TOWARDS EFFICIENT AND SCALABLE DATA MINING USING SPARK

Jie Deng[1], Zhiguo Qu[2], Yongxu Zhu[1], Gabriel-Miro Muntean[2] and Xiaojun Wang[2]

The Rince Institute,
Dublin City University, Ireland
E-mail:[1]{jie.deng3,zhu.zhuyonz2}@mail.dcu.ie
[2]{zhiguo.qu,gabriel.muntean,xiaojun.wang}@dcu.ie

Keywords: PrefixSpan algorithm; MapReduce model; Spark platform; Distributed system.

Abstract

Following the requirements of discovery of valuable information from data increasing rapidly, data mining technologies have drawn people's attention for the last decade. However, the big data era makes even higher demands from the data mining technologies in terms of both processing speed and data amounts. Any data mining algorithm itself can hardly meet these requirements towards effective processing of big data, so distributed systems are proposed to be used. In this paper, a novel method of integrating a sequential pattern mining algorithm with a fast large-scale data processing engine Spark is proposed to mine patterns in big data. We use the well-known algorithm PrefixSpan as an example to demonstrate how this method helps handle massive data rapidly and conveniently. The experiments show that this method can make full use of cluster computing resources to accelerate the mining process, with a better performance than the common platform Hadoop.

1 Introduction

With the development of information technology, data can be collected faster and more easily. This raises the problem of how to extract valuable information from data. Various techniques have been proposed including statistical analysis, data mining and machine learning, etc. These techniques have different characteristics and could be used on different areas of data analysis. One of the most interesting techniques is data mining, which mainly focuses on knowledge discovery from datasets. It discovers patterns and transforms these patterns into understandable information for users.

One of the classic algorithms in data mining is PrefixSpan [1, 2], which was originally proposed by Jian Pei in 2001. Many improvements have been made on this algorithm since then, such as construction of the projected database with restriction [3], removing infrequent items at the beginning of the mining process to reduce the search space, or even performing load balancing between parallel PrefixSpan processes [4, 5]. Furthermore, this algorithm has been used in different areas, such as malware detection [6] and intrusion detection [7].

However, with the challenge of big data, traditional data mining algorithms like PrefixSpan do not have the ability to process massive datasets [8]. Big data processing has been proved to be useful in many different areas such as wireless sensor network analysis [9], and market analysis [10]. The analysis of big data requires algorithms to handle more varied and complex structures with difficulties in storing, analyzing and visualizing [11]. Different storage and processing methods [12] have to be implemented to meet the requirement of processing large-volume, growing datasets [13]. Although different frameworks have been proposed to solve this issue [14], most of them are grid based [15] and are using the MapReduce model [16,17,18] to process the data; however, specialised algorithms running on the framework have not still been properly explored.

In this paper, we proposed a method of integrating sequential pattern mining algorithms with a distributed data processing engine Spark. Since no algorithm has been proposed specialized for the distribution system, porting existing algorithms to the distribution system would be a rational choice.

The rest of this paper is organized as follows, section 2 gives an introduction of distribution platforms including MapReduce and Spark, section 3 gives an example of build the classical algorithm PrefixSpan on both platforms. All the experiment results are in section 4 which compared the performance of Spark on different number of nodes and with Hadoop as well.

2 Distribution platform

This section provides the architecture of two well-known distributed frameworks. As traditional data mining algorithms could not fulfil the requirements of massive data processing, new methods are introduced for data mining algorithms to handle big data. MapReduce and RDD are two such methods.

2.1 MapReduce

MapReduce is a parallelizable data process framework aiming to provide a generic method to processing data on cluster or a grid. It has been used in many different areas such as graph pattern analysis [19, 20], itemset mining [21], support vector machine [22] and also sequential pattern mining [23]. To make full use of computing resources and storage resources in

cluster, HDFS is always used to store data files as multiple copies, so that MapReduce can take advantage of locality of data, and decrease data transmission time.
MapReduce employs a master-slave architecture. In either Map or Reduce phases, the core concept is that the master distributes processing tasks to slave nodes, and these slave nodes will return processing result after finishing the job. There is a major difference between Map and Reduce. The Map function always faces the minimum fragments of datasets, and the results are always obtained by manipulating the fragment itself, such as characters' filtering. It can be described as:

map :: (key1, value1) => list (key2, value2)) (1)

