International Journal of Database Theory and Application
Vol. 5, No. 2, June, 2012

Cloud Technology for Mining Association Rules in Microarray Gene Expression Datasets

Md. Rezaul Karim[1], A. T. M Golam Bari[1], Byeong-Soo Jeong[1] and Ho-Jin Choi[2]

[1]Department of Computer Engineering, Kyung Hee University, Republic of Korea
[2]Dept. of Computer Science,
Korea Advanced Institute of Science and Technology, Republic of Korea
{asif_karim, bari, jeong}@khu.ac.kr; [2]hojinc@kaist.ac.kr

Abstract

Microarray gene expression techniques and tools have become of a substantial importance and widely used to analyze the protein-protein interaction (PPI) and gene regulation network (GRN) research in recent years since it can capture the expressions of thousands of genes in a single experiment. Such dataset poses a great challenge for finding association rules in a faster way because of the presence of large number of columns but a small number of rows. Therefore, to meet the challenge of high volume of gene expression and the complexity of microarray data, various data mining methods and applications have been proposed for analyzing gene expressions. However, it is not trivial to extract biologically meaningful information from the huge amount of gene expression data in understanding of gene regulation networks and cellular state, because most cellular processes are regulated by changes in gene expression. Association rule mining techniques are helpful to find relationship between genes, but most of the developed association rule mining algorithms are based on main memory and single processor based techniques which are not capable of handling ever increasing large data and producing result in a faster way. In this paper, we proposed a MapReduce framework for mining association rules from a huge microarray gene expression dataset on Hadoop; which not only overcomes of the main memory bottleneck but also highly scalable in terms of increasing data size. When we apply this new method to the mice lungs and spinal cord microarray compendium data, it identifies a majority of known regulons as well as novel potential target genes of numerous key transcription factors. Extensive experimental results show that our proposed approach is efficient for mining high confident association rules from large microarray gene expression datasets in terms of time and scalability.

Keywords: *Microarray Data, Gene Expression, MapReduce, Hadoop, Association Rules, Bioinformatics, Parallel Processing*

1. Introduction and Motivations

Bioinformatics is a promising new field which applies computing technology in the molecular biology and develops algorithms to analyze biological data. Since biological data, such as DNA, protein sequence, gene expression dataset exist in huge volumes and reveal biological information as well, it is important to develop effective methods to compare and align DNA or protein sequences and find relationship between gene expression data. Gene is a segment of DNA sequence, which contains the formula for the chemical composition of one particular protein; hence genes serve as the blueprints for proteins and some additional products. Nowadays, the expression levels of thousands of genes, possibly all genes in an organism, can be measured simultaneously in a single experiment using a microarray. This new technology gives rise to a challenge to interpret the meaning of this immense amount of biological information formatted in numerical matrices. A key step in the analysis of gene expression data is to find association and correlation relationship between gene expression patterns [1, 17-19]. Also, using microarray data can reveal the structure of the transcriptional gene regulation processes, which is called reverse engineering [2].

In microarray gene expression data analysis, it is often of interest to identify genes that share similar expression profiles with a particular gene such as a key regulatory protein. Multiple studies have been conducted using various correlation measures to identify co-expressed genes. While working well for small datasets, the heterogeneity introduced from increased sample size inevitably reduces the sensitivity and specificity of these approaches. This is because most co-expression relationships do not extend to all experimental conditions. With the rapid increase in the size of microarray datasets, identifying functionally related genes from large and diverse microarray gene expression datasets is a key challenge [22].

Although successful in analyzing small datasets, the above mentioned correlation or distance measures will be less helpful for searching large datasets, such as microarray compendium data. This is because for most functionally related genes, tight correlation only occurs under specific experimental conditions. Therefore global correlation measures taken across diverse experimental conditions will be significantly reduced, and thus undermine its ability to recognize functional related genes. Given the microarray compendium scenario, we hypothesized that statistically significant correlation can still be detected using Microarray, but strong correlation will be confined to a subset of samples/experimental conditions [22].

Many researchers have focused to construct the gene regulatory network using association rules mining method. Traditional methods [3-5, 17-19] for finding association rules firstly extract frequent itemsets, and then mine high confident association rules from frequent itemsets. Literature [6] was proposed to mine frequent itemsets by using sample enumeration, which explores the enumeration space by constructing projected transposed tables recursively. However, it has some limitations, for example: (i) when the microarray dataset is dense, it is very time consuming. (ii) It keeps the historical frequent closed patterns in memory, which limits the scalability (iii) generated patterns can not reveal the complex gene regulation from microarray data.

Literatures [8] and [9] describe two algorithms called FARMER and Top-k covering rule groups respectively that are specially designed to discover association rules from microarray datasets. Instead of finding individual association rules, FARMER and top-k discovers upper bounds of interesting rule groups by performing depth-first row enumeration using chi-square value. Although experiments on real biological datasets show faster mining time [8] and [9] than previous approaches; however, the FARMER cannot handle activation and inhibition support of genes to generate frequent items [1].

Recently SAW [1], was proposed to generate high confident association rules without using frequent itemsets. This method first generates all paired rules, and then strong association rules are produced by combing them using the forward and backward combined approach. Although this method can avoid some unnecessary computing; however, it produces huge amount of paired rules including invalid rules, in paired rules as well as forward and backward combined step. Besides, this method cannot perform mining operations in a faster way because of column enumeration approach [20]. Therefore, it needs different level wise pruning techniques; hence it shrinks the memory space exponentially.

