IJCCS (Indonesian Journal of Computing and Cybernetics Systems)
Vol.12, No.2, July 2018, pp. 149~160
ISSN (print): 1978-1520, ISSN (online): 2460-7258
DOI: 10.22146/ijccs.34102

The MapReduce Model on Cascading Platform for Frequent Itemset Mining

Nur Rokhman *[1], Amelia Nursanti [2]
[1]Department of Computer Science and Electronics, FMIPA UGM, Yogyakarta, Indonesia
[2]Computer Science Study Program, FMIPA UGM, Yogyakarta, Indonesia
e-mail: ***[1]nurrokhman@ugm.ac.id**, [2]amelia.nursanti@mail.ugm.ac.id

Abstrak

Implementasi algoritma paralel telah menjadi penelitian yang menarik dewasa ini. Paralelisme sangat cocok untuk menangani pemrosesan data berukuran besar. Saat ini tersedia beberapa model pemrograman paralel dan terdistribusi seperti Mapreduce, MPI dan CUDA. Implementasi algoritma paralel menghadapi beberapa kendala ketika ukuran data dan kompleksitas bertambah. Cascading menyediakan skema yang mudah bagi sistem Hadoop yang menerapkan model MapReduce untuk melakukan refactor, testing, eksekusi aplikasi kompleks, dan konversi aplikasi yang telah dibangun ke sistem Hadoop.

Frequent Itemset adalah obyek-obyek yang sering muncul dalam himpunan data. Frequent Itemset Mining (FIM) memerlukan komputasi yang kompleks. FIM merupakan masalah kompleks bila diterapkan pada data berukuran besar.

Makalah ini mendiskusikan penerapan model MapReduce pada Cascading untuk keperluan FIM. Eksperimen dilaksanakan dengan menggunakan himpunan data pembelian produk Amazon. Eksperimen menunjukkan fakta bahwa mekanisme sederhana pada Cascading seperti yang identik dengan merangkai sistem pipa dapat digunakan untuk menyelesaikan masalah FIM. Hal ini menghasilkan kompleksitas waktu O(n), lebih efisien dari proses non paralel yang memiliki kompleksitas $O(n^2/m)$.

Kata kunci— *Frequent Itemset Mining, MapReduce, Cascading*

Abstract

The implementation of parallel algorithms is very interesting research recently. Parallelism is very suitable to handle large-scale data processing. There are parallel and distributed programming models, such as MapReduce, MPI, and CUDA. The implementation of parallel programming faces difficulties when the data size and complexity increase. The Cascading gives easy scheme of Hadoop system which implements MapReduce model to refactor, test, execute a complex application and converting an application into Hadoop system.

Frequent itemsets are objects which most often appear in a dataset. The Frequent Itemset Mining (FIM) requires complex computation. Therefore, FIM is a complicated problem when implemented on large-scale data.

This paper discusses the implementation of MapReduce model on Cascading for FIM. The experiment uses the Amazon dataset product co-purchasing network metadata. The experiment shows the fact that the simple mechanism of Cascading which like assembling a pipe system can be used to solve FIM problem. It gives time complexity O(n), more efficient than the nonparallel which has complexity $O(n^2/m)$.

Keywords— *Frequent Itemset Mining, MapReduce, Cascading*

Received March 20th,2018; Revised July 7th, 2018; Accepted July 30th, 2018

1. INTRODUCTION

1.1 Previous works

The fast-growing of computer technology causes a tremendous data increasing. Frequent itemsets are objects that often appear on a dataset. Objects are said to be frequent if their appearance greater than a specified support value. By finding the frequent itemsets in a system, the patterns of the system can be recognized. Frequent itemsets can mine the relevant evidence of computer crime, mine crime trends, and mine connections among different crimes. It can help polices detect case and prevent crime with clues and criterions [1]. Frequent itemset mining also plays an important part in college library data analysis. RFP-Growth algorithm was used to find the frequent itemset college library database. There are a lot of redundant data in a library database. The mining process may generate intra-property frequent itemsets [2].

Frequent Itemsets Mining (FIM) is a process of finding the frequent itemsets by using data mining. FIM is a very interesting problem. Some research focus on the algorithm such as MRApriori algorithm [3], parallel balanced mining algorithm for Closed Frequent Itemsets based on the MapReduce [4], Hadoop-MapReduce model for handling massive datasets in mining infrequent itemsets [5], Sequence-Growth algorithm on MapReduce framework [6], data partitioning strategy on Hadoop [7], and the mining algorithm of frequent itemsets based on MapReduce and FP-tree (MAFIM algorithm) [8]. Some other research focus on the algorithm implementation for specific objects.

A substantial frequent itemset mining algorithms and their MapReduce implementations are introduced and investigated [9]. The use of Hadoop MapReduce framework makes the execution time linear to the number of transactions per batch. It was found that the increasing stock size did not give much impact on execution time. Execution time is also inversely proportional to the number of nodes [10]. The MapReduce framework can be used for mining frequent itemsets to infer greater scalability and speed in order to find out the meaningful information from large datasets [11].