The input of Reduce function is from the output of Map phase. Normally, the Reduce function aims to combine the same or different items, get result by statistic or mathematic processing of these items, such as summarize. It can be described as:

reduce :: (key2, list (value2)) => list (value3) (2)

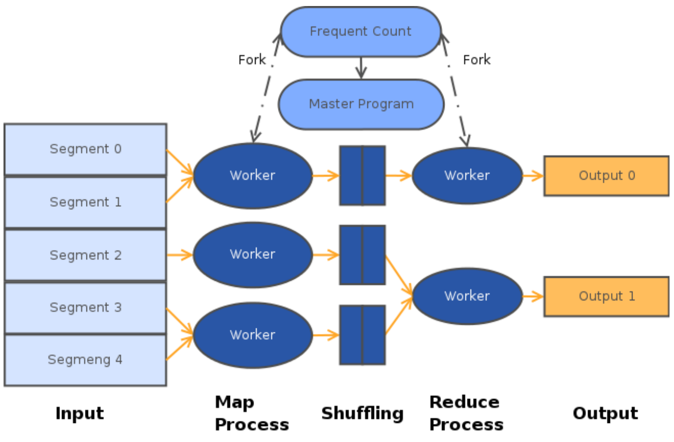

Figure 1 Architecture of MapReduce

2.2 Spark

Based on the fundamental concept of Hadoop, Spark [24] is a more effective cluster computing system that makes data processing easier and faster [25]. By introducing a new processing method and data structure RDD (resilient distributed dataset) [26], Spark can provide a wrap for distribution computing, so the program itself does not need to implement MapReduce. Furthermore, Spark also offers different levels of storage, ranging from memory and hard disk to stream [27]. It could greatly speed up the process if program prefers to use memory compared with HDFS in Hadoop.
An RDD can be seen as an encapsulation of both distributed data and operation. Physically, it is a collection of objects partitioned across the cluster, but logically, RDD also provides a set of operations including map, reduce and other common behaviours used in distributed computing. By using RDD, the program cares less about storage, operation and processing flow.

The processing function does not have to set MapReduce instance explicitly, which means the developer can focus more on the algorithm. The operations provided by RDD not only contain map and reduce, but also a few encapsulation of transformations like flat map, sort, filter and group. By using these methods, PrefixSpan program could save great efforts for fitting into the MapReduce framework, which means not only simplifying programming model, but also improving processing performance.
This section has introduced two of the famous distribution systems. Both of them could help a traditional data mining algorithms handle massive datasets. In the next section, we will use PrefixSpan as an example to show how to use these platforms to speed up the mining process.

3 Distributed PrefixSpan

In this section, implementing PrefixSpan on two distribution platforms is demonstrated. With the help of RDD, the algorithm not only makes use of the computing cluster, but also simplifies the programming model, while making the algorithm easily to be built and maintained.

3.1 PrefixSpan on MapReduce

PrefixSpan is a well-known data mining algorithm aiming to find sequential patterns from a dataset without the need of multiple scans to the dataset. This algorithm can find length-one items on each round, and is recursively performed afterwards to find all high frequency-occurring items to form patterns. During the recursion, the dataset is always purged based on the found frequent items, which largely reduces the dataset and further decreases the mining time. This algorithm has been proven as an effective algorithm compared with other point-based sequential pattern mining algorithms.
PrefixSpan uses a pattern-growth method, which can find patterns with only one scan of the sequential database. Three fundamental steps are involved into the PrefixSpan as follows.

1) Find length-1 sequential patterns, by scanning the database and counting the items in transactions. The frequent items will be length-1 patterns in this round.
2) Project database to reduce search space and build the project database for each pattern found in step 1, that is to remove all the items before the first appearance of the pattern in each transaction. Then the pattern itself will become a prefix to the projected database.
3) Find sequential patterns in the projected database. By following the same procedure in Step 1, the frequent patterns with lengths such as length-2, length-3, etc, in the projected database will also be found.