On the other hand, microarray data gets bigger and reaches larger as the web based applications have grown in the world. Before the web did not exist, we did not have enough publically available microarray data to analyze with the limited volumes of data. However, collecting and analyzing large datasets with the existing file systems and Relational Database Management Systems (RDBMS) are not sufficient to store and handle the data efficiently. Besides, the legacy computing power and platforms were not useful for the big data. There is an increasing interest in approaches to data analysis in scientific computing as essentially every field is seeing an exponential increase in the size of the data deluge. The data sizes imply that parallelism is essential to process the information in a timely fashion [20, 21]. This is generating justified interest in new runtimes and programming models that, unlike traditional parallel models, directly address the data-specific issues where we assume that data is partitioned and transmitted to the computing nodes in advance. Therefore, it is important to efficiently partition and distributes the data to other nodes for parallel computation. In this environment lot of message passing and I/O operation occurs due to MPI. Experience has shown that the initial (and often most time consuming) parts of data analysis are naturally data parallel and the processing can be made independent with perhaps some collective (reduction) operation [20, 21].

From the above discussion, we can conclude that i) main memory and single processor based hardware resources are not capable of handling ever increasing gene expression dataset. Since, the gene expression dataset could not be fit into the main memory. ii) The traditional Apriori or FP like [3-5] algorithms is not suitable. iii) The traditional parallel and distributed data mining algorithms are also not suitable, since, these also poses very impractical to use distributed systems for large datasets mining [8]. This structure has motivated the important MapReduce [11] paradigm and many follow-on extensions. But, MapReduce [11-13] is relatively new and suitable platform to mine these sorts of datasets, since it only needs to share and pass support of an individual candidate itemset rather passing the itemset itself. Therefore, communication cost is low compared to existing approaches. Another advantageous thing is memory requirement is not a major concern in a MapReduce framework and Hadoop/MapReduce based applications also has been developed recently for bioinformatics research [20, 21].

Our contributions in this work can be summarized as follows: (i) we proposed a cloud based MapReduce framework for mining association rules from large microarray gene expression dataset for the first times ever (ii) we developed an algorithm namely 'BMR algorithm' which not only mines our desired association rules but also highly scalable in terms of increasing data load; (iii) we optimized the effectiveness of SAW algorithm [2] using vertical data layout format instead of horizontal layout format. iv) We have showed how to efficiently and vertically partition the microarray dataset and perform our MapReduce based computation on Hadoop.
The rest of this paper is organized as follows: The problem formulation and related knowledge is illustrated in Section 2. Section 3 presents our proposed approach. We devised some experimental results in section 4. Finally we conclude at section 5.

2. Problem Statement and Background Study

2.1. Analysis of Microarray Dataset

The microarray dataset can be seen as a matrix, denoted by real expression numbers shown in Table 1. The columns denote different samples [2] and rows denote genes. In order to mine frequent patterns, microarray data need to be converted each gene expression number into one of the three numbers: 1, -1 and 0, which denotes expressed, depressed and non-expressed, respectively, as shown in table 2. Since existing methods [2-5] of finding association rules from a large data set have been adapted or directly applied to gene expression data, these association rules mining algorithms have been proven useful for identifying biologically relevant association among the genes.

Table 1. An Example of Microarray

Gene	HiFA1	HiFA2	HiFA3	HiFA4	HiFA5	HiFA6
A	2.39	0.27	2.04	0.26	1.4	1.23
B	0.33	0.68	0.46	-0.06	2.16	0.09
C	1.26	1.70	1.58	1.13	1.13	1.13
D	0.77	1.01	0.73	0.63	0.34	0.63
E	0.62	0.99	0.87	0.42	1.42	0.35
F	0.33	0.25	0.10	2.15	0.15	1.17

Table 2. Converted Microarray Dataset

Gene	HiFA1	HiFA2	HiFA3	HiFA4	HiFA5	HiFA6
A	1	0	1	0	1	1
B	0	1	0	0	1	0
C	1	1	1	1	1	1
D	1	1	1	1	0	1
E	1	1	1	0	1	0
F	0	0	0	1	0	1

2.2 Cloud Technologies and the Services for Analyzing Big Dataset

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility over a network. Cloud computing entrusts, typically centralized, services with your data, software, and computation on a published application programming interface (API) over a network. It has a lot of overlap with software as a service (SaaS) [23]. End users access cloud based applications through a web browser or a light weight desktop or mobile app while the business software and data are stored on servers at a remote location. Cloud application providers strive to give the same or better service and performance than if the software programs were installed locally on end-user computers. At the foundation of cloud computing is the broader concept of infrastructure convergence (or Converged Infrastructure) and shared services. This type of data centre environment allows enterprises to get their applications up and running faster, with easier manageability and less maintenance, and enables IT to more rapidly adjust IT resources (such as servers, storage, and networking) to meet fluctuating and unpredictable business demand [23].