A deep review of different FIM techniques shows that the current distributed FIM algorithms often suffer from generating huge intermediate data or scanning the whole transaction database for identifying the frequent itemsets[12]. The MapReduce framework is used to build a collaborative filtering. It makes automatic predictions (filtering) about the interests of a user by collecting the preferences or taste information from many users (collaborating) [13]. Three MapReduce tasks are implemented to complete the mining of big datasets by using the parallelism among computing nodes of clusters to improve the performance of frequent pattern mining on Hadoop clusters [14].

MapReduce is a programming model for distributed and parallel computing which is very suitable for large-scale data processing. MapReduce was originally developed by Google for parallel and distributed processing [15]. MapReduce was developed to work on thousands of machines and massive datasets [16].

The implementation of three Aeste-based a priori algorithm based on Hadoop MapReduce namely MRApriori, one-phase, and *k*-phases have been compared [3]. The MRApriori algorithm took only two phases of MapReduce jobs to search for all Frequent k-Itemsets. Experimental results show that the MRApriori algorithm outperforms comparing the other two algorithms.

MapReduce-based balanced mining algorithm for closed frequent itemset has been presented [4]. The algorithm adopts the Greedy strategy to balance the parallel computing. The algorithm consists of three steps: parallel computation, global construction of the frequent list and group maps as well as parallel mining for closed frequent itemset. The experiment showed the effectiveness and scalability the close FIM on a large scale data.

The MapReduce Apriori algorithm on FIM was used to speed up the response time [9]. It

found a solution for porting the Count Distribution algorithm to MapReduce.

Parallel Improved Single Pass Ordered (PISPO) based on cloud-computing framework and MapReduce has been proposed [4]. The algorithm improved SPOTree, FP-Growth and MapReduce algorithms. PISPO was used to find the frequent itemset in electronic evidence.

There are many other application which use FIM on Hadoop MapReduce. Among of this generates the association rules in the transactional data stream [10] and handles FIM in Social Network Data [11].

MapReduce is a complex and difficult framework to be implemented even for software engineers. The Cascading platform may be used to simplify the process of writing program code. The Cascading libraries abstract the complex data flow on MapReduce programming model [17].

This paper explores the use of Cascading platform on simplifying the MapReduce programming code for FIM problem. Then, the program is used to find the frequent itemset of Amazon transaction data. The time needed to solve the problem is observed. The time needed by the parallel program which implemented on Cascading platform and the non-parallel program are compared. Also, the effect of data size and support number to the execution time are observed.

1.2 Related Works

1.2.1 MapReduce

MapReduce is a programming model for processing large scale data. MapReduce model has two main processes namely Map process and Reduce process. Figure 1 shows the relation between Map and Reduce processes. The MapReduce process is begun by breaking up the input data into multiple data items. The Map function outputs one or more key-value pairs. The key-value pairs then sorted and grouped based on the key value. For each distinct key, Reduce function processes and outputs one or more key values to a file as the final result [18].

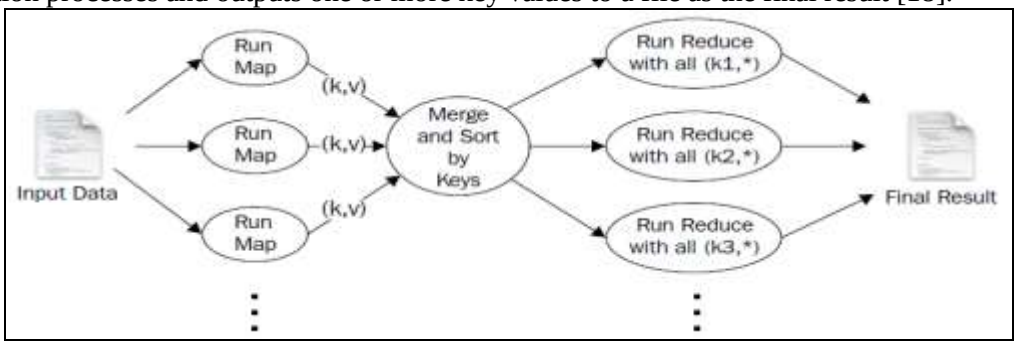

Figure 1. Map and Reduce function [18].

1.2.2 Hadoop

Hadoop is the most popular implementation of MapReduce model. Hadoop is a software framework for reliable, scalable, parallel and distributed computing [16]. The Hadoop framework consists of libraries and utilities required for other Hadoop modules, Hadoop Distributed File System (HDFS), and Hadoop Yet Another Resource Negotiator (YARN). HDFS is a distributed system that provides high access via data applications. YARN is a framework for job scheduling and cluster resource management. YARN provides APIs for resource management. YARN also serves another application framework such as Spark and Tez. Hadoop MapReduce is a YARN-based system for large-scale parallel data processing. Figure 2 shows the Hadoop MapReduce model as a YARN-based system.

1.2.3 Cascading

Cascading is an application development platform for building big data applications on Hadoop. Cascading has Java Application Programming Interface (API) which is used to

simplify the complexity of MapReduce-based programming that run on the Hadoop. Cascading creates and executes complex data workflow processing on Hadoop. Cascading consists of API for data processing, integration, process design and process scheduling. Cascading can be used directly as Hadoop has been installed [17]. Figure 2 shows the Hadoop MapReduce model.