Finally, all the sequential patterns can be found by recursively performing these 3 steps above.
Related to the MapReduce framework, the same steps will be processed to implement the PrefixSpan. However, using a single Map and Reduce function may not be enough to finish the whole PrefixSpan process, so multiple MapReduce processed will be used.

The count frequency part in step 1 needs a whole MapReduce process to be finished. For every transaction going into the map function, the item appearance will be recorded and the reduce function will summarize the occurrence number for each item. The final result consists of the items with the occurrence number above the minimum support.

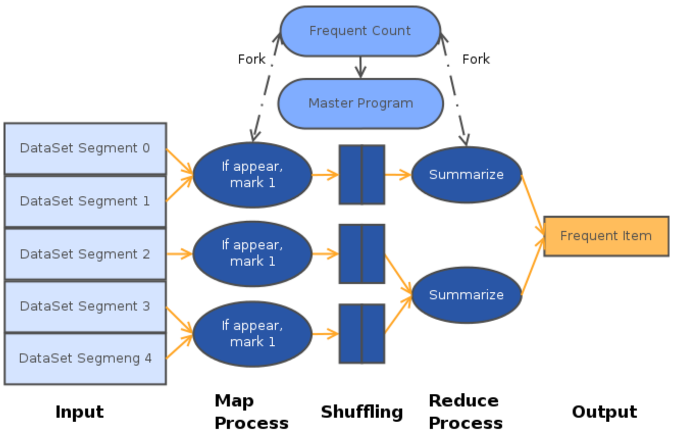

Figure 2 The frequent count phase of PrefixSpan

The project database part in step 2 also needs a whole MapReduce process to be accomplished. The input key is the result from the count frequency function, and the map function will remove items which have occurred before the given key. The reduce function can simply collect all the transactions after processing, and assemble them as a projected database. Similar to the general PrefixSpan process, the process is recursively performed for the first two MapReduce. All the output of the first MapReduce process will be the final result for the sequential pattern mining.

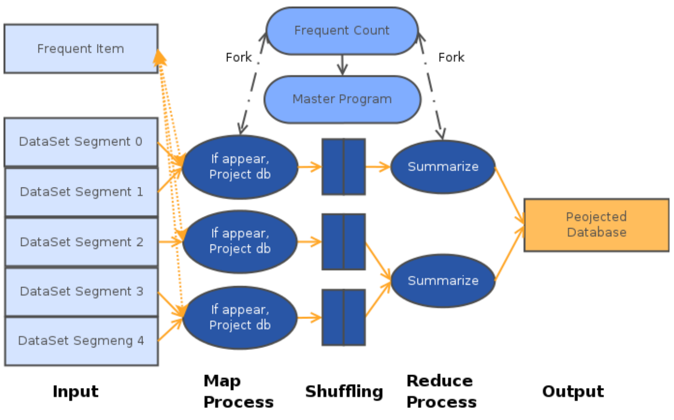

Figure 3 Project phase of PrefixSpan

With the implementation of the PrefixSpan on the MapReduce framework, the PrefixSpan algorithm can benefit from the distributed system to perform the computing on a cluster. Although the structure needs two individual MapReduce processes, this framework provides the algorithm the ability to handle massive data, which is impossible on a single machine.

3.2 PrefixSpan on Spark

Like in the previous case, the PrefixSpan program on Spark needs two parts to be complete. The concept of frequency calculation part is to count items in each transaction, and compare the occurrence number with minimum support. Flat map and map in RDD could be used for counting items, and each item will be stored in a tuple and for further comparison. The project database part could be implemented by map and filter functions. The map function will return each transaction after purging the prefix and the filter will be used to remove transactions without items.

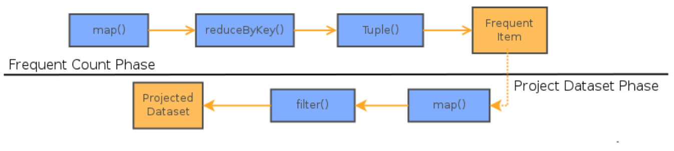

Figure 4 PrefixSpan based on RDD

By using RDD to write the PrefixSpan program, the processing model is simplified. Furthermore, all the intermediate results can be stored in the memory instead of hard disk, which will be another improvement to speed up the processing.

