \succ m_2$ while holding the above condition.

In the Dominance Definition, the basis of dominance is based on the larger value between two corresponding dimensions. In other words, the greater value is a better value based on the Dominance Definition. Dominance Definitions define the goodness of value in a dataset and are a vital part of deciding what values are considered as dominant in any particular use case. Going back to the sample item m_1, by comparing $m_1 = (2, -, 1, 0)$ and item $m_5 = (-, -, 3, 2)$ as an example, the dominance is given to $m_5[d_3] = 3$ in compare to $m_1[d_3] = 1$. Based on Dominance Definition, item m_5 dominates m_5 due to the larger value it has. Missing values are not considered because they are not making any changes to the result. For example, users that have not submitted any rating for item m_5 cannot be a part of TKD query processing for item m_5.

3.2 Skyband Based Algorithm

In order to apply TKD query to a dataset, the easiest approach that first comes to mind is to compare the values of the whole dataset by doing a pairwise comparison. This method can be helpful for small datasets but following this approach for larger dataset causes poor performance and complete failure because of the mass resource-exhaustive one-by-one comparisons. There are different defects to this approach. The pairwise comparisons require more runtime to examine every single value in the dataset and massive storage to keep track of the progress. The inefficiency gets worse while dealing with the larger datasets known as big data.

To make the process of finding top-k dominant values possible and efficient we need to have more sophisticated algorithms, which are designed to make the application of TKD queries possible. One of the algorithms for this purpose is the Skyband Based algorithms, which have been a proper solution for incomplete data. Extended Skyband Based [MGZ+16] and Expired Skyline

algorithms [GMC+14] are among those algorithms which use the same concept. This process uses normalization methods by categorizing the data into different parts based on their missing values. Each item in the dataset redirects to its corresponding bucket based on the pattern in their missing values. Each bucket contains uniform items and their values. For instance, (-, 2, 4, 6) and (-, 8, 3, 2) both go into the same bucket due to having the missing values in the same dimensions. Following the same pattern, each bucket populates its members, and eventually, dataset would be completely divided into different groups.

	d_1	d_2	d_3	d_4		d_1	d_2	d_3	d_4
m_1	–	1	2	–	m_6	–	–	–	3
m_2	1	–	3	2	m_7	1	1	–	–
m_3	3	1	–	–	m_8	–	3	2	–
m_4	–	–	–	1	m_9	2	–	2	2
m_5	–	2	1	–	m_{10}	3	2	–	–

Table 3.1: Sample Incomplete Dataset

Among each group, the dominant candidate sets will be calculated. The candidate sets are created for each bucket. Candidate sets can have multiple potential values and depict all of the values in a bucket which can be a dominant value. By looking at the candidate set which is smaller in size and contains less data the TKD query can be applied easier. By intersecting the whole candidate sets from the buckets, the TKD query answer can reveal the final top-k result of the dataset.

b_1			
–	1	2	–
–	3	2	–
–	2	1	–

b_2			
–	–	–	1
–	–	–	3

b_3			
3	1	–	–
1	1	–	–
3	2	–	–

b_4			
1	–	3	2
2	–	2	2

Table 3.2: Skyband Based Data Bucketing Method

To clarify the concept, suppose the Table 3.1 as a sample incomplete dataset. As can be seen in Table 3.2, m_1,m_5 and m_8 has been bucketed into b_1 far as they have the same missing-data dimensions. Starting from the second dimension, can be seen that $m_8 \succ m_5$ on both dimensions d_2 and d_3 as well as the $m_5 \succ m_1$ on the dimension d_2. As far as, the dominancy on the dimension d_3 is not strictly distinguishable as both values are the same, further processing passes to next steps

and comparison to all candidate sets from the whole dataset. The candidate set (S_c) can be formed as Table 3.3:

The Skyband bucketing method cannot always be efficient as shown in Table 3.3. The worst case scenario to this approach is when the size of S_c equals the size of the dataset. In the Table 3.3, no bucket found to efficiently candidate one item and every bucket has more than one candidate, and the worse happens b_4 when all of the values goes into S_c. In general, the performance of Skyband bucketing method decreases as the size of the S_c increase.

b_1			
–	1	2	–
–	3	2	–
–	2	1	–

b_2			
–	–	–	3
–	–	–	1

b_3			
3	2	–	–
3	1	–	–
1	1	–	–

b_4			
1	–	3	2
2	–	2	2

Table 3.3: Candidate Sets (S_c) formation for potential TKD items

This approach gives us the ability to eliminate multiple dimensions at once and making a temporary complete dataset for the Skyband algorithm which helps to process the remaining values much faster. Looking at the real examples with thousands of items and dimensions can show the significant difference in speeding up the process. However, as shown above, there are still major defects to this approach which in some cases can make the whole process inefficient.

After obtaining all of the candidate values from each bucket, it is possible to make final pair-wise comparisons and reach to the final TKD result. Having the S_F set as the combined version of the candidate sets, the following summation gives the final input for performing the TKD query based on the Table 3.3:

$$S_F = \sum_{n=1}^{4} S_c[b_n] \tag{3.3}$$

By having the Table 3.4, final results for the top-k dominant values can be found by comparing the values at each dimension separately. It can be inferred that this process involves pairwise comparison as well and inherit the defects of that methods as well.

As mentioned earlier, the performance of Skyband based algorithms is strongly dependent on the size of S_F. The size of this set can directly increase the complexity of the TKD process and affect the performance of the whole system.

By considering Skyband Based algorithms for the MapReduce framework, using small

S_F			
3	2	–	–
2	1	–	–
2	–	2	2
1	–	3	2
–	3	2	–
–	1	2	–
–	–	–	3

Table 3.4: Final Candidate Set (S_F) for Skyband Based Algorithm

S_F can exhaust the MapReduce clusters by making them busy with simple calculations that waste the runtime by massive communications cost spent for synchronizing the computing nodes. Alternatively, by considering single machine procedure, large S_F can overload the single machine with large inoperable values. It is also possible that S_F equals the dataset size which mentioned before. This exceptional situation motivates to use another algorithm that can address these issues.

3.3 Upper Bound Based Algorithms

We described the defects of Skyband algorithm and the processing overload problems while facing massive data. Having a new approach that can address the issue can hugely improve the performance of finding top-k dominance queries. Another method for TKD is to use an integrated evaluation of items that is no longer based on the pair-wise comparison. UBB is one of the algorithms which is available to apply TKD queries on incomplete data and relies on the upper bound values in a dataset for retrieving the answer of TKD query [MGZ+16].

In any given dataset there is always one criterion to evaluate the values to empower the decision to choose top-k dominant values based on the dominance definition. So for each dominance definition, the first goal is to define criteria that work uniquely to distinguish between different values. After finding the criterion, a unique approach is to score values based on their powerfulness toward the dominance definition. A score is a number that is dedicated to each dataset to evaluate the values and make the decision of final TKD value possible. Scoring helps us to have fewer calculations and eliminate pair-wise comparison.

The UBB algorithm is mainly based on finding the frequency of the items that our picked value dominates. For complete data, the challenge is to compare each value with the other ones in the same dimension. For our desired multi-dimensional incomplete data, as can be seen in Sample

Dataset in Table 3.1, UBB takes each dimension independently and compares the picked value to other corresponding values and finds the frequency of values it dominates.

Upper Bound Based algorithm alters the dataset to make it easier to deal with by separating each dimension. This characteristic of UBB algorithm, can be imagined on looking at the columns of a two-dimensional array and process each column separately. Having dataset R, having m items and d dimensions, there are d separated column to process. Based on the dominance definition, dominance check can tell which value is dominant over a picked item and assign it to the proper B_i set. B_i is a list of all values that i dominates. For example, B_2 for m_5 shows all values that m_5 dominates over on second dimension. Obviously, the B_i for missing value is not considered as it is the same as the whole dataset for each dimension. By having B_i sets and obtaining the size of each B_i, UBB stores all the sizes into a single structure. Depending on the dominance definition the answer retrieves based on the scores.

The UBB algorithm suffers from the generic high volume exhausting pairwise comparison problem. This algorithm tries to separate the dimensions and then compare the whole dimension with the value it has. Having massive data requires a tremendous processing procedure to apply the Upper Bound method on the real world problems.