2.3 Apache Hadoop, HDFS and Relevance of MapReduce to Many-Task Computing

Thus, Google implemented Google File Systems (GFS), BigTable, and MapReduce parallel computing platform, which Apache Hadoop and HBase projects are motivated from. Hadoop is the parallel programming platform built on Hadoop Distributed File Systems (HDFS) for MapReduce computation that processes data as <key, value> pairs. HBase runs on HDFS with Hadoop MapReduce to store and process big data. HBase and Hadoop have been adopted dramatically for the enterprise computing because business world always has the big data such as log files for web transactions, which is not easy to store and compute. Especially, when Amazon AWS supports Hadoop instances, it becomes much easy for people to run [11-13]. MapReduce was developed within Google as a mechanism for processing large amounts of raw data. This data is so large, so it must be distributed across thousands of machines in order to be processed in a reasonable time. This distribution implies parallel computing since the same computations are performed on each CPU, but with a different dataset. The user of the MapReduce library expresses the computation as two functions: Map and Reduce [11-13]. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per reduce invocation. The intermediate values are supplied to the user's reduce function via iterator. This allows us to handle lists of values that are too large to fit in the main memory. On the other hand, Hadoop is a popular open source implementation of MapReduce, which is a powerful tool designed for deep analysis and transformation of very large data sets inspired by MapReduce and Google File System [11-13]. It enables applications to work with thousands of nodes and petabytes of data. Hadoop uses a distributed file system called Hadoop Distributed File System [11-13], which creates multiple replicas of data blocks and distributes them on computing nodes to enable reliability and has extremely rapid computations to store data and intermediate results [11-13]. In a Hadoop cluster, a master node controls a group of worker nodes on which the Map and Reduce functions run in parallel. Apache Hadoop has a similar architecture to Google's MapReduce [11-13] runtime. Hadoop accesses data via HDFS [24], which maps all the local disks of the compute nodes to a single-file system hierarchy, allowing the data to be dispersed across all the data/computing nodes. HDFS also replicates the data on multiple nodes so that failures of nodes containing a portion of the data will not affect the computations which use that data. Hadoop schedules the MapReduce computation tasks depending on the data locality, improving the overall I/O bandwidth. The outputs of the map tasks are stored in local disks until the reduce tasks access them (pull) via HTTP connections. Although this approach simplifies the fault handling mechanism in Hadoop; it adds a significant communication overhead to the intermediate data transfers, especially for applications that produce small intermediate results frequently. Apache Hadoop runs only on Linux operating systems [24].

MapReduce programming model comprises of two computation steps (map/reduce) and an intermediate data shuffling step, which can be used to implement many parallel applications [11-13]. The three steps collectively provide functionality beyond the requirements of simple many-task computations (MTC). However, this does not hinder the usability of MapReduce to MTC applications. For example, when only a map operation is used, the MapReduce programming model reduces to a "map-only" version that is an ideal match for MTC applications. Furthermore, with the capability of adding a "reduction" operation MapReduce can be used to collect or merge results of the embarrassingly parallel phase of some of the MTC applications as well. Our research on applying MapReduce to computations was motivated by these observations [11-13].

2.3 Mining Association Rules

Let the data set $D_M = \{t_1, t_2, t_n\}$ be a set of n transactions from microarray dataset after transactionization of the microarray dataset in table 1 and shown at table 3. Also suppose $I = \{i_1, i_2, i_m\}$ be the set of items. The support of an item X is the number of transactions in D_M where item X occurs. Item X is a frequent item if and only if the support of *X* is greater than the minimum support threshold. An association rule is described as the form LHS=>RHS denoted as R; where LHS and RHS are disjoint itemset. LHS is referred to as the antecedent itemset and RHS as the consequent itemset. The support of R is the number of transactions in D that contain both LHS and RHS. The confidence of R is the ratio supports (LHS U RHS)/support (LHS). A rule which satisfies the minimum support and minimum confidence thresholds is said to be a strong association rule.

3. Proposed MapReduce Framework

In this section, we present an efficient algorithm BMR (Bio-MapReduce for association rules mining), to generate high confident association rules. First, we describe the input and output schemes for the proposed framework. After that we describe the BMR algorithm. Finally we describe a step by step example to demonstrate our proposed framework.

Table 3. Key/Value Pairs for the Proposed MapReduce Framework

I/O	Map-1	Map-2	Reduce-1	Reduce-2
Input: key/value pairs	Key: *itemset* Value: *GID*	Key: *CI* Value: *support*	Key: *FI* Value: *support*	Key: *AR*, Value: *confidence*
Output: key/value pairs	Key: *CI* Value: *support*	Key: *FI* Value: s*upport*	Key: *SAR* Value: *confidence*	Key: *CSAR* Value: *confidence*

Legend: CI is candidate itemset, *FI* is frequent itemset, S*AR* is strong association rules and CSAR is for closed strong association rules.

3.1 Proposed Programming Model

Since microarray dataset is very huge with a large number of columns and small number of rows, it is very time consuming process to use horizontal format of the dataset, hence we used the concept of vertical format on modified parallel balanced FP-growth [13], as like transactional dataset. Before dealing with microarray datasets with MapReduce on Hadoop platform we need to convert the gene expression microarray dataset into converted microarray as shown by table 2 and after that we transactionize the converted datasets into two tuples as gene sample id or GID (i.e. HiFA1, HiFA2,…) and itemset shown by table 3; based on the following assumption; we can consider table 2 as a transactional dataset, where A denotes the expression number of A is 1, while –A means the value is -1 which cannot be seen in this example and 0 means inactive. Then transactionized dataset in disk files are splited into smaller segments automatically after stored on HDFS [13-15]. Therefore, after splitting the transactionized dataset into smaller segments the master node assign task to idle worker nodes. Table 3 has shown the input/output schemes for our proposed MapReduce framework. After that worker nodes scan the transactions in the smaller segments as < *itemset,* G*ID*> pairs and generate output as <candi_itemset, support> pairs. These values are inputted to reduce phase after sorting and merging operations.