4 Experimental Testing

The experimental environment contains one master machine and four working machines. The master machine is a PC with an Intel Core i3-3220 CPU and 4 GB memory; the working nodes are virtual machines sharing two cores from an AMD Opteron 2350 CPU and 2 GB memory. The Hadoop platform is version 1.2.1, and Spark platform is version 0.8.0.

4.1 IBM Synthetic Data Generator

Test dataset used in experiments is generated by the IBM Quest Synthetic Data generator [28], which can generate emulated custom transactions. The parameters of the generator can be configured, including item number, pattern number, etc. The configuration used in this paper is shown in Table I.

Parameters	Value
Average items per transaction(T)	10
Number of transactions (D)	200k
Number of items (N)	1000
Average length of maximal pattern (I)	10

Table 1 Parameters of the synthetic data generator

4.2 Spark

The first experiment illustrated the performance of PrefixSpan using Spark against different minimum supports. By comparing the time consumption of PrefixSpan between different working nodes, we can find out how this algorithm scales by using the Spark platform.

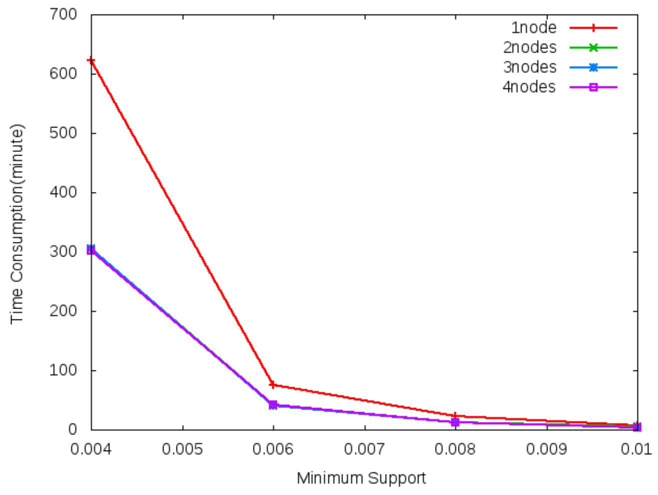

Figure 5 Time consumption of Spark with different nodes

From the result in figure 5 we can see that firstly, the processing time increases dramatically when the minimum support drops from 0.6% to 0.4%. Secondly, the processing time used in 2 nodes is much less than the processing time in 1 node. However, increasing nodes from 3 to 4 did not clearly improve the processing performance. The same trend can be found in another experiment of only run one frequency count function and one project database function, which is shown in Fig.6.

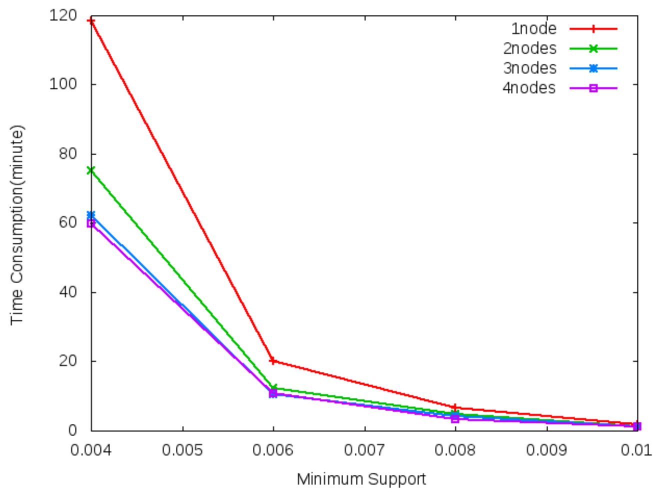

Figure 6 Time consumption of Spark with running partial PrefixSpan

4.3 Spark and Hadoop

The second experiment compares the performance of Spark and Hadoop. Both of the frameworks are running test dataset on minimum support 1% against different computing nodes. The result in Fig.7 shows Spark is generally 5 to 10 times faster than Hadoop. And this result can also be confirmed with the output from running PrefixSpan frequency count and project database functions only once, which shows in Fig. 8. Furthermore, by perform the same experiment with lower minimum support, the same trend can be found in figure 9 and figure 10.