3.4 Bitmap Index Guided Algorithm

As reviewed in the previous sections, present methods enhance the performance in different ways but they still have various problems which make slow and inefficient in some cases. Using the naive approach we discussed at the beginning which contains pairwise comparison, requires a significant amount of time and inefficiency even with smaller data. Using the Skyband algorithms is also substantially dependent on the size of the dataset and having large datasets creates huge buckets of data, and using the TKD query on each would still be inefficient. UBB algorithms cannot be efficient as far as they require numerous comparisons between values. The Bitmap Indexed Guided is another algorithm that helps us to find the top-k dominant values by firstly generating a bitmap index table, and then accommodating values based on the same format. This method includes the bitmap indexing method to solve the scoring problem and fasten the process.

First, an overview of the Single Machine algorithm which is the conventional way of using one machine to apply the TKD query on incomplete big data will be provided. Followed by the Single Machine algorithm, the proposed MRBIG algorithm will be introduced and discussed in-depth.

3.4.1 Single Machine Algorithm

For a given dataset R containing numbers of items in different dimensions, each dimension are represented as a group of values v. The value v_i represents the range of all numbers dimension i in R. For example, in Table 3.1, v_1 is $= (-, 1, 2, 3)$ which shows all of the present values in dimension d_1. In the provided example in Table 3.1, all v_i values are the same because all dimensions have the same range of present values. In the real-world problems, the benefit of having v_i is to give the power of creating dynamic bitmap index tables for each dimension d to save storage and improve performance.

The bitmap index creates separate columns for each value in v_i. Having v_i gives a full representation of present numbers in each dimension among all items. The bitmap index table would be initialized to 0 and then based on each item we would modify the values in the bitmap index table as described below.

- For each missing value, leave the fields in the corresponding row without any changes. So each v_i with a missing value remains as all 0s.
- For each number we have in v_i, insert number 1 in the corresponding row, and all of the following right rows.

For a better illustration of what the values would be after modifying the bitmap, suppose we consider all v_i values for the item in the Sample Dataset in Table 3.1. As can be seen, the items have four different values overall. Missing values and also the numbers 1, 2 and 3 are among the members of v_i for this dataset. These three are among all of the possible values in the dimensions of items in R. The generated Bitmap Index table can be seen in the Table 3.5, by having dataset R.

Using a single computing machine to execute the process of Algorithm 1 and creating bitmap index tables for incomplete data is not possible while dealing with large datasets (with hundreds of thousands of columns and rows). The mentioned approach and Algorithm 1 works with its best performance when the data does not exceed specific thresholds. Otherwise, massive run-time and storage are required to run the TKD query using the mentioned algorithm.

By keeping the mentioned problems as an incentive, the proposed algorithm is an effort to make the process of finding top-k dominant values in incomplete data efficient in time and storage while dealing with large files. To avoid redundancy, P and Q sets are defined in the later sections.

Items	d_1	–	1	2	3	d_2	–	1	2	3	d_3	–	1	2	3	d_4	–	1	2	3
m_1	–	0	0	0	0	1	0	1	1	1	2	0	0	1	1	–	0	0	0	0
m_2	1	0	1	1	1	–	0	0	0	0	3	0	0	0	1	2	0	0	1	1
m_3	3	0	0	0	1	1	0	1	1	1	–	0	0	0	0	–	0	0	0	0
m_4	–	0	0	0	0	–	0	0	0	0	–	0	0	0	0	1	0	1	1	1
m_5	–	0	0	0	0	2	0	0	1	1	1	0	1	1	1	–	0	0	0	0
m_6	–	0	0	0	0	–	0	0	0	0	–	0	0	0	0	3	0	0	0	1
m_7	1	0	1	1	1	1	0	1	1	1	–	0	0	0	0	–	0	0	0	0
m_8	–	0	0	0	0	3	0	0	0	1	2	0	0	1	1	–	0	0	0	0
m_9	2	0	0	1	1	–	0	0	0	0	2	0	0	1	1	2	0	0	1	1
m_{10}	3	0	0	0	1	2	0	0	1	1	–	0	0	0	0	–	0	0	0	0

Table 3.5: Bitmap Index Table for the Sample Dataset

3.4.2 MapReduced Modified Algorithm

As the single machine procedure is not capable of dealing with big datasets, another method is required to enhance this approach. The single machine procedure (referred as the BIG algorithm) provides strong performance for small datasets, but as the size of dataset increases, the performance weakens. So, another notion is imperative to make the BIG algorithm possible on big files.

Incomplete big data is ubiquitous these days, and TKD query processing on this datatype needs to be addressed. In this paper, we propose the MRBIG algorithm, which enables us to apply TKD queries on incomplete big data using MapReduce framework. The complete explanation and our proposed algorithm have been represented in the following Chapter 4.

Algorithm 1 Single Machine(BIG) Pseudo Code

Score calculation for item m_i
Create [P] and [Q]
for each item m_i **do**
    for each column in v_i **do**
        temp $\leftarrow$ the first 0 in the m_i-th row
        ind $\leftarrow$ Indext(temp)
        [P] $\leftarrow$ append($\sum_{j=1}^{n}$ [m_j, ind])
        [Q] $\leftarrow$ append($\sum_{j=1}^{n}$ [m_j, ind+1])
        nonD(m_i)
        $P \cap P^*$
        $Q \cap Q^*$
    end for
    $\alpha = P^*$ - Q^* - $nonD$
    $\beta = count(P^*-)$
    $score = \alpha + \beta$
    $maxscore[i] \leftarrow score$
end for
finish when length($maxscore$) = n
Finding Top-k
sort(descending ($maxscore$))
return top-k

Chapter 4

MRBIG: MapReduce Enhanced Bitmap Index Guided Algorithm

4.1 Overview

MapReduce framework evolved when data became too large for machines to process. The processing time of some tasks can take months using a single machine and efforts to make it as powerful as possible were too improbable and expensive. As mentioned before, MapReduce framework has two fundamental functions called Mapper and Reducer, which are used to separate huge tasks between multiple nodes to make them faster. Each task applied to MapReduce framework splits into different chunks based on internal patterns and gets divided between nodes. After assigning data fractions to Mapper, they process each piece and return the output. Later, the Mapper results are converged to calculate the final result using the Reducer.

In the MapReduce framework, the communication time is one issue that has to be considered. Synchronizing the nodes and making them informed about the status of tasks, in addition to sending and receiving data from them, are major factors that can impact communication cost in MapReduce. There are vast areas that MapReduce can be applied to enhance performance. Time inefficiency in top-k dominance is one major issue that this paper tries to address by using MapReduce framework. The majority of complex problems such as text processing systems and data mining technologies are handled using MapReduce framework that helps to make the systems act more real-time and reveal outputs in small time fractions.

One generic MapReduce example is counting the frequency of words in large text files. In this example, MapReduce calculates all of the numbers of word appearances in a particular large

Apple	Orange	Mango
Orange	Grapes	Plum
Apple	Plum	Mango
Apple	Apple	Plum

Mapper1	Mapper2	Mapper3	Mapper4
Apple, 1	Orange, 1	Apple, 1	Apple, 1
Orange, 1	Grapes, 1	Plum, 1	Apple, 1
Mango, 1	Plum, 1	Mango, 1	Plum, 1

Figure 4.1: MapReduce Framework Example Top: Dataset; Bottom: Mapper Results

text file. The dataset first splits into different slices that each contains a smaller portion of the original dataset. Each piece of the dataset goes to a different Mapper, and each Mapper performs the same process of word counting. In this manner, Mapper generates a simple line by line output wherein each line contains the word itself and the number of its appearance as shown in Figure 4.1.

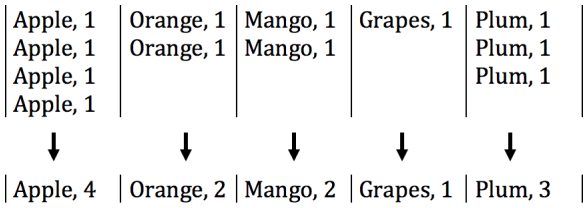

Figure 4.2: Top: Results after Sort-and-Shuffle; Bottom: Reducer appended results

The process starts with the first word of the document and creates a line that contains the word and the number of its appearance. The output is sorted and cleaned, which is done by the framework automatically using Sort-and-Shuffle. Sort-and-Shuffle aligns the words in the separated places and put them together. This internal function of MapReduce framework make the results prepared for the Reducer (Figure 4.2). After having the results of Sort-and-Shuffle, Reducer appends all the results and converges the mapper results to make the final result. The Figure 4.2 shows the sorted and shuffled result and the Reducer result at the bottom.