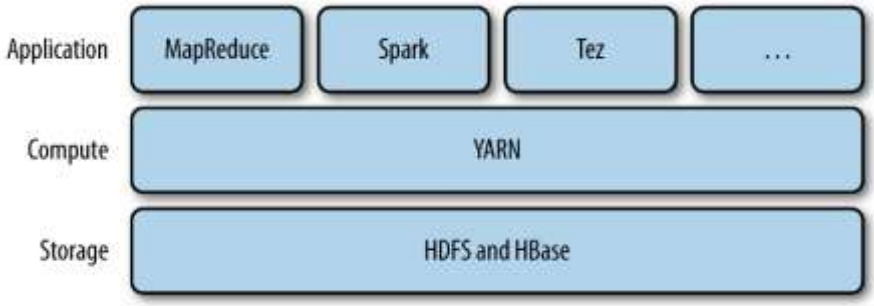

Figure 2. The Hadoop MapReduce model as a YARN based system [19]

Cascading does not change the layer of mapper-reducer and sub-system layers structure in Hadoop. Cascading provides an abstraction for the MapReduce programming model. The workflow used in Cascading is called "Source-Pipe-Sink". Figure 3 shows the workflow of the Cascading.

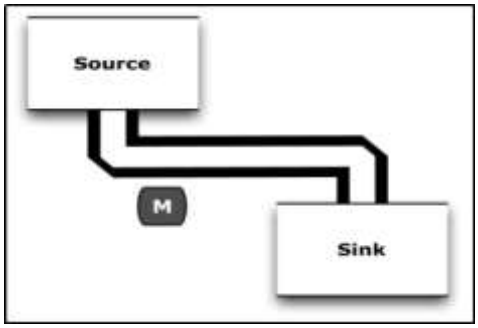

Figure 3. The work flow of the Cascading [20]

In the Cascading model, data is saved in the input part called "s*ource*". Then, data is sent to the output part called "*sink*", through the path called "*pipe*". Additional processes may be executed while the data flows from the "*source*" to the "*sink*".

A Cascading application may have many "*flow"*. Every "*flow*" represent physical plan which analog to the scheduling topology on Hadoop. Every "*pipe*" has head and tail. A "*flow*" works independently and parallel to the other "*flow"*. Cascading uses tuple-centric data model. All data is represented as tuples. Tuples are a list of values. Tuples flow in the "*pipe*".

Cascading has **`pipe`** types which defined as operations in the stream. Among of this operation are **`Each`**, **`Merge`**, **`GroupBy`**, **`Every`**, **`CoGroup`** and **`HashJoin`** pipes. The **`Each`** operation is an operation for the individual tuple. It contains filter, replace value, and remove tuple operations. The **`Merge`** operation merges two or more streams. The **`GroupBy`** operation groups the tuple based on the field and its value.

The grouping operation prepares the stream to be processed by using aggregator operation and buffer in the group such as counting, totaling, or averaging. The **`Every`** operation works on the grouped stream tuple, the output of **`GroupBy`** or **`CoGroup`** operation. The **`CoGroup`** and **`HashJoin`** are grouping operation which group two or more streams to get the specific field of output stream.

1.2.4 Frequent Itemset

Frequent itemsets are objects that often appear on a dataset. Objects are said to be frequent if their appearance greater than the specified support value [3]. Table 1 shows examples

of transaction data.

Table 1. Examples of transaction data

ID	***Item***
1	*Processor, motherboard, memory*
2	*processor, motherboard, memory*
3	*Processor*
4	*processor, motherboard*
5	*Motherboard*
6	*processor, motherboard.*
7	*Processor, memory*
8	*motherboard, memory*
9	*Motherboard*
10	*Memory*

The appearance of each item in the transaction is counted. Support count is the frequency number of each item in the transaction. Suppose n is an integer number, L_n is the number of item in the itemset.Table 2 shows the support count of the itemset. If the minimum support count is 4 then the Frequent Itemset is shown in Table 3.

Table 2. Support Count of Each Item

L_n	**Product**	**ID**	***Support count***
L_1	*Processor*	1,2,3,4,6,7	6
L_1	*Motherboard*	1,2,4,6,8,9	6
L_1	*Memory*	1,2,7,8,10	5
L_2	*Processor, motherboard*	1,2,4,6	4
L_2	*Processor, memory*	1,2,7	3
L_2	*Motherboard, memory*	1,2,8	3
L_3	*Processor, motherboard, memory*	1,2	2

Table 3. The Frequent Itemset with minimum support count 4

Products
Processor
Motherboard
Memory
Processor, motherboard

2. METHODS

This research focuses on the application development of parallel FIM based on MapReduce by using Cascading. The application is used to find the FIM in Amazon product co-purchasing network metadata [21]. The time needed to execute is observed. The effect of data size and support number are observed. The observations are used to determine the complexity.

2.1 Data preprocessing

The experiment uses Amazon product co-purchasing network metadata. It is 35,4 MB data which contains the product metadata and review information about 548,552 different products such as Books, music CDs, DVDs and VHS video tapes. For each product, the following information is available: title, sales rank, list of similar products, detailed product categorization, and product reviews (time, customer, rating, number of votes, number of people

that found the review helpful).