We use two level pruning- local pruning and the global pruning, using two thresholds ‘global_min_sup’ and ‘min_conf’. Global pruning is applied on map phase in each segment and local pruning is applied in reduce phase. For this purpose we modified the balanced FP-growth [13] using MapReduce library function using in Java. In reduce phase--1 a worker node takes input as <freq_itemset, support> pairs then checks the value of minimum confidence for each frequent itemset generated and generate output as <SAR, conf> pairs. In reduce phase-2 idle worker nodes find the closed strong association from the strong association rules generated in reduce phase 1. Finally the output is stored on the output files as <CSAR, conf> pairs. Figure 1 shows the workflow and Figure 2 shows the BMR algorithm.

Table 4. Transactional Dataset for Table 2 in Vertical Layout Format

GID (Gene Sample ID)	Items
HiFA1	A, C, D, E
HiFA2	B, C, D, E
HiFA3	A, C, D, E
HiFA4	C, D, F
HiFA5	A, B, C, E
HiFA6	A, C, D,F

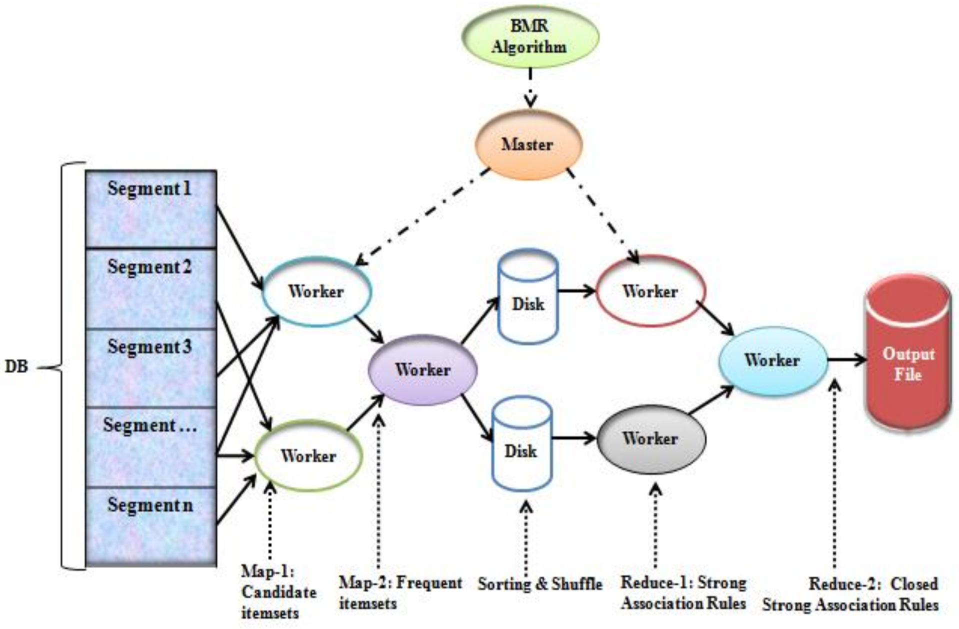


Figure 1. Workflow of the proposed MapReduce framework for mining association rules in microarray gene exppression dataset

3.2 AN EXAMPLE

Suppose the global_min_sup is 3 and min_conf is 75%. Now according to our proposed framework the converted microarray dataset presented in table 3 has been splited into two segments on each has three transactions. GIDs 1, 2, 3 are in first segment and transaction GIDs 4, 5, 6 are in second segments. Let the master node assigns segment 1 to worker node 1and segment 2 to worker node 2. Mapper maps the <itemset, GID> pairs and generate output as <candi_itemset, support> pairs in map phase 1 and <freq_itemset, support> pairs in map phase 2 respectively as intermediate values and inform the master node. Table 5 and 6 shows the result of map phase 1 and 7 shows the result of map phase 2. In reduce phase worker nodes take input as <freq_itemset, support> pairs then first generate only strong association rules then generate closed strong association rules presented by table 8 and 9 as <SAR, conf> and <CSAR, conf> pairs respectively.