MapReduce framework is a fast procedure for dealing with big datasets by using simple programming methods, but it still has some defects. The amount of time which is required to synchronize the data in different nodes with each other is a factor of efficiency. Network congestion and delay are among the important factors which cannot be left unconsidered. Also, recovering from

error is another aspect of MapReduce which can worsen performance. If a node gets disconnected or if some error happens that makes the node stop, data recovery or node suspension for continuing the process can add up some overload in timing. However, MapReduce framework now can handle most of the issues autonomously.

MapReduce framework does not require a high volume of processing and the results can be calculated rapidly while dealing with big files and datasets. There is no significant change between using the Hadoop MapReduce framework or traditional single machine code in the small cases. In the small-size cases, using the single machine can be more effective. Spending no time for synchronizing the computing nodes, exchange the metadata. Not having those parameters make the processes faster in small-size datasets. However, the single machine procedure is not helpful with big datasets as the number of calculations and comparisons increase exponentially.

Based on the characteristics mentioned above and by having large datasets, MapReduce can be a reliable method to implement our proposed algorithm for finding top-k dominant values in incomplete big data. Hadoop clustering method used for implementation and the mathematically proven lemmas have been provided to evaluate the effectiveness of the MRBIG algorithm.

4.2 Implementation and Preliminaries

To start implementing the MRBIG algorithm, preparing the dataset and pre-formatting are vital steps to ensure the accuracy of the input dataset. Bitmap indexing is another aspect of implementation that has to be defined. A brief description of making the Bitmap Index table has been provided in the Section 3.4.1, but in this section, we provide more explanation to illustrate the concept.

4.2.1 Bitmap Indexing

Dataset R is a given multi-dimensional incomplete big dataset that has m items and d dimensions. The size of the dataset is million times larger than the previous sample datasets used by the single machine approach. Throughout the dataset R, a two-dimensional matrix can accommodate rows and columns in itself. As discussed in Section 3.4.1 and shown in Table 3.5, v_i provides a range of all present values in dimension i, if each column is considered separately, a range of all present values in that specific column can be found. Having the v_i helps us to construct the bitmap index table for the algorithm and gives the power to create dynamic bitmap index tables for based on the

dataset values. For instance, $v.[6] = [-, 1, 2, 5]$ shows that in the 6_{th} dimension of data all present values are missing values and 1, 2, 5. This helps us to construct the bitmap index table smarter and tells us the number of columns required for the bitmap index table for each dimension.

By having v_i for each dimension, Bitmap Index table can be constructed. By having d as dimensions and v_i there exists the following condition and bitmap index table can be initialized after this step:

$$\sum_{i=1}^{d} v_i \tag{4.1}$$

To generate the bitmap index table to continue the steps of the MRBIG algorithm, as can be seen in Table 3.5, the table would be initialized to 0. By following the filling-up rules the proper values are inserted into the table.

As mentioned earlier, there are two rules to follow for filling up the values in the Bitmap Index Table. First, for each missing value, we leave the fields of that row and the range of columns that spans without any changes. Second, for each non-missing value in v_i, the corresponding field and the following rightmost fields changes to 1.

By repeating the mentioned process for every object and dimension, Bitmap Index Table will be generated. In Section 3.4.1, a bitmap index table representation has been provided to clarify the concept. Bitmap index table will be used to use the table to run the algorithm and find the top-k dominant value(s) in any incomplete dataset.

4.2.2 Internal Structure of MRBIG

To evaluate the values based on a powerfulness which is defined by the dominance definition, a scoring method is vital to examine values in a comparable way based on the dominance definition. Having scoring method helps us to enable comparing values by their corresponding score they acquire throughout the dataset. The more value is good based on the dominance definition; the higher score will earn. The scoring method has been modified to match and take advantage of the MapReduce framework structure. The pre-defined scoring function is embedded into MRBIG algorithm.

Based on MapReduce characteristics, the most logical approach is to calculate the score for each particular dimension. The mapper component in the MapReduce calculates the score as it always accommodates a fixed amount of dimensions.

There are also three internal sets that form the MRBIG algorithm and helps us to acquire the top-k dominant values in a dataset for each object m.

One of the important sets that have a significant role in the algorithm is called $[Q]$. The $[Q]$ is defined as a set of objects which is not better than m or the values missing excluding m based on the Dominance Definition in that particular dimension. The other set $[P]$ is a subset of $[Q]$ and can be defined as a set of objects that are strongly worse than object m or missing from that specific dimension. The last and one of the most important components of the MRBIG algorithm is the $[nonD]$ set, as it can be implied from its name, demonstrates the set of objects that are not dominated by the considered object m.

To provide a better understanding, suppose a movie recommender system that calculates the most popular movies among users' ratings. Suppose that the approach is to calculate the score of each dimension in the Bitmap Index Table (sample shown in Table 3.5) by using one Mapper for processing each dimension. The Mapper calculates the candidate sets $[Q]$ and $[P]$ for each processed dimension which is useful to find the top-rated movies of a particular user. After Mapper calculations, Reducer intersects the result sets of Mapper which led to $[Q^*]$ and $[P^*]$ respectively. Furthermore, the $[nonD]$ calculation takes place after finding the $[Q^*]$ and $[P^*]$ values. Finding the $[Q^*]$ and $[P^*]$ would be the main objective of each MapReduce process, and the score calculation of top-k values from one specific dimension takes place in the rest of the MRBIG algorithm.

One of the main requirements for applying the TKD query is to have the score for each item in the dataset. The main objective of the MRBIG algorithm is to help to make the score calculation of each item as fast as possible by taking advantage of MapReduce framework. The scoring method is a key feature that can affect the algorithms performance strongly. In this context, the score is a criterion that shows how powerful an item must be to be a top-k dominant value. This value would be calculated based on how many items an item is dominating and the ratings for that item. The highest scores would be considered as the answer to the TKD query.

Finding the score of each item requires a look at the whole dimensions and finding the $[P]$,$[Q]$, and $[nonD]$ sets for each dimension in a Mapper, and later on append all of them to find the final score of the item. Therefore, for each item, one complete review of all the dimensions is required. Later, an outer loop is used to follow this procedure and find all of the scores for each object. The scores are stored in $maxscore$, which is a complete data structure containing the scores of all items. By looking at the top values the TKD final answer of the large incomplete input dataset be found.

To clarify the MRBIG procedure and the whole TKD process concept so far, we review an example by looking to find the score of the item m_4 based on the Table 3.5. MRBIG algorithm starts to send each dimension to one Mapper and calculate $[P]$ and $[Q]$ sets from each dimension. $\sum_{i=0}^{4} = P_i$ and $\sum_{i=0}^{4} = Q_i$ are the answers from all of the Mappers which are available for the Reducer to process. It intersects the sets together and calculates the $[P^*]$ excluded the zero values ($\emptyset$) which mostly helps us to eliminate any values that are incomparable and $[Q^*]$. Then, the Reducer calculates the α and β values as can be seen in Algorithm 2 and appends the score of m_4 to the *maxscore* data structure to its corresponding index that represents m_4 score. By repeating this process for each item we reach to a complete *maxscore* data structure containing all of the scores, and at this level by sorting the *maxscore*, finding the top-k dominant values are reachable.

Our approach in the MRBIG algorithm is based on numerous mathematical assumptions. It is worth mentioning the mathematical proofs for each of the processes taking place during the transition to MapReduce framework. The method of candidate sets calculations for $[P]$ and $[Q]$ are the major differences between Single Machine and MapReduce approach. MRBIG Algorithm splits the sets to make the process faster and send each portion to different computing nodes, while the single machine uses one calculated $[P]$ and $[Q]$ and proceed to the rest of the algorithm. These different ways of $[P]$ and $[Q]$ calculations have been proven mathematically that results to the same output.