The first step of the experiment is transforming the experiment data into transaction data. The transaction data consists of two columns, the customer column, and the ASIN (Amazon Standard Identification Number) columns. This is carried out by using MapReduce Model. The Amazon dataset is inserted into the Hadoop Distributed File System (HDFS) for subsequent processing by Hadoop which gives output key-value pair of <Customer ID, Item purchased>. Figure 4 shows the data preprocessing.

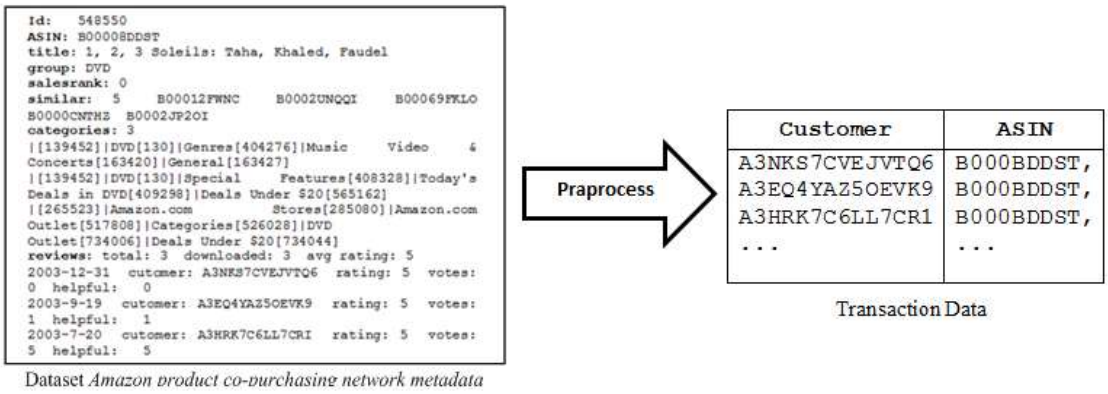

Figure 4. Data preprocessing

2.2 Algorithm design and program implementation

In this experiment, L_1 and L_2 itemsets are mined from the transactional data. The transactional data are put into the Cascading input tab. The L_1 itemsets are mined during the transactional data flow from the input tab to the output tab. The Cascading output tab outputs the L_1 itemsets. This process is depicted in Figure 5.

The output of the process in Figure 5 is used as the input of finding the L_2 itemsets. In this process, **`HadoopDistributedCache`** is used to take the L_1, followed by the process in the pipe which same with the process in Figure 5. The output of this process is L_2 where the key is 2-Frequent itemsets and the value is the support count. Figure 7 shows the flowchart of mining L_k itemsets in the pipe.

The implementation of flowcharts in Figure 5, 6, and 7 are started by defining the input tap and the output tap. The program code is shown in Figure 8. This step is followed by creating the pipe. It contains the main operations of FIM, namely **`Each`**, **`GroupBy`**, and **`Every`** operations. The program code is shown in Figure 9.

The detail process of finding the L_1 itemsets in the pipe is depicted in Figure 7. In the Cascading, the **`Each`**, **`GroupBy`** and **`Every`** operation are abstractions of MapReduce functions. Transaction data is processed one by one by **`Each`** operation. The operation takes *itemID* and converts each *itemID* into *<itemID, 1>*. Then, the **`GroupBy`** operation groups the data by itemID. The **`Every`** operation counts the appearance of the item. The **`Every`** operation gives Frequent Itemset in the format of *<itemID, support count>*.

The **`Each`** operation works for the individual tuple. It needs stream tuples which will be processed by **`Each`** operation. Figure 10 shows the **`CreateL1`** class which will act as the tuples stream. The **`Each`** operation is same with the map phase in MapReduce system.

The **`GroupBy`** operation is used to group the result of the **`Each`** operation. The **`GroupBy`** operation is same with the reduce phase in MapReduce system. The **`Every`** operation works for a group of tuples. This operation needs an aggregator. The aggregator code for FIM is shown in Figure 11.

The sequential processes in Figure 5, 6, and 7 can be duplicated by using MapReduce. The transaction data is converted into *<CID, item>* by HDFS. HDFS also distributes the transaction data to the mapper. The output of the mapper is *<key, 1>* where the *key* is the *CustomerID* and *1* as the value. The **`GroupBy`** operation groups all the outputs of the mapper

based on the key. The results of this operation are the candidate of itemset (C_k) in the form of *<Item, {1..n}>*. Then, the reducer uses the **`Every`** operation to add the value of the itemset candidate. The reducer outputs the key and its support count. The final result of L_1 itemsets is the union of all reducer output. Figure 12 shows the detailed process of L_2 itemsets by using MapReduce. The mappers give output in the form of *<item1, item2, 1>*. The **`GroupBy`** operations give output in the form of *<item1, item2, {1..n}>*. The reducers give output in the form of *<frequent-2 itemsets, count>*. The union of these outputs gives the L_2 final result.