Table 5. Candidate Itemset: Worker 1

C. itemset	Support	C. itemset	Support
A	2	CE	3
C	3	DE	3
D	3	ACD	3
E	3	ACE	2
AC	2	ADE	2
AD	2	CDE	3
AE	2	ACDE	2
CD	3		

Table 6. Candidate Itemset: Worker 2

C. itemset	Support	C.itemset	Support
A	2	AE	1
C	3	CD	2
D	2	CE	1
E	1	ACD	1
AC	2	ACE	1
AD	1		

Table 7. Frequent Itemset: Worker 3

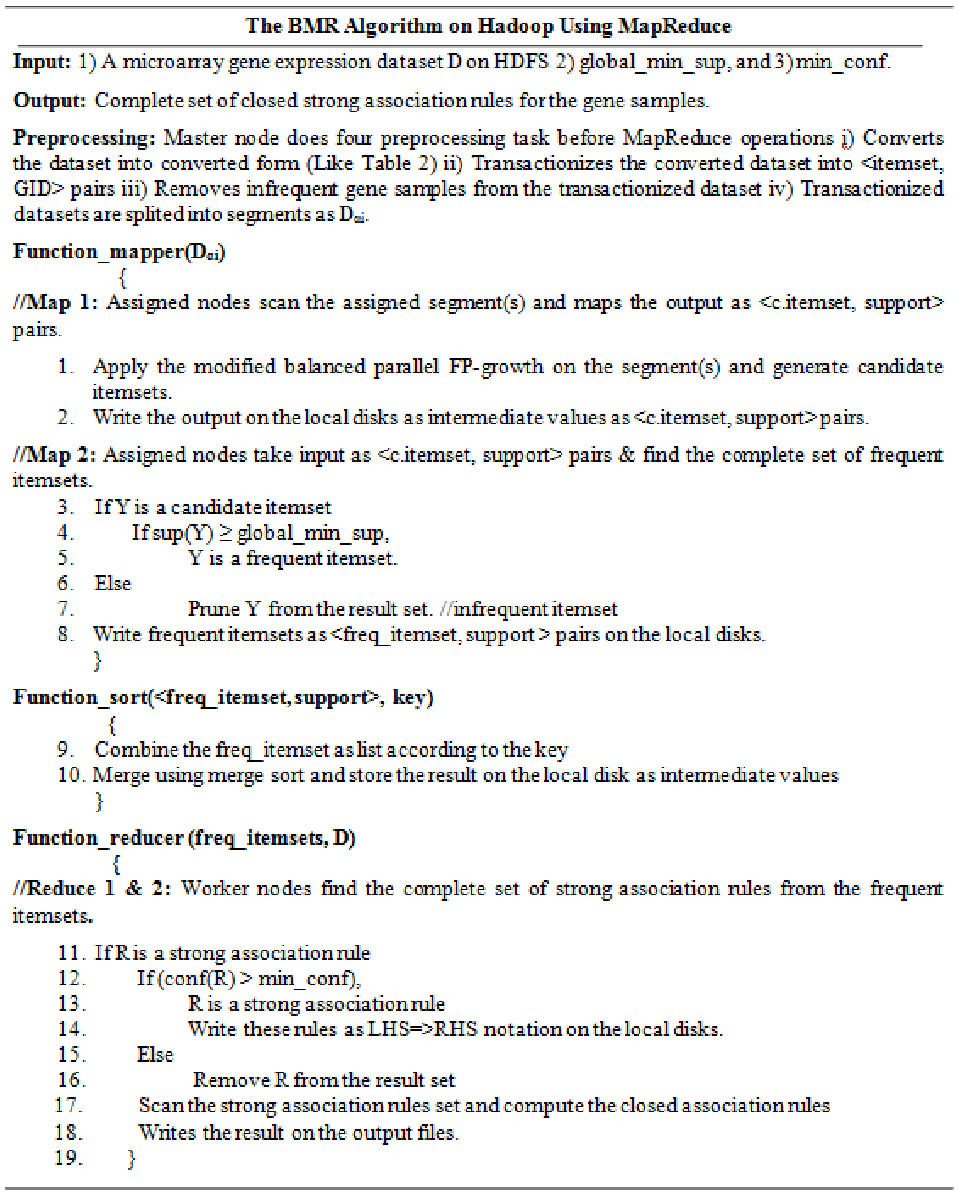
Frequent itemset	Support	Frequent itemset	Support
A	4	AC	4
C	6	CD	5
D	5	CE	4
E	4	CDE	3
DE	3	ACD	3
AE	3	ACE	3
AD	3		

Table 8. Strong Association Rules: Worker 4

Rules	Confidence	Rules	Confidence	Rules	Confidence
A=>C	100%	CE=>D	75%	AC=>D	75%
A=>D	75%	CE=>A	75%	A=>CE	75%
A=>E	75%	D=>C	100%	C=>D	84%
AD=>C	100%	DE=>C	100%	E=>AC	75%
AE=>C	100%	E=>C	100%	E=>CD	75%
AC=>E	75%	E=>A	75%	A=>CD	75%

Table 9. Closed Strong Association Rules: Worker 5

Closed Rules	Confidence	Closed Rules	Confidence
A=>CD	75%	AE=>C	100%
A=>CE	75%	AC=>E	75%
AD=>C	100%	AC=>D	75%
CE=>D	75%	DE=>C	100%
CE=>A	75%	E=>AC	75%
E=>CD	75%		



The BMR Algorithm on Hadoop Using MapReduce

Input: 1) A microarray gene expression dataset D on HDFS 2) global_min_sup, and 3) min_conf.

Output: Complete set of closed strong association rules for the gene samples.