Lemma 4.2.1. *Let* $[P] = \bigcap_{i=1}^{d} [P^i]$ *where each* $[P^i]$ *is a binary string of length* n*. Let* $P_1, P_2, ..., P_k$ *be an arbitrary partitioning of the set of all* $[P^i]$*. If* $[P^*] = \bigcap_{j=1}^{k} (\bigcap_{P^m \in P_j} [P^m]$*, then* $[P] = [P^*]$*.*

Proof. Intersection (or bitwise AND) is associative, so the order of intersection does not matter. □

It is clear that $[P^*]$ and $[Q^*]$ are sets that have been aggregated from other smaller ones which come from computing nodes and get intersected in a master node. The same naming convention is followed for sets $[P]$ and $[Q]$ for Single Machine resulting sets and $[P^*]$ and $[Q^*]$ for MapReduce resulting sets. Lemma 4.2.1 shows a proof that for $[P^*]$, based on the definition for $[P]$, it is possible to claim that resulting $[P^*]$ from MRBIG algorithm (Algorithm 2) is mathematically similar to the resulting $[P]$ of Single Machine Algorithm (Algorithm 1).

Following the same explanation for $[Q]$ candidate set, each $[Q]$ which has been calculated by Single Machine Algorithm is similar to $[Q^*]$ candidate set coming from MRBIG Algorithm. In other words, there are several $[Q]$ sets that have been calculated in different computing nodes for MRBIG Algorithm.

By having three computing nodes, each node calculated its own $[Q_i]$ set based on the in-hand fraction of the input file, and eventually, the resulting sets after mapper processing are $[Q_1]$, $[Q_2]$ and $[Q_3]$ for each mapper respectively. The MRBIG algorithm intersects the sets by $\bigcap_{i=1}^{3}[Q^i]$ and reach to $[Q^*]$. Lemma 4.2.2 shows that $[Q^*]$ equals $[Q]$, which $[Q^*]$ is the resulting set of MRBIG algorithm and $[Q]$ is the resulting set of Single Machine approach.

Lemma 4.2.2. *Let* $[Q] = \bigcap_{i=1}^{d}[Q^i] - \{o\}$ *where each* $[Q^i]$ *is a binary string of length* n *and* o *is the current object. Let* $Q_1, Q_2, ..., Q_k$ *be an arbitrary partitioning of the set of all* $[Q^i]$*. If* $[Q^*] = \bigcap_{j=1}^{k}(\bigcap_{Q^m \in Q_j}[Q^m]) - \{o\}$*, then* $[Q] = [Q^*]$.

Proof. By Lemma 4.2.1, we know $\bigcap_{j=1}^{k}(\bigcap_{Q^m \in Q_j}[Q^m]) = \bigcap_{i=1}^{d}[Q^i]$, subce each $[Q^i]$ is just a binary string. Subtracting o from both sides gives the desired result. □

By using Lemma 4.2.1 and Lemma 4.2.2, it is proven that the resulting sets are mathematically correct and the approach in MRBIG algorithm can be genuinely referenced and followed.

Algorithm 2 MRBIG Algorithm (Having n items and r dimensions)

Create $[P^*]$ and $[Q^*]$
for item m_i in $\{m_1, m_2, ..., m_n\}$ **do**
    Mapper:
    Map(split($\sum_{j=1}^{r} d_j$)) //Spliting by dimensions
    Bitmap(d_j)
    for each dimension d_j **do**
        Create $[P_j]$, $[Q_j]$ and $nonD_j$
        for each v_{d_j} in m_i-th row **do**
            Candidate $\leftarrow$ index(if $(m_i, v_{d_j}) == 0$)
            $[P_j] \leftarrow$ append($\sum_{i=1}^{n}$ [m_i, Candidate])
            $[Q_j] \leftarrow$ append($\sum_{i=1}^{n}$ [m_i, Candidate+1])
        end for
    end for

    Reducer:
    for $\sum_{1}^{r} i$ **do**
        $([P_i] \cap [P^*]$
        $[Q_i] \cap [Q^*]$
    end for
    $\lambda = [Q^*] - [P^*]$
    for each i in λ **do**
        $\phi =$ count (if $\lambda_i \not\prec m_i$)
        $nonD \leftarrow \lambda_i$
    end for
    $\alpha = |\lambda - nonD|$
    $\beta = count(P^* - \emptyset)$
    $score = \alpha + \beta$
    $maxscore[i] \leftarrow score$
    finish when length($maxscore$) $= n$

end for
Finding Top-k
sort (descending ($MaxScore$))
return top-k

Chapter 5

Experiments and Analysis

This section goes through the effectiveness of the BIG and MRBIG algorithms and a comparison between their performance using different real and synthetic dataset models. The experiment starts with performance evaluation of Single Machine algorithm and follows through the assessment of the proposed MRBIG algorithm based on required metrics analysis such as CPU Time, item frequency, dimension fluctuations, and missing rate.

The real dataset originates from an authentic user-initiated source that can help us evaluate the MRBIG performance in a real-world problem. The *MovieLens* dataset obtained in different sizes for testing and analysis purposes. The number of present values in the dataset is what distinguishes between the various real datasets and using different sizes give the most detailed overview of the MRBIG algorithm effectiveness. To provide more in-depth analysis of MRBIG performance, diverse groups of synthetic datasets is used. Synthetic datasets' generation has various mathematical and statistical metrics to help highly examine the MRBIG algorithm.

The experiments conducted in computing nodes equipped with one core of *Intel*® *Xeon*® E5-2695 v4 @ 2.10GHz CPU with 32GB assigned RAM and running Linux *Ubuntu*® 16.04 LTS operating system. For our parallel computing purposes, we have used four computing nodes each equipped with the same configuration as mentioned earlier.

Based on different incomplete data inputs and its volume, the number of computing clusters can be easily adjusted. Based on the data in this context, the number of computing cluster is believed to be optimal. Having big data requires the computing nodes to have a significant amount of RAM available for the algorithm. This will help to be able to initialize and process the necessary components such as the bitmap index table.

MRBIG parallel computing runs through three slave computation nodes in addition to

a master node as a hub point for the others. The single machine code (Algorithm 1) runs on the master node with the configurations as mentioned earlier, and the MRBIG processing (Algorithm 2) takes place on *Apache Spark* configured clusters pre-built for *Hadoop* 2.7 with the mentioned configurations.

5.1 Data Information and Preparation

The *MovieLens* datasets include a range of movies with the submitted ratings for them from a range of users. This dataset can be used to build a movie recommender system that can dynamically evaluate the influential movies to design smart systems. Sizes of the dataset can be seen in the Table 5.1 in which the numbers depict the frequency of present values in each dataset. The smallest dataset containing $100k$ values has around $1.7M$ values and the $20M$ version which is one of the biggest real datasets over $138,000$ users and around $27,000$ movies.

Name	No. of Users	No. of Movies
$100k$	$1,000$	$1,700$
$1M$	$6,040$	$3,706$
$10M$	$71,000$	$11,000$
$20M$	$138,000$	$26,000$

Table 5.1: Real *MovieLens* Dataset Information

5.1.1 Evaluation and Creation

Other than the size of the data, there are several other parameters of data that has been considered for analyzing the algorithms such as deviation of ratings, missing rate, mean ratings, etc. These parameters help examine the dataset carefully by having detailed information about each dataset. The artificially generated datasets have been created very carefully by keeping track of 6 different manually-defined parameters to preserve the accuracy of the experiments. These components are such as the missing rate, average value for inserted numbers, and standard deviation threshold. Also, standard missing rate deviation, the standard deviation for each dimension which depicts the values discrepancy for each dimension are among the other parameters for the synthetic data.

Eventually, the standard deviation for the standard deviation for each dimension is the last parameter that assigns the difference of deviation between different items for dimensions deviation. For the real dataset, not all of the mentioned parameters get tracked as it is published as a real-world data and such information is not available. By defining the parameters mentioned above,

we can procure the most productive information to see the efficiency of the MRBIG algorithm and provide a way to elicit the most accurate experiments, which can bring innovation and speed to the systems.

5.1.2 Pre-formatting and Preparation

To make the different datasets uniform and compatible with our desired input for the experiments, we have developed a formatting process. As discussed, the real dataset is from the publicly available *MovieLens* dataset. Each value in this dataset has three different characteristics including movie identification, user identification, and the rating submitted. A small sample of the raw input is depicted in Figure 5.1. A formatting process needed to empower the experiments accuracy and making the data suitable for the Hadoop MapReduce framework. As you can see in Figure 5.1, user 101 has submitted four ratings for movies $01, 02, 04, 05$, but not any rating for movie 03. Not having rating submission for movie 03 indicates a missing value and no row exists for this particular value. After pattern change occurrence, instead of having multiple lines for each object, one line represents user 101 and the ratings of this user as seen in Figure 5.1.