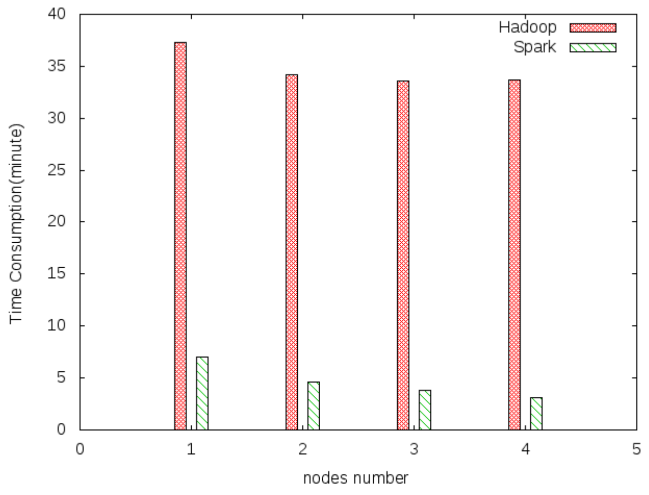

Figure 7 Time consumption of Hadoop and Spark with different nodes with minimum support 0.01

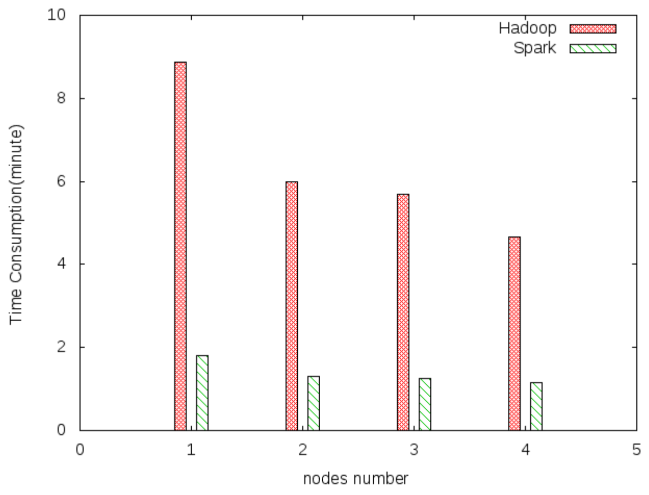

Figure 8 Time consumption of Hadoop and Spark running partial PrefixSpan with minimum support 0.01

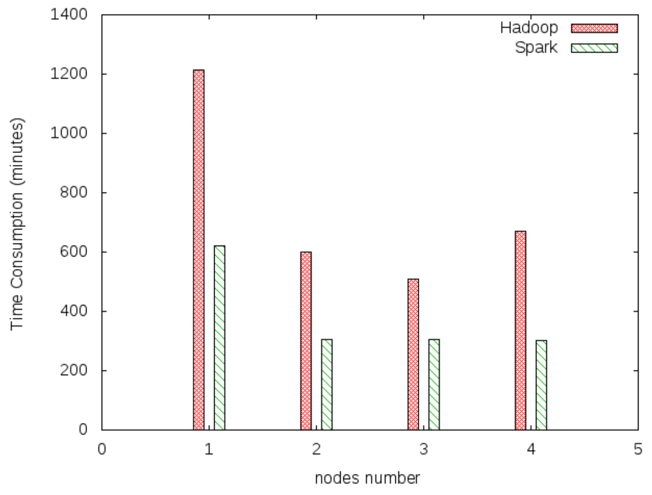

Figure 9 Time consumption of Hadoop and Spark with different nodes with minimum support 0.004