Preprocessing: Master node does four preprocessing task before MapReduce operations i) Converts the dataset into converted form (Like Table 2) ii) Transactionizes the converted dataset into <itemset, GID> pairs iii) Removes infrequent gene samples from the transactionized dataset iv) Transactionized datasets are splited into segments as D_{ai}.

```
Function_mapper(D_ai)
        {
//Map 1: Assigned nodes scan the assigned segment(s) and maps the output as <c.itemset, support> pairs.
   1.  Apply the modified balanced parallel FP-growth on the segment(s) and generate candidate itemsets.
   2.  Write the output on the local disks as intermediate values as <c.itemset, support> pairs.
//Map 2: Assigned nodes take input as <c.itemset, support> pairs & find the complete set of frequent itemsets.
   3.  If Y is a candidate itemset
   4.      If sup(Y) ≥ global_min_sup,
   5.          Y is a frequent itemset.
   6.  Else
   7.          Prune Y from the result set. //infrequent itemset
   8.  Write frequent itemsets as <freq_itemset, support> pairs on the local disks.
       }
Function_sort(<freq_itemset, support>, key)
       {
   9.  Combine the freq_itemset as list according to the key
   10. Merge using merge sort and store the result on the local disk as intermediate values
       }
Function_reducer (freq_itemsets, D)
        {
//Reduce 1 & 2: Worker nodes find the complete set of strong association rules from the frequent itemsets.
   11. If R is a strong association rule
   12.     If (conf(R) > min_conf),
   13.         R is a strong association rule
   14.         Write these rules as LHS=>RHS notation on the local disks.
   15.     Else
   16.         Remove R from the result set
   17.     Scan the strong association rules set and compute the closed association rules
   18.     Writes the result on the output files.
   19.   }
```

Figure 2. The BMR Algorithm using MapReduce

4. Experimental Results

4.1 Hardware Configurations

We used Hadoop version 0.20.0, running on a cluster with 6 machines (1 master, 5 worker). Master node has 3.7 GHz Intel core-2 duo with 4GB of RAM and each worker machine has Pentium-D 2.60GHz processor with 2GB RAM. The balanced FP-growth [16] was modified in Java using MapReduce library functions and configured HDFS on Ubuntu-11.04.

4.2 Description of the Datasets

We used the gene expression array of lungs and spinal cord dataset, catalogs of gene expression of mice [16]. The expression changes for 8,932 and 11514 genes respectively. The mice were of ages 1, 6, 16, and 24 month. Fig.3 shows the compared mining time of SAW [1], FARMER [8] and BMR algorithm at different thresholds. Datasets were splited across and run on small instances of Hadoop which allows to instantiate for 3 and 5 nodes requested for the first and second experiment respectively. We follow the load distribution for the nodes according to [16]. Table 10 shows the characteristics of these datasets.

Table 10. Characteristics of the Datasets

Gene expression dataset	# Of gene expression	Age (months)**	Size of the dataset
Mice age lungs catalog	8, 932	1, 6, 16 and 32	100MB
Mice age spinal cord catalog	11,514	1, 6, 16 and 32	450MB

** The mice were of ages 1, 6, 16, and 24 month.

4.3 Performance Analysis

In the first experiment we observed that BMR outperforms both FARMER and SAW on each threshold shown in Figure 3(a). In the second experiment speed up process has been shown by increasing computing worker nodes. Figure 3(b) shows the running time on mice lungs datasets across 5 data nodes. We can observe almost 50% improvement on mining time. Table 11 and 12 shows execution time per node for the Mapper and reducer nodes respectively for min_sup=0.1 and min_conf=0.2 for lungs and spinal cord datasets in second. It is noted that we only showed execution time on the time of copying and sorting.

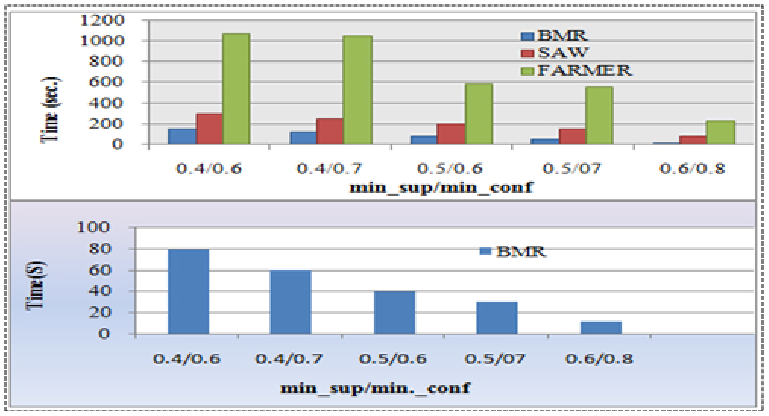


Figure 3. Upper: time with change of min_sup & min_conf on mice lungs dataset with 3 nodes; Lower: time with 5 worker nodes (speed up)

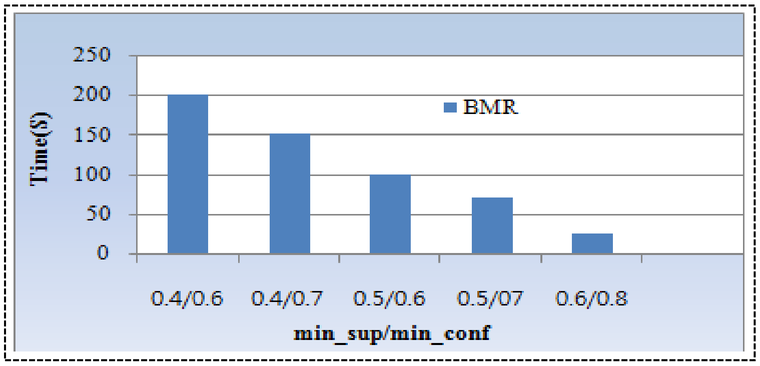


Figure 4. Scalability of the framework with change of min_sup/min_conf on mice spinal cord

Figure 4 shows the scalability of the framework on spinal cord dataset with 5 working nodes; it implies that more the nodes are, the faster the computation times are. Since the algorithm is simply to sort the data set and then convert it to <key, value> pairs, the linear result is expected.