UserID	MovieID	Rating
101	01	3
101	02	2
101	04	1
101	05	3
236	03	4
236	05	1
87	01	4
512	01	2

	01	02	03	04	05
101	3	2	-	1	3
236	-	-	4	-	1
87	4	-	-	-	-
512	2	-	-	-	-

Figure 5.1: Sample input formatting conversion for the MRBIG algorithm. Top: Raw Input; Bottom: Processed Input

Further formatting process involves operations to make the data compatible with the algorithm, such as removing special characters and separators and gives the prepared data, ready to be used by the MRBIG algorithm.

There are multiple reasons for having a pre-processing method. Comparing uniform datasets with the same format to make the data analysis accurate is one the reasons. Not tracking

the spent times for arranging or formatting the dataset is another reason which helps to provide valid comparisons. This process helps us to make implementation of the algorithm easier and ensures our comparison and analysis are logical and correct. The experiments on MRBIG and BIG algorithm using real data in Table 5.1 and synthetic data with the details found in Table 5.2 all follows the same formatting style.

No. of Users	No. of Movies	No. of Users	No. of Movies
500	3,500	6,000	500
1,000	3,500	6,000	1,500
2,000	3,500	6,000	2,500
3,000	3,500	6,000	3,500
4,000	3,500	6,000	4,500
5,000	3,500	6,000	5,500
7,000	3,500	6,000	6,500
8,000	3,500		

Table 5.2: Artificial Dataset Information

The synthetic data contains different missing rates, and the experiment has been done using various combinations of objects in several dimensions in different ranges such as 2,000 6,000. The synthetic datasets use a standard deviation for creating numbers for each object. Each object is created based on different parameters such as missing rate, deviation function, and the pre-defined mean value. The parameters are defined before generating the datasets and has been described previously in the context. These datasets have been created to cover all of the experimental purposes and testing the MRBIG performance.

5.2 Algorithm Developement and Evaluation

MRBIG algorithm implementation methods can be logically and fundamentally different based on the needs. Hence, different results may not indicate consistent approaches for applying the TKD queries on incomplete data.

Our implementation process goes more in-depth to analyze the full available information in each dataset. MRBIG algorithm calculates the final score for every single object in the dataset. Furthermore, MRBIG algorithm stores all objects' score values in a permanent structure which can be utilized even after the termination of the algorithm. The final scores are available to retrieve k objects after running the algorithm. This helps to acquire the TKD application result faster for later use.

By considering alternative ways of reducing the run-time, it is also possible to pre-define the k value as some objects which have to be returned by MRBIG as top-k dominant values by performing some modifications to the algorithm. The approach that MRBIG follows helps to recall the results faster for later needs.

However, using a pre-defined k value helps to execute the algorithm much faster when finding the top-k dominant value, but not in all and any case. Making a selection between using either a pre-defined k value for running the algorithm or implementing the algorithm using the overall scoring procedure is not easy and clear. Therefore, there is no perfect answer or approach for using which k calculation method. There are however several advantages and disadvantages associated with each method.

By pre-defining the k, multiple executions of MRBIG is required for every desired top-k value. In some cases that dataset changes more often. This method may be helpful as running the algorithm would be performed intermittently to find new answers for the TKD query after the data is changed. On the other hand, if the score calculation covers the whole dataset like the MRBIG approach, after the initial implementation of the algorithm, the scores are permanently present and can be recalled whenever required. This approach would help the enterprise systems to retrieve answers for different top-k values in milliseconds just by looking up the scores and retrieving the appropriate dominant values based on the top-k results.

As shown in Figure 5.2, by using the real datasets, the performance of the MRBIG algorithm is directly dependent on the size of the dataset. When using small data sets in MapReduce framework, communication cost spent for synchronizing nodes and sending and receiving data leads to the slower run-time for the algorithm. On the contrary, by increasing the size of the dataset, using the Algorithm 1 exhausts the resources and make the process exponentially slow while MRBIG shows a high performance in handling a large flow of data and speeding up the process.

MRBIG algorithm does not show a huge difference in compare to the Bitmap Index Guided algorithm while facing smaller sizes of Synthetic data. The improved performance of MRBIG algorithm has been examined by both items size and dimension size for the incomplete synthetic datasets as shown in Figure 5.3 and Figure 5.4. However, as soon as the data size increases, the MRBIG performance becomes more and more useful and time efficient. Our experiment shows that this difference can be more than two times faster in speed.

As can be seen in MRBIG Algorithm (Algorithm 2), the overall execution of the algorithm can be considered as two major parts that handle the process of finding TKD query on incomplete

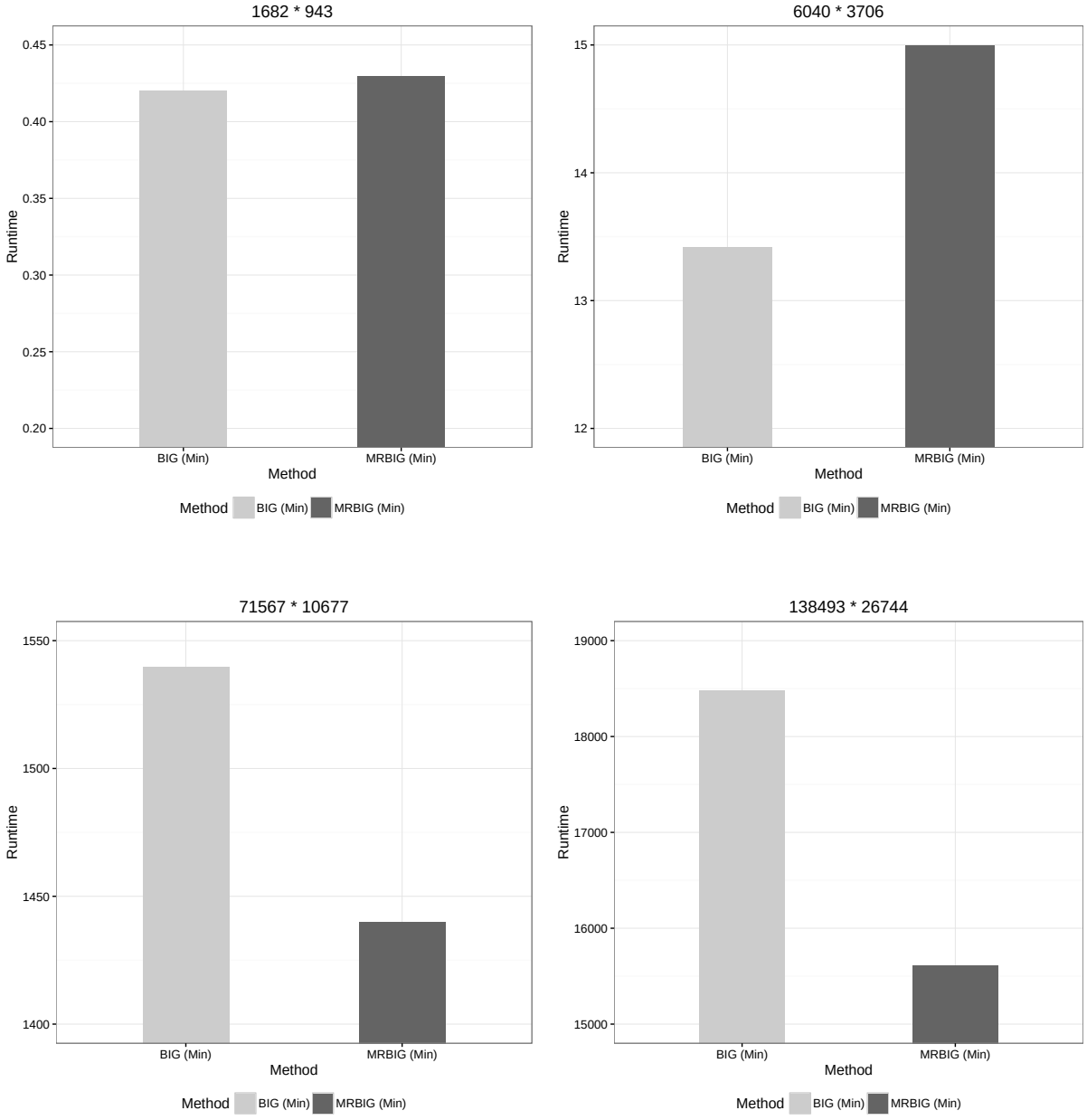

Figure 5.2: Simulation results of comparing BIG and MRBIG algorithms based on real MovieLens dataset in uniform configured clusters

data. The first part includes operations that take place in the MapReduce clusters. This part includes those calculations that have been transferred to clusters to speed up the process by splitting the workload between different computing nodes.