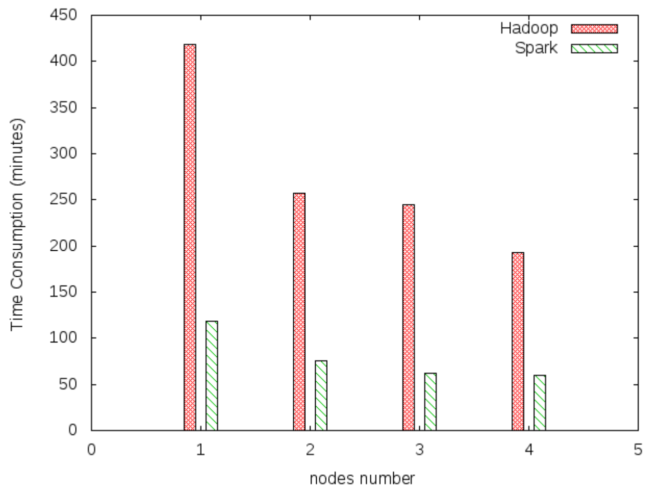

Figure 10 Time consumption of Hadoop and Spark running partial PrefixSpan with minimum support 0.004

By analyzing these experiment results, it can be clearly seen that the performance of Spark advantages that of Hadoop. By storing all the necessary information inside memory instead of hard disk, Spark can rapidly fetch necessary information without extra time. Besides, the encapsulation of operation and storage could simplify the program model, and further reduce development cost.

5 Conclusion

In this paper, we proposed a novel method of implementing traditional data mining algorithm on distributed systems. With this method, we can extend traditional data mining algorithms like PrefixSpan to handle massive data, significantly increasing the capability of the traditional data mining algorithms. Secondly, by using platform Spark, we can make full use of system memory, which leads to a better performance than the common distributed system Hadoop. Furthermore, with the benefit of RDD, the structure of the program does not have to follow the MapReduce explicitly, so the mining algorithm can more easily be implemented and maintained.

However, the experiment results also show that the performance does not scale linearly or more otherwise with the increase of number of working nodes, especially when the working node number increases to three and more. The reason may be that the overload of these distributed systems affects the system performance as well, which means the more nodes are included in the system, the more time and resources are spent on communication between these nodes. Future work may focus on the optimization towards specific mining algorithms, and the structure and procedure of the relevant implementations could be improved to fit the architecture of distributed system as well.

Acknowledgements

This work was funded by Enterprise Ireland Innovation Partnership Programme with Ericsson Ireland under grant agreement IP/2011/0135 [29].

References

[1] Jian Pei, H Pinto, and Qiming Chen. “Prefixspan: Mining sequential patterns effciently by prefix-projected pattern growth”. Data Engineering, 2001. Proceedings. 17th International Conference on, (2001).

[2] Jian Pei, Jiawei Han, and B Mortazavi-Asl. “Mining sequential patterns by pattern-growth: The prefixspan approach”. Knowledge and Data Engineering, IEEE Transactions on, pp. 1424-1440, (2004).

[3] LIU Pei-yu, Gong Wei, and JIA Xian. “An Improved PrefixSpan Algorithm Research for Sequential Pattern Mining”, IT in Medicine and Education (ITME), 2011 International Symposium on. (1995).

[4] Jie Deng, Zhiguo Qu, Yongxu Zhu, Gabriel-Miro Muntean and Xiaojun Wang,"Performance Evaluation and Optimization of A Hybrid Temporal Pattern Mining Algorithm", Information Technology & Telecommunications Conference,(2014).

[5] Makoto Takaki, Keiichi Tamura, and Hajime Kitakami. “Dynamic Load Balancing Technique for Modifyed PrefixSpan on a Grid Environment with Distributed Worker Model”. PDPTA, (2006).

[6] Lina Wang, Xiaobin Tan, Jianfeng Pan, and Hongsheng Xi. “Application of PrefixSpan* Algorithm in Malware Detection Expert System”. 2009 First International Workshop on Education Technology and Computer Science, pp. 448-452, (2009).

[7] Qingsen Xie and Tianqi Yang. “Application of the improved PrefixSpan algorithm in Intrusion Detection”. In 2010 8th World Congress on Intelligent Control and Automation, pp. 6099-6103. IEEE, (2010).

[8] Avita Katal, Mohammad Wazid, and R. H. Goudar. “Big data: Issues, challenges, tools and Good practices”. 2013 Sixth International Conference on Contemporary Computing (IC3), pp. 404-409, (2013).