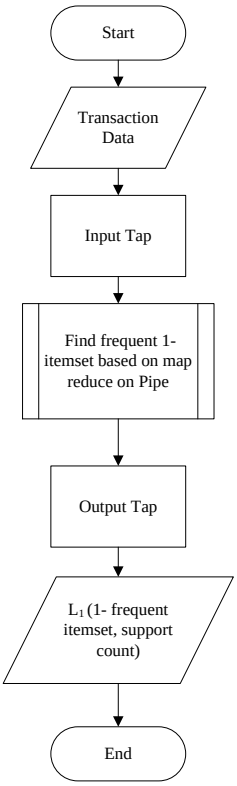

Figure 5. Mining L_1

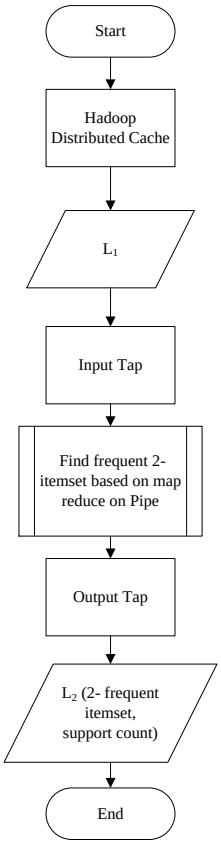

Figure 6. Mining L_2

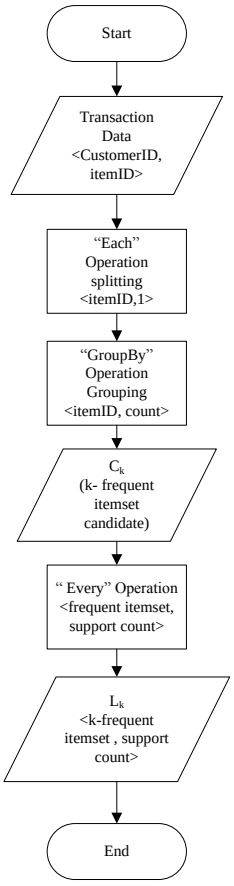

Figure 7. Mining L_k in the pipe

```
// Input Tap
Fields sourceField = new Fields("customerID", "products");
Scheme sourceScheme = new TextDelimited(sourceField);
Tap source = new Hfs(sourceScheme, inputFolder);
// Output Tap
Fields sinkField = new Fields("items", "count");
Scheme sinkScheme = new TextDelimited(sinkField);
Hfs sink = new Hfs(sinkScheme, outputFolder, SinkMode.REPLACE);
```

Figure 8. The definition of the input tap and the output tap.

```
//Operasi Each
assembly = new Each(assembly, new CreateL1(new Fields("items")));
//Operasi group by
assembly = new GroupBy(assembly, new Fields("items"));
//Operasi Every
assembly = new Every(assembly, new FrequentItem(500, new Fields("count")));
```

Figure 9. The main operations of FIM.

```
public class CreateL1 extends BaseOperation<Context> implements Function<Context> {

    public CreateL1(Fields fields) {
        super(1, fields);

    }

    public void operate(FlowProcess flowProcess, FunctionCall<Context> functionCall) {
        // Panggil Fungsi

        TupleEntry arguments = functionCall.getArguments();

        String products = arguments.getString("products");
        String[] product = products.split("\\,");
        for (int i = 0; i < product.length; i++) {
            Tuple tuple = new Tuple();
            tuple.add(product[i]);
            functionCall.getOutputCollector().add(tuple);
        }
```

Figure 10. The tuples stream codes

```
public class FrequentItem extends BaseOperation<FrequentItem.Context> implements Aggregator<FrequentItem.Context>
    class Context {
        int size;
    }
    private int support;
    private Path path;
    private JobConf conf;
    public FrequentItem(int support, Fields fields) {
        super(1,fields);
        this.support = support;
    }
    public void aggregate(FlowProcess flowProcess, AggregatorCall<Context> aggregatorCall) {
        Context context = aggregatorCall.getContext();
        context.size++;
    }
    public void complete(FlowProcess flowProcess, AggregatorCall<Context> aggregatorCall) {
        Context context = aggregatorCall.getContext();
        if(context.size >= this.support){
            Tuple tuple = new Tuple();
            tuple.add(context.size);
            aggregatorCall.getOutputCollector().add(tuple);
        }
    }
```

Figure 11. The aggregator code for FIM

2.3 Experiments

Two experiments have been done in this research [22]. The MapReduce and the non-MapReduce processes for L_1 and L_2 FIM have been observed. Four values of support count are used: 50, 75, 100, and 125. These support counts are used for three different size data. For each experiment, the time needed to accomplish the FIM processes are observed.

3. RESULTS AND DISCUSSIONS

3.1 Results

The first step of the experiment is transforming the transactional data into key-value pair data of *CustomerID* and *itemID*. This process gives 5.524.141 bytes which consist of 156.852 transactional data. The L_2 FIM is mined from three different size transactional data: 156.852, 78.426, and 39.213. Table 4 shows the experiment result. Figure 13 shows the comparison of L_2 FIM execution time on a non-MapReduce system. Figure 14 shows the comparison of L_2 FIM execution time on MapReduce system. Figure 15 shows the comparison of the whole L_2 FIM execution time. The line at the bottom of Figure 15, actually represents all the execution time on MapReduce system.