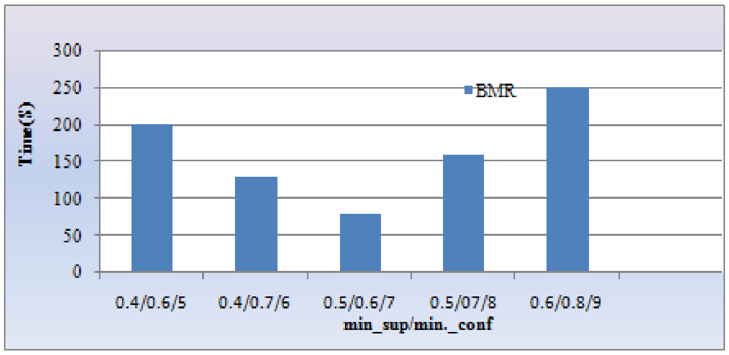


Figure 5. Performance Bottleneck of the Proposed Framework in Hadoop Small Instance with 8 Working

The performance is linearly increased by adding more nodes for microarray data sets but it has limitation. Figure 5 shows that there is a bottleneck in Hadoop small instance, which shows that there is a trade-off between the number of nodes and the operations of distributing transactions data to nodes, aggregating the data, and reducing the output data for each key so that it should not have much performance gain even though adding mode nodes for faster parallel computation.

Table 11. Execution time per node for the Mapper; min_sup=0.1 and min_conf=0.2

# Nodes	Execution time (sec.) (Lungs - 110MB)	Execution time (sec.) (Spinal cord - 450MB)
1	1,535	3,312
2	1,034	2,235
3	737	1,123
4	455	825
5	152	539

Table 12. Execution time per node for Reducer nodes for min_sup=0.1 & min_conf=0.2

# Nodes	Execution time (sec.) (Lungs - 110MB)	Execution time (sec.) (Spinal cord - 450MB)
1	853	1,582
2	663	968
3	487	758
4	223	485
5	92	259

The output of the proposed framework on Mice lungs and spinal cord as the number of closed strong association rules has been shown in table 13 for 5 working nodes.

Table 13. Number of Strong and Closed Strong Association Rules

Parameters: (min_sup/min_conf)	# Closed Strong Association Rules in Datset#1 (Mice lungs)	# Closed Strong Association Rules in Dataset#2 (Mice spinal cord)
0.4/0.6	2,285	4,786
0.4/0.7	1,985	3,239
0.5/0.6	11,46	2,223
0.5/0.7	833	1,665
0.6/0.8	322	1,127

5. Conclusions

Hadoop with MapReduce motivates the needs to propose new algorithms for the existing applications that have some efficient algorithms for sequential computation. Besides, it is <key, value> based restricted parallel and distributed computing so that the legacy parallel algorithms need to be redesigned with MapReduce framework using some programming language Java, Perl or PHP. In this paper, we proposed a MapReduce framework for mining association rules from huge gene expression microarray dataset on Hadoop. It overcomes the single processor and main memory based rule mining algorithms and highly scalable in terms of increasing load. Two experimental results show that our proposed BMR algorithm outperforms the latest approaches for mining high confident association rules in terms of time and scalability. The gene expression dataset shows that associated genes can be paired with MapReduce approach. Once we have the paired genes, it can be used for more studies by statically analyzing them even sequentially, which is beyond this paper. The algorithm has been executed on Hadoop small instances with 3 and 5 data nodes. The execution times of the experiments show that the proposed algorithm gets better performance while running on large number of nodes to a certain point. However, from a certain point, MapReduce does not guarantee to increase the performance even though we add more nodes because there is a bottle-neck for distributing, aggregating, and reducing the data set among nodes against computing powers of additional nodes.

Acknowledgement

This work was supported by the National Research Foundation (NRF) grant (No. 2011-0018264) of Ministry of Education, Science and Technology (MEST) of Korea.

References

[1] Anandhavalli GM, “Analysis of DNA Microarray Data using Association Rules: A Selective Study”, World Academy conference of Science, Engineering and Technology, **(2008)**.

[2] Miao W, Xuequn S and Zhanhuai L, “Research on Microarray Dataset Mining”, VLDB PhD Workshop, **(2010)** September, Singapore.

[3] Agrawal R and Srikant R, “Fast algorithms for mining association rules in large databases”, In proc. of 20th Int’l conf. on VLDB, **(1994)**, pp. 487-499.

[4] Han J, Pei J and Yin Y, “Mining Frequent Patterns without candidate generation”, In proceedings of the ACM-SIGMOD, **(2000)**.

[5] Gyorodi C and Gyorodi R, “Mining Association rules in Large Databases”, in proceedings of the EMES conference, Romania, **(2002)**, pp. 45-50.