[9] Z. Yuan and G.-M. Muntean, "A Prioritized Adaptive Scheme for Multimedia Services over IEEE 802.11

WLANs", IEEE Transactions on Network and Service Management, no.4, vol.10, pp.340 - 355, (2013).

[10] Akinori Abe. “Curating and Mining (Big) Data”. 2013 IEEE 13th International Conference on Data Mining Workshops, pp. 664-671, (2013).

[11] Seref Sagiroglu and Duygu Sinanc. “Big data: A review”. 2013 International Conference on Collaboration Technologies and Systems (CTS), pp. 42-47, (2013).

[12] Yang Song, Gabriel Alatorre, Nagapramod Mandagere, and Aameek Singh. “Storage Mining: Where IT Management Meets Big Data Analytics”. 2013 IEEE International Congress on Big Data, pp. 421-422, (2013).

[13] Xindong Wu, Xingquan Zhu, and Senior Member. “Data Mining with Big Data”. pp. 97-107, (2014).

[14] Firat Tekiner and John a. Keane. “Big Data Framework”. 2013 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1494-1499, (2013).

[15] Dan Garlasu, Virginia Sandulescu, Ionela Halcu, Giorgian Neculoiu, Oana Grigoriu, Mariana Marinescu, and Viorel Marinescu. “A big data implementation based on Grid computing”. 2013 11th RoEduNet International Conference, pp. 1-4, (2013).

[16] Hongyong Yu and Deshuai Wang. “Mass log data processing and mining based on Hadoop and cloud computing”. 2012 7th International Conference on Computer Science & Education (ICCSE), pp. 197-202, (2012).

[17] Faisal Zaman, Sebastian Robitzsch, Zhuo Wu, John Keeney, Sven van der Meer, and Gabriel-Miro Muntean."A Heuristic Correlation Algorithm for Data Reduction through Noise Detection in Stream-Based Communication Management Systems". 2014 Network Operations and Management Symposium, (2014).

[18] Antonia Azzini and Paolo Ceravolo. “Consistent Process Mining over Big Data Triple Stores”. 2013 IEEE International Congress on Big Data, pp. 54-61, (2013).

[19] Chun-Chieh Chen, Kuan-Wei Lee, Chih-Chieh Chang, De-Nian Yang, and Ming-Syan Chen. “Effecient large graph pattern mining for big data in the cloud”. 2013 IEEE International Conference on Big Data, pp. 531-536, (2013).

[20] Hui Chen, TY Lin, Zhibing Zhang, and Jie Zhong. “Parallel Mining Frequent Patterns over Big Transactional Data in Extended MapReduce”. ieeexplore.ieee.org, (60763002) pp.43-48, (2013).

[21] Sandy Moens, Emin Aksehirli, and Bart Goethals. “Frequent Itemset Mining for Big Data. 2013 IEEE International Conference on Big Data”, pp. 111-118, (2013).

[22] Dijun Luo, Chris Ding, and Heng Huang. “Parallelization with Multiplicative Algorithms for Big Data Mining”. 2012 IEEE 12th International Conference on Data Mining, pp. 489-498, (2012).

[23] Yong-qing Wei, Dong Liu, and Lin-shan Duan. “Distributed PrefixSpan algorithm based on MapReduce”. 2012 International Symposium on Information Technologies in Medicine and Education, pp. 901-904, (2012).

[24] Cliff Engle, Antonio Lupher, and Reynold Xin. Shark: fast data analysis using coarse-grained distributed memory. . . . Management of Data, pp. 1-4, (2012).

[25] Matei Zaharia and Mosharaf Chowdhury. “Spark: cluster computing with working sets in cloud computing”, (2010).

[26] Zaharia, Matei, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pp. 2-2, (2012).

[27] THTDM Zaharia, Alexandre Bayen, P Abbeel, and Timothy Hunter. “Large-Scale Online Expectation Maximization with Spark Streaming”. eecs.berkeley.edu, pp. 1-5.

[28] IBM data generator: http://sourceforge.net/projects/ibmquestdatagen/

[29] Dublin City University and Ericsson. EStream Project.