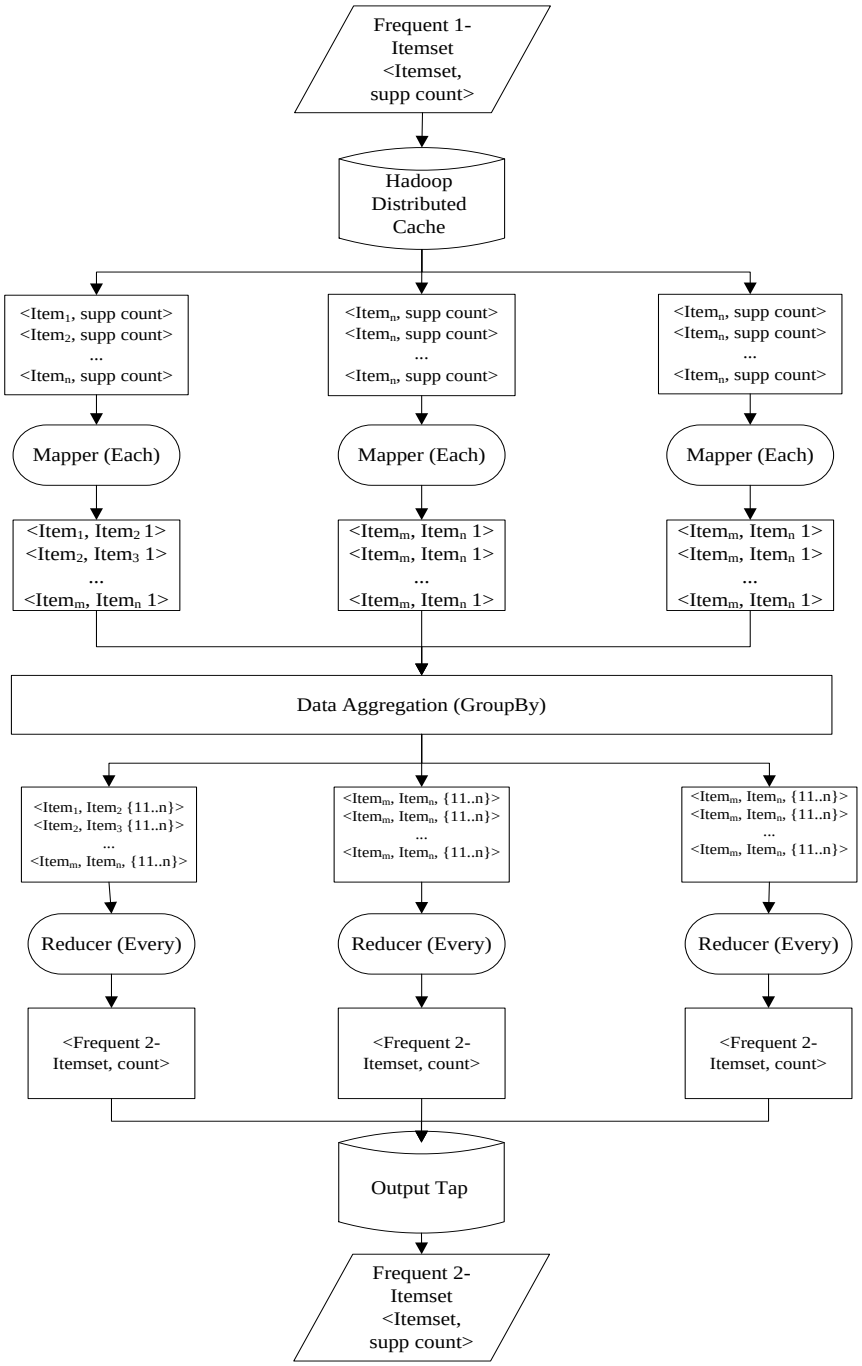

Figure 12. The L_2 Frequent itemset mining based on MapReduce

3.2 Discussions

Two processes of L_2 frequent itemset mining have been observed in the experiment, a non-MapReduce process dan MapReduce processes. Both processes worked on three different sizes data and four minimum support counts namely 50, 75, 100, and 125.

Both MapReduce and non-MapReduce processes give the same result, but as shown in Table 4, the time needed to accomplish the FIM are very different. The MapReduce system runs faster than the non-MapReduce system.

The change of minimum support count affect significantly the time needed to accomplish the L_2 FIM on the non-MapReduce system as shown in Figure 13, but not significant for the MapReduce system as shown in Figure 14. By comparing the whole experiment result in Figure

15, the execution time of a non MapReduce system increases in $O(n^2)$ as the number of datasets increasing. On the other hand, it decreases in $O(1/m)$ as the minimum support count increasing. The time complexity of L_2 FIM for a non-MapReduce system is $O(n^2/m)$ with n dataset and m minimum support count.

The execution time of MapReduce system increases in $O(n)$ as the number of datasets increasing, but the minimum support count does not affect the execution time, as shown in Figure 14 and Figure 15.

4. CONCLUSIONS

Based on the experiment and the discussion, it can be concluded that:

1. Cascading platform can be combined with Hadoop to implement MapReduce to mine the L_2 Frequent Itemset.
2. The execution time of the L_2 frequent itemset mining with Cascading platform is $O(n)$, while the regular process is $O(n^2/m)$, with n dataset and m minimum support count.