[6] Pan F, Cong, Tung K and Zaki M, “Carpenter: Finding closed patterns in long biological datasets”, In proc. of the Intl. Conf. on KDD, **(2004)**.

[7] Pei J and Han J, “Mining Sequential Patterns by Pattern-growth: The PrefixSpan Approach”, IEEE Transactions on Knowledge and Data Engineering, vol. 6, no. 10, **(2004)**, pp. 1-17.

[8] Gao C, Anthony KHT, Xin X, Feng P and Jiong Y, “FARMER: Finding Interesting Rule Groups in Microarray Datasets”, IGMOD, **(2004)** June 13-18, Paris, France.

[9] Gao C, Kian-Lee T, Anthony KHT and Xin X, “Mining Top-k Covering Rule Groups for Gene Expression Data: SIGMOD”, **(2005)** June 14-16, Baltimore, Maryland, USA.

[10] Chad C and Samir H, “Mining gene expression databases for association rules”, Journal of Bioinformatics, vol. 19, no. 1, **(2003)**, pp. 79–86.

[11] Dean J and Ghemawat, “MapReduce: Simplified data processing on large clusters”, ACM, **(2008)**.

[12] Panda B, Basu S and Bayardo RJ, “Massively Parallel Learning of Tree with MapReduce”, ACM. VLDB ‘09, **(2009)** August 24-28, Lyon, France.

[13] Zhou L, ong Z, Chang J, Li J, ng JZ and FengS, “Balanced Parallel FP-Growth with MapReduce”, IEEE Youth Conf. on Information Computing and Telecom (YC-ICT), **(2010)**.

[14] Ekanayake J, Pallickara S and Fox G, “MapReduce for data intensive scientific analyses”.

[15] Xie J, Manzanares A and Qin X, “Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters”, IEEE Intl. Symp. On PDPW & PhD Forum, **(2010)**.

[16] Mouse aging dataset, http://cmgm.stanford.edu/~kimlab/aging_mouse/.

[17] Tajnisha N and Saravanan V, “A new approach to improve the clustering accuracy using informative genes for unsupervised microarray data sets”, IJAST, vol. 27, **(2011)** February, pp. 85-94.

[18] Huynh HT, Kim J and Won Y, “Classification Study on DNA Microarray with feed forward Neural Network Trained by Singular Value Decomposition”, IJBSBT, vol. 1, no. 1, (2009) December.

[19] Han Y, “Bioworks: A Workflow System for Automation of Bioinformatics Analysis Processes”, IJBSBT, vol. 1, no. 1, **(2009)** December, pp. 59-68.

[20] Ekanayake J, Gunarathne T and Qiu J, “Cloud Technologies for Bioinformatics Applications”, IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 6, June.

[21] Meng Z, Li J, Zhou Y, Liu Q, Liu Y and Cao W, “bCloudBLAST: an Efficient MapReduce Program for Bioinformatics Applications”, 4th Intl. Conf. on Biomedical Engineering and Informatics (BMEI), **(2011)**.

[22] Hu M and Qin ZS, “Query Large Scale Microarray Compendium Datasets Using a Model-Based Bayesian Approach with Variable Selection”, Plosone open J. Liverpool Univ., **(2009)** February 13.

[23] Cloud Computing, http://en.wikipedia.org/wiki/Cloud_computing.

[24] Apache Hadoop, http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html.

Authors

Md. Rezaul Karim: Received the BS degree from the Dept. of Computer Science & Engineering, University of Dhaka, Bangladesh in 2009. Currently he is a MS degree candidate at the Dept. of Computer Engineering, Kyung Hee University, Korea. His research interest includes data mining, ubiquitous data management and bioinformatics.

E-mail: asif_karim@khu.ac.kr

A.T.M. Golam Bari: Received the BS degree from the Dept. of Computer Science & Engineering, University of Dhaka, Bangladesh in 2007. Currently he is a MS degree candidate at the Dept. of Computer Engineering, Kyung Hee University, Korea. His research interest includes knowledge discovery & data mining and social networking.

E-mail: bari@khu.ac.kr

Byeong-Soo Jeong: Received the BS degree in Computer Engineering from Seoul National University, Korea, in 1983. MS degree in Computer Science from the Korea Advanced Institute of Science and Technology in 1985, and the PhD in Computer Science from the Georgia Institute of Technology, Atlanta, in 1995. From 1996 he is a professor at the Department of Computer Engineering, Kyung Hee University, Korea. From 1985 to 1989, he was on the research staff at Data Communications Corporation, Korea. From 2003 to 2004, he was a visiting scholar at the Georgia Institute of Technology, Atlanta. His research interests include database systems, data mining, and mobile computing.

E-mail: jeong@khu.ac.kr

Ho-Jin Choi: Received the BS degree in Computer Engineering from Seoul National University, Korea, in 1985. MS in Computing Software and Systems Design from Newcastle University, UK and the PhD in Artificial Intelligence from Imperial College, London in 1995. Between 1995 and 1996, he was a post-doctoral researcher at IC-PARC, Imperial College, and London. From 1997 to 2002, he worked as an assistant professor of Computer Engineering at Korea Aerospace University. Currently he is an associate professor at the Dept. of Computer Science at KAIST, Korea. His research interests include artificial intelligence, data mining, software engineering, and biomedical informatics.

E-mail: hojinc@kaist.ac.kr