The second part that has a minimal role throughout the algorithm includes those operations which remain in the single machine procedure. These are not processed by either Mapper or Reducer, and the calculations are based on the conventional single-machine method.

The first component of the algorithm is where the efficiency starts to improve and enhances the performance. Based on different workloads and desired outcomes, the number of clusters can be adjusted. Adjusting the clusters by increasing or decreasing the number of computing nodes, can have different effects. Based on the volume of data, having more cluster might increase the runtime and worsen the performance. Based on the synthetic data and real datasets in this context, having four clusters is considered as an optimal frequency.

Parallel computing approaches, always have a special point that can affect the results badly. The network structure is a vital part of the process. Based on the network status and the queues and congestion, dropping performance is always expected. As it can be seen in the Figure 5.3 and Figure 5.4, the MRBIG algorithm has always been able to provide a stable performance throughout the different size by an optimal amount of increase in the runtime.

In the conducted experiments, garbage collection is also included in the runtimes which are about 20 to 25 percent of the elapsed time. Each portion of data would be assigned dynamically to different clusters by controlling the specific number of *dimensions* assigned to each mapper. The results of each mapper which includes the $[P]$, $[Q]$ sets for each assigned dimensions(s) would be aggregated (As depicted in Algorithm 2)finalized in the reducer. The final parts are mostly done on the master node.

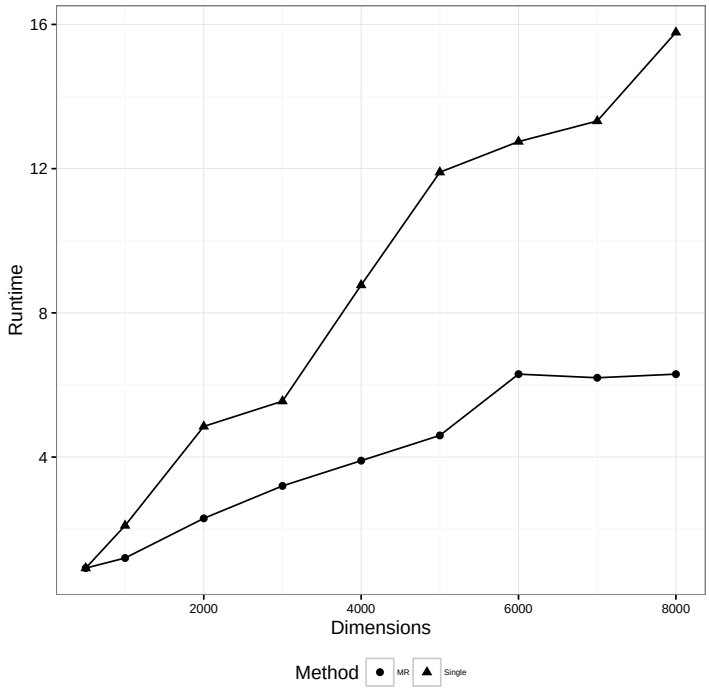

Figure 5.3: MRBIG algorithm simulation results based on synthetic incomplete data with different numbers of dimensions comparing BIG algorithm and MRBIG algorithm in uniform configured cluster

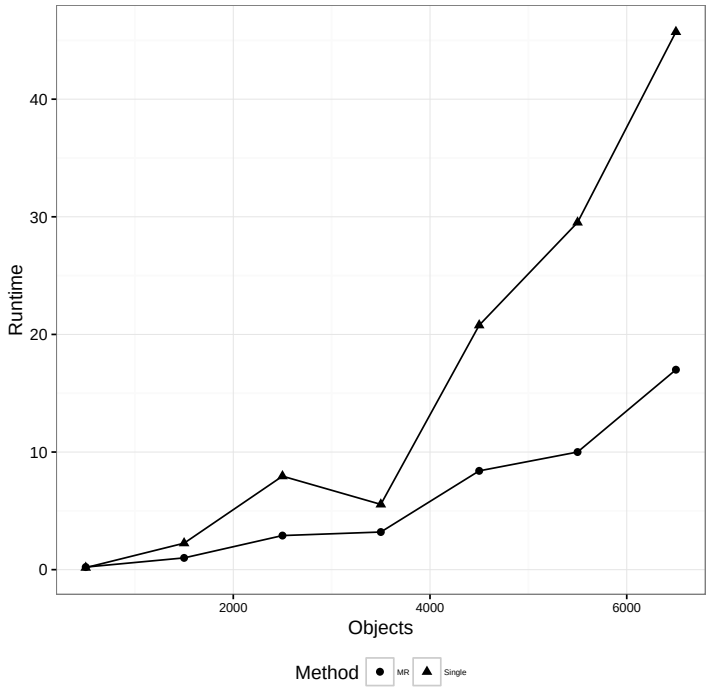

Figure 5.4: MRBIG algorithm simulation results based on synthetic incomplete data with different number of items (objects) comparing BIG algorithm and MRBIG algorithm in uniform configured cluster

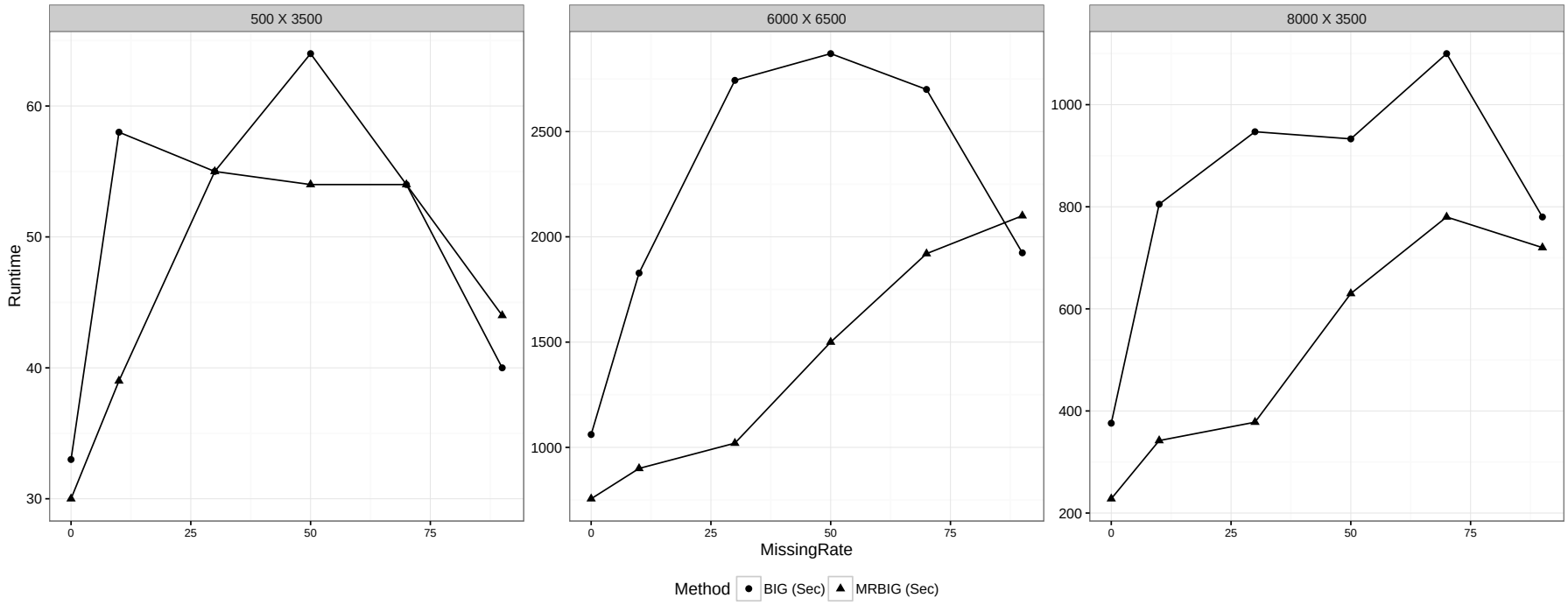

Figure 5.5: MRBIG algorithm simulation results for different missing rates available on data compared to BIG algorithm. Missing Rates ranges from 0% to 90%.