Table 4. L_2 FIM execution time

Number of transactional data	Time (seconds)							
	MapReduce				Non MapReduce			
	Min *Supp count* 50	Min *Supp count* 75	Min *Supp count* 100	Min*Supp count* 125	Min *Supp count* 50	Min *Supp count* 75	Min *Supp count* 100	Min*Supp count* 125
39.213	12,723	12,556	12,530	12,675	42,52	23,663	19,32	17,36
78.426	24,431	23,543	18,398	22,274	303,253	132,52	91,15	64,83
156.852	34,169	33,826	32,658	33,423	2572,76	1101,61	643,92	404,105

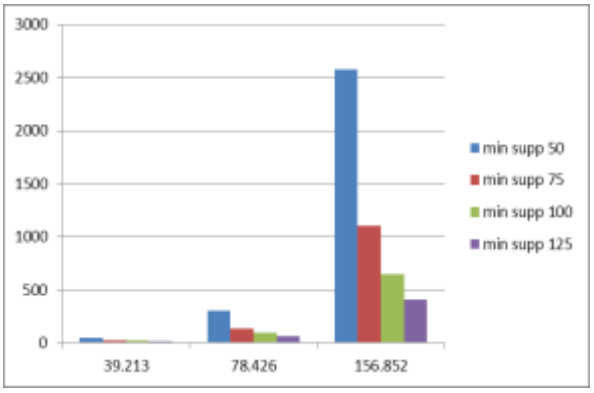

Figure 13. Comparison of L_2 FIM execution time on non MapReduce system

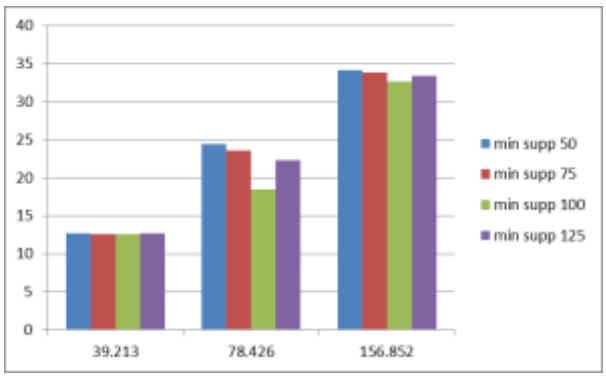

Figure 14. Comparison of L_2 FIM execution time on MapReduce system

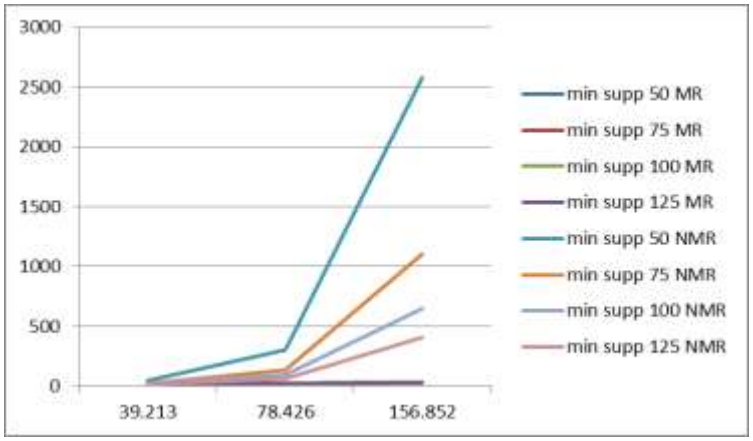

Figure 15. Comparison of L_2 FIM time complexity

REFERENCES

[1] X. Jiang and G. Sun, "MapReduce-based Frequent Itemset Mining for Analysis of Electronic Evidence ", Eight International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE), 2013. Available: http://ieeexplore.ieee.org/document/6911549/ [Accessed: 19-Mar-2018].

[2] X. Li, "An Algorithm for Mining Frequent Itemsets from Library Big Data ", Journal of Software, Vol. 9, No. 9, September 2014. Available: http://www.jsoftware.us/vol9/jsw0909-18.pdf [Accessed: 19-Mar-2018].

[3] O. Yahya, O. Hegazy, and E. Ezat, "An Efficient Implementation of Apriori Algorithm Based on Hadoop-MapReduce Model", International Journal of Reviews in Computing, 31st Dec 2012, Vol.12, pp.59-67 [Online]. Available: http://www.ijric.org/volumes/Vo12/Vol12No7.pdf [Accessed:19-Feb-2018].

[4] G.P. Chen, Y. B. Yang, and Y. Zhang, "MapReduce-based Balanced Mining for Closed Frequent Itemset", IEEE 19th International Conference on Web Services, 2012. Available : http://ieeexplore.ieee.org/document/6257941/ [Accessed: 19-Mar-2018].

[5] T. Ramakrishnudu and R.B.V. Subramanyam, "Mining Interesting Infrequent Itemsets from Very Large Data based on MapReduce Framework", I.J. Intelligent Systems and Applications, 2015, 07, 44-49, Published Online June 2015 in MECS (http://www.mecs-press.org/). Available: http://www.mecs-press.org/ijisa/ijisa-v7-n7/IJISA-V7-N7-6.pdf [Accessed: 19-Mar-2018].