The missing rate is another factor that can affect the performance of the MRBIG algorithm. Having less missing rate inclines the datasets to be closer to complete. The experiments show that ranging missing rates can influence the behavior of the MRBIG algorithm as well.

The range of measured missing values varies from 0 to 90 percent, as shown in Figure 5.5. Different sizes of synthetic data help to get a better insight of the MRBIG algorithm that has been represented in Figure 5.5. By keeping the deviation components to similarly evaluate the performance, multiple synthetic data generated to help identify the MRBIG performance regarding different missing rate.

The first zero percent missing rate is representing a complete data. In those datasets, the MRBIG algorithm still goes through the same procedure although there are no missing values. However, MRBIG is not inherently designed to handle complete datasets; therefore, the performance improvement is not significant in that cases as experiments show.

Also, by reviewing the evaluations on the higher missing rates that conclude to more suitable datasets for MRBIG evaluation, the performance enhances and the runtime decreases. The significant difference of performance improvement helps the recommender systems to retrieve the desired answers faster.

By carefully reviewing the results of the MRBIG algorithm, it can be inferred that applying TKD query on small incomplete datasets using MRBIG can result differently. However, as the size of the dataset increases, the performance of the algorithm becomes more stable, and fluctuations in the processing runtime are much less than smaller datasets.

For further evaluating the MRBIG algorithm and apply more experiments, various synthetic datasets has been generated as shown in Table 5.2. The results of using the mentioned synthetic incomplete datasets on MRBIG algorithm can be seen in Figure 5.6.

In the same way, the synthetic data shows that the performance of applying TKD on synthetic data is highly dependent on the size of the dataset. By changing the number of dimensions, MRBIG becomes faster, and this pattern will continue by changing the number of objects (or items) as well.

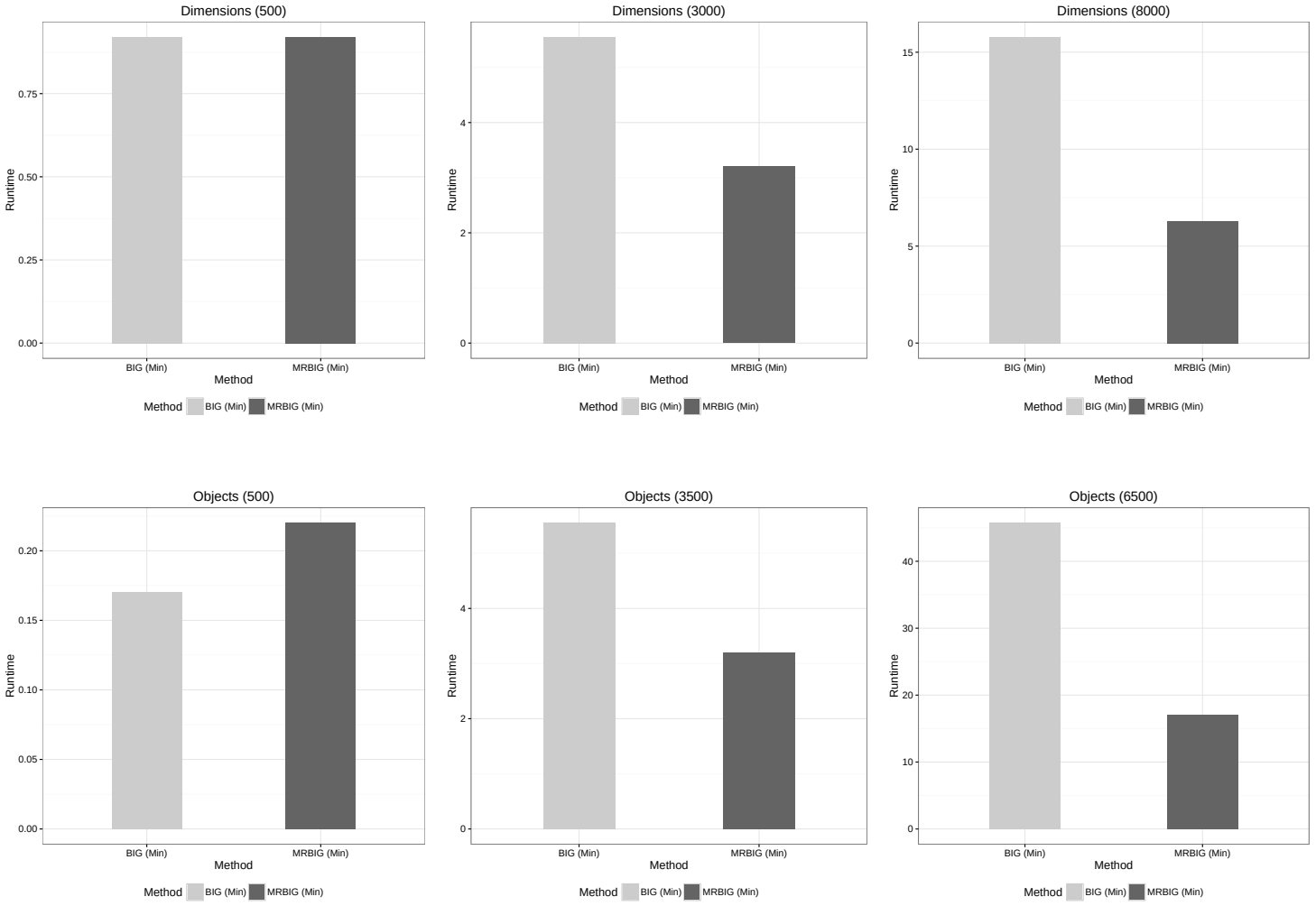

Figure 5.6: Simulation results of comparing BIG and MRBIG algorithms based on different synthetic dataset in uniform configured clusters

Chapter 6

Conclusion and Future Work

In this thesis, we proposed an algorithm to apply top-k dominating queries using MapReduce framework on incomplete big data. MapReduced Enhanced Bitmap Indexed Guided algorithm (MRBIG) is the basis of the work that develops a new way to handle large incomplete data and uses the MapReduce framework to enable parallel computing to manage the problem faster.

Throughout the context, the Single Machine algorithm has been reviewed as well as the MRBIG structure and the algorithm. Based on the experiments, Single Machine algorithm cannot be an optimal way for applying TKD queries on big files. Not being resource-efficient, process failure due to resource insufficiency, and having exponential processing time are among the major defects when it comes to finding top-k dominant values in massive incomplete data.

MRBIG algorithm is a faster way to process incomplete big datasets while carefully managing the machine resources and maintaining time efficiency simultaneously. The MRBIG algorithm provides strong performance, and in most of the cases, it is two or more times faster than the single machine procedure. The results and experiments have also been considered in Chapter 5 to provide an overview of the efficacy and consistency of the approach.

BIG and MRBIG are both considered as a resulting method from Skyband based algorithm and Upper Bound Based algorithms that use their key advantages to designing better algorithms. Skyband based and Upper Bound based algorithms reviewed in previous sections. Skyband based algorithm promotes a notion of data bucketing to normalize the data. The similar way has been proposed in MRBIG as the bitmap indexing method. Using bitmap indexing helps to deal with big data to reduce and speed up the pair-wise comparison by representing binary strings for each value in the bitmap index table.

The contribution follows to Upper Bound Based by having the scoring methods embed-

ded into the MRBIG and BIG algorithm that evaluate the fields independently and assigns the appropriate score to each area. Having a scoring method can be considered as continued work on UBB algorithms. Scores help to retrieve desired top-k based on the demand much faster and in more real-time manner.

MapReduce framework has different logical approaches to handle TKD query processing. The assumptions for those logical approaches promoted for creating $[P^*]$ and $[Q^*]$ can be mathematically proven using lemmas provided in the context. Furthermore, having different structural implementations such as using a pre-defined top-k value and generating the proper score disregarding the other items scores can lead to different results on the performance of MRBIG. This notions and modifications can later be addressed to either improve the performance more or to provide better recommender systems based on needs.

Following those as mentioned above, it worth to mention some logical approaches that can be followed by future works by using the same notion behind the MRBIG:

- Separating the dataset using the mappers that use specific chunks of *objects* for performing TKD. In this case, we split a range of objects; then we apply the TKD query to them. To be clear, this means splitting the data by rows by taking a particular variety of rows, with every dimension included.

- Separating the dataset using mappers that split the dataset by a different range of *dimensions*. This method uses specific dimensions from every object and tries to obtain candidate sets while applying the TKD query. As seen on Algorithm 2 (MRBIG), it can be said that the dataset would be separated by different ranges of columns, which is having some dimensions of all objects and performing the TKD query on each of them.

- Separating larger datasets than what we have considered in this paper, by both *objects* and *dimensions*. In other words, we separate the data by rows and columns and obtain a specific range limited by objects and dimension, we perform the TKD query and return the candidate sets to the reducer(s).

By extracting the details for each aforementioned method, the most efficient MRBIG approach can be discovered based on the needs. We believe that the performances should not show a large difference, but the exact results and analysis help to identify the most helpful approach.

The analysis for MRBIG algorithm performance shows a significant improvement as reviewed in Chapter 5. But there are a few areas that can be addressed to enhance the performance

even more. One of these areas can be where the incomplete data has an enormous missing rate that gets so close to an empty dataset or having low missing rates that make the dataset too close to be considered as complete. It has experimented that having big incomplete data with a reasonable missing rate can have a substantially stable performance. However, In the two cases mentioned above, the MRBIG algorithm can be altered to provide better performance.

It worth to mention that Finding top-k dominance for incomplete data is one of the fields that has not been fully reviewed, and has different aspects that can be addressed as mentioned earlier. By emerging big data, MRBIG can be highly utilized to design recommender systems and provide a more real-time answer to TKD query processing.

Bibliography

[AKO07] Lyublena Antova, Christoph Koch, and Dan Olteanu. From Complete to Incomplete Information and Back. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 713–724, New York, NY, USA, 2007. ACM.

[BCZW14] Fanyu Bu, Zhikui Chen, Qingchen Zhang, and Xin Wang. Incomplete Big Data Clustering Algorithm Using Feature Selection and Partial Distance. *2014 5th International Conference on Digital Home*, pages 263–266, 2014.

[CJS+14] W Cheng, X Jin, J T Sun, X Lin, X Zhang, and W Wang. Searching Dimension Incomplete Databases. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):725–738, mar 2014.

[CWZ+15] Zhen Chen, Yuhao Wen, Wenxun Zheng, Jiahui Chang, Guodong Peng, and Yinjun Wu. A Survey of Bitmap Index-Compression Algorithms for Big Data. *Tsinghua Science and Technology*, 20(1):100–115, 2015.

[GLMC13] Shen Ge, U. Leong Hou, Nikos Mamoulis, and David W L Cheung. Dominance relationship analysis with budget constraints. *Knowledge and Information Systems*, pages 1–32, 2013.

[GMC+14] Yunjun Gao, Xiaoye Miao, Huiyong Cui, Gang Chen, and Qing Li. Processing k-skyband, constrained skyline, and group-by skyline queries on incomplete data. *Expert Systems with Applications*, 41(10):4959–4974, 2014.

[HLG15] Xixian Han, Jianzhong Li, and Hong Gao. TDEP: efficiently processing top-k dominating query on massive data. *Knowledge and Information Systems*, 43(3):689–718, 2015.

[HLLG16] Xixian Han, Xianmin Liu, Jianzhong Li, and Hong Gao. TKAP: Efficiently processing top-k query on massive data by adaptive pruning. *Knowledge and Information Systems*, 47(2):301–328, 2016.

[HMA09] Parisa Haghani, Sebastian Michel, and Karl Aberer. Evaluating Top-k Queries over Incomplete Data Streams. *Cikm*, pages 877–886, 2009.

[HPZL08] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. *Proceedings - International Conference on Data Engineering*, pages 1403–1405, 2008.

[IL84] Tomasz Imieliński and Witold Lipski Jr. Incomplete Information in Relational Databases. *J. ACM*, 31(4):761–791, sep 1984.

[KML08] Mohamed E Khalefa, Mohamed F Mokbel, and Justin J Levandoski. Skyline Query Processing for Incomplete Data. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 556–565, Washington, DC, USA, 2008. IEEE Computer Society.

[LC09] Xiang Lian and Lei Chen. Top-k Dominating Queries in Uncertain Databases. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 660–671, New York, NY, USA, 2009. ACM.

[LC13] Xiang Lian and Lei Chen. Probabilistic top-k dominating queries in uncertain databases. *Information Sciences*, 226:23–46, 2013.

[LEB13] Christoph Lofi, Kinda El Maarry, and Wolf-Tilo Balke. Skyline Queries in Crowd-enabled Databases. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 465–476, New York, NY, USA, 2013. ACM.

[Lib14] Leonid Libkin. Incomplete Data: What Went Wrong, and How to Fix It. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 1–13, New York, NY, USA, 2014. ACM.

[MBP06] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, page 635, 2006.

[MCYC06] Nikos Mamoulis, Kit Hung Cheng, Man Lung Yiu, and David W. Cheung. Efficient aggregation of ranked inputs. *Proceedings - International Conference on Data Engineering*, 2006(X):72, 2006.

[MGZ+16] Xiaoye Miao, Yunjun Gao, Baihua Zheng, Gang Chen, and Huiyong Cui. Top-k dominating queries on incomplete data. *2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016*, 28(1):1500–1501, 2016.

[NC16] Ha Thanh Huynh Nguyen and Jinli Cao. *Top-k Dominance Range-Based Uncertain Queries*, pages 191–203. Springer International Publishing, Cham, 2016.

[PFS05] Dimitris Papadias, Greg Fu, and Bernhard Seeger. *Progressive Skyline Computation in Database Systems*, volume 30. 2005.

[RN11] Simon Razniewski and Werner Nutt. Completeness of queries over incomplete databases. *Proc. VLDB Endow*, 4(11):749–760, 2011.

[SC14] Bagus Jati Santoso and Ge Ming Chiu. Close dominance graph: An efficient framework for answering continuous top-k dominating queries. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1853–1865, 2014.

[SIBD10] Mohamed A Soliman, Ihab F Ilyas, and Shalev Ben-David. Supporting Ranking Queries on Uncertain and Incomplete Data. *The VLDB Journal*, 19(4):477–501, aug 2010.

[TV14] Eleftherios Tiakas and George Valkanas. Metric-Based Top-k Dominating Queries. *Proc. 17th Inter- national Conference on Extending Database Technology (EDBT)*, pages 415–426, 2014.

[WLLW13] Yijie Wang, Xiaoyong Li, Xiaoling Li, and Yuan Wang. A survey of queries over uncertain data. *Knowledge and Information Systems*, 37(3):485–530, 2013.

[WOS02] Kesheng Wu, Ekow J Otoo, and Arie Shoshani. Compressing bitmap indexes for faster search operations. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 99–108. IEEE, 2002.

[WSS08] Kesheng Wu, Arie Shoshani, and Kurt Stockinger. Analyses of Multi-level and Multi-component Compressed Bitmap Indexes. *ACM Trans. Database Syst.*, 35(1):2:1—-2:52, feb 2008.

[YM09] Man Lung Yiu and Nikos Mamoulis. Multi-dimensional top-k dominating queries. *The VLDB Journal*, 18(3):695–718, 2009.

[ZZY13] Jinfang Zhang, Farong Zhong, and Zhenguo Yang. Efficient approach to top-k dominating queries on service selection. *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP*, pages 1–8, 2013.

Curriculum Vitae

Graduate College

University of Nevada, Las Vegas

Payam Ezatpoor

Degrees:

Bachelor of Science (B.Sc.) in Information Technology 2014

Sanandaj Azad University

Thesis Title: Finding Top-k Dominance on Incomplete Big Data Using Map-Reduce Framework

Thesis Examination Committee:

Chairperson, Dr. Justin Zhan, Ph.D.

Committee Member, Dr. Kazem Taghva, Ph.D.

Committee Member, Dr. Fatma Nasoz, Ph.D.

Graduate Faculty Representative, Dr. Ge Lin Kan, Ph.D.