[6] Y. H. Liang and S.Y. Wu, "Sequence-Growth : A Scalable and Effective Frequent Itemset Mining Algorithm for Big Data Based on MapReduce Framework ", IEEE International Congress on Big Data, 2015. Available: http://ieeexplore.ieee.org/document/7207249/ [Accessed: 19-Mar-2018].

[7] C. V. Suneel, K. Prasanna, and M.R. Kumar, "Frequent Data Partitioning using Parallel Mining Item Sets and MapReduce", International Journal of Scientific Research in Computer Science, Engineering and Information Technology, Volume 2 | Issue 4 |, 2017. Available: http://ijsrcseit.com/CSEIT1724152 [Accessed: 19-Mar-2018].

[8] B. He, H. Zhang and J. Pei, "The Mining Algorithm of Frequent Itemsets based on Mapreduce and FP-tree", International Conference on Computer Network, Electronic and Automation, 2017. Available: https://www.computer.org/csdl/proceedings/iccnea/2017/3981/00/3981a108.pdf [Accessed: 19-Mar-2018].

[9] F. Kovacs and J. Illes, "Frequent Itemset Mining on Hadoop", IEEE 9th International Conference on Computational Cybernetics (ICCC 2013), July 8-10, 2013, pp.241–245. Available: http://ieeexplore.ieee.org/document/6617596/ [Accessed: 19-Mar-2018].

[10] H. Chaudhary, "MapReduce Based Frequent Itemset Mining Algorithm on Stream Data", Global Conference on Communication Technologies (GCCT 2015), pp.598–603, 2015. Available: http://ieeexplore.ieee.org/document/7342732/ [Accessed: 19-Mar-2018].

[11] S. Saha and M. S. I. Islam, "Comparative Analysis of Mapreduce Framework for Efficient Frequent Itemset Mining in Social Network Data", Global Journal of Computer Science and Technology Cloud and Distributed, 2016, Volume 16 Issue 3. Available : https://globaljournals.org/GJCST_Volume16/7-Comparative-Analysis-of-Mapreduce.pdf [Accessed: 19-Mar-2018].

[12] M.A. Shinde and K.P. Adhiya, "Frequent Itemset Mining Algorithms for Big Data using MapReduce Technique - A Review", International Conference on Global Trends in Engineering, Technology and Management (ICGTETM-2016), 2016. Available: http://www.ijettjournal.org/Special%20issue/ICGTETM-2016/ICGTETM_2016_paper_131.pdf [Accessed: 19-Mar-2018].

[13] A. Padmapriya and R. Venkatachalam, "Collaborative-Frequent Itemset Mining of Big Data Using Mapreduce Framework", International Journal of Computer Science and Engineering (NCSACT–2017), 2017. Available: http://www.internationaljournalssrg.org/IJCSE/2017/Special-Issues/NCSACT/IJCSE-NCSACT-P119.pdf [Accessed: 19-Mar-2018].

[14] S. Tribhuvan and B.P. Vasgi, "Parallel Frequent Itemset Mining for Big Datasets using Hadoop-MapReduce Paradigm", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 6, Issue 6, June 2017. Available: https://www.ijarcce.com/upload/2017/june-17/IJARCCE%2035.pdf [Accessed: 19-Mar-2018].

[15] A. N. Nandakumar and N. Yambem, "A Survey on Data Mining Algorithms on Apache Hadoop Platform", International Journal of Emerging Technology and Advanced Engineering. Vol. 4, Issue 1, January 2014. Available: http://www.ijetae.com/files/Volume4Issue1/IJETAE_0114_95.pdf [Accessed: 19-Mar-2018].

[16] M. R. Ghazia and D. Gangodkara, "Hadoop, MapReduce and HDFS: A Developers Perspective", International Conference on Intelligent Computing, Communication & Convergence (ICCC-2015), 2015. Available: https://www.researchgate.net/publication/277935711_Hadoop_MapReduce_and_HDFS_a_developers_perspective [Accessed: 19-Mar-2018].

[17] ----, "Cascading 2 User Guide", Concurrent, Inc., Publication date October 2012. Available: http://docs.cascading.org/cascading/2.0/userguide/pdf/userguide.pdf [Accessed: 19-Mar-2018].

[18] S. Perera and T. Gunarathne, "Hadoop MapReduce Cookbook", February 2013, Packt Publishing Ltd.

[19] T. White, " Hadoop: The Definitive Guide", 2015, O'Reilly Media, Inc.

[20] P. Nathan, "Enterprise Data Workflows with Cascading", 2013, O'Reilly Media, Inc.

[21] *http://snap.stanford.edu/data/#amazon* [Accessed: 19-Mar-2018].

[22] A. Nursanti, "Frequent Itemset Finding Based On Mapreduce Using Cascading Platform", 2017. Available : *http://etd.repository.ugm.ac.id/index.php?mod=penelitian_detail&sub=PenelitianDetail&act=view&typ=html&buku_id=107287&obyek_id=4* [Accessed: 19-Mar-2018].