Review of Parallel Apriori Algorithm on MapReduce Framework for Performance Enhancement

Ruchi Agarwal, Sunny Singh and Satvik Vats

Abstract Finding frequent itemsets in the large transactional database is considered as one of the most and significant issues in data mining. Apriori is one of the popular algorithms that widely used as a solution of addressing the same issue. However, it has computing power shortage to deal with large data sets. Various modified Apriori-like algorithms have been proposed to enhance the performance of traditional Apriori algorithm that works on distributed platform. Developing efficient and fast computing algorithm to handle large data sets becomes a challenging task due to load balancing, synchronisation and fault-tolerance issue. In order to overcome these problems, MapReduce model comes into existence, originally introduced by Google. MapReduce model-based parallel Apriori algorithm finds the frequent itemsets from large data sets using a large number of computers in distributed computational environment. In this paper, we mainly focused on parallel Apriori algorithm and its different versions based on approaches used to implement them. We also explored on current major open issues and extensions of MapReduce framework along with future research directions.

Keywords Frequent itemsets · Parallel Apriori · Big data · Hadoop · MapReduce

R. Agarwal (✉) · S. Singh · S. Vats
Department of Computer Science and Engineering, Sharda University, Greater Noida, India
e-mail: ruchi.agarwal@sharda.ac.in

S. Singh
e-mail: aryans1027@gmail.com

S. Vats
e-mail: satvik.vats@gmail.com

V.B. Aggarwal et al. (eds.), *Big Data Analytics*, Advances in Intelligent Systems and Computing 654, https://doi.org/10.1007/978-981-10-6620-7_38

1 Introduction

We are living in Big Data era where data is growing exponentially with time and size of data is moving from terabytes to petabytes [1]. This trend brings out challenges to store this vast amount of data effectively and demands for analytical technology. Analysis of Big Data helps the organizations as well as government in decision-making and setting polices to provide better services to people. Various data mining tools are available from last decades to extract useful information, but they failed to process the large data sets because of time and space complexity. Association rules mining (ARM) technique [2] is used to find out the interesting patterns, sequences or itemsets from large database [3]. Apriori algorithm is used to implement ARM, but effectiveness of this algorithm reduces as the size of the data sets increases to compute, because of its iterative fashion of working which leads to further increment in the value of time complexity. Lots of work has been done to make Apriori algorithm run parallel to reduce the time complexity of traditional Apriori, originally proposed by R. Agarwal. As a result, several parallel Apriori algorithms come into existence such as count distribution (CD), candidate distribution (CaD) and data distribution (DD). These algorithms provide some key features such as dynamic itemset counting [4], data and task parallelism [5]. However, these algorithms come with some major weakness of synchronisation of data, communication issues due to message passing interface (MPI) framework which mainly support for homogeneous environment rather than heterogeneous environment and only work with low-level language like C and FORTRAN [6, 7]. Further, workload balancing [8] and fault-tolerance issue make them incapable to handle Big Data in distributed environments. Above problems lead to the development of MapReduce programming model, introduced by Google [9] for processing large database which enables the programmer to write programming code using map and reduce functions to run parallel applications. Google's MapReduce framework [10] is one of the current approaches which are available to process the Big Data using commodity machines or nodes in distributed computational environment. Hadoop provides platform to run the MapReduce programming model [11, 12] and enables the developers to code analytical applications under the hood of strong fault tolerance where guarantee is offered by Hadoop. Despite of various advantages of MapReduce model, it has also been criticised in terms of its limitation and complexity [13]. This leads to extensive research on MapReduce characteristics, to identify various issues in terms of performance and complexity of the model and current implementations [14–16]. To overcome these difficulties, various extensions are proposed where each one of the extensions fix one or more limitations and drawbacks of MapReduce framework. The scope of this paper is strictly limited to open issues and extensions of MapReduce model to enhance it, not to discuss generalised data-flow systems such as Spark, Dryad and Stratosphere.

This paper is organised as follows. Section 2 presents an overview of Big Data and MapReduce as a programming model under the title background study. Section 3 presents the parallel Apriori algorithm and its implementation on

MapReduce framework. Section 4 presents the open issues as limitations of MapReduce model and various extensions of MapReduce to improve it. We conclude in Sect. 5 with possible future research direction.

2 Background Study

2.1 Big Data and Its Characteristics

Generally, Big Data term is used to describe the data that is very large in size and yet growing exponentially with time. It can be characterised by using following four parameters, commonly known in terms of "4 V" parameters: (i) volume: refers to the size of data, (ii) velocity: refers to the speed of generation of data, (iii) variety: refers to the nature of data whether it is structured or unstructured data and (iv) variability: refers to inconsistency in the data. In current scenario, Big Data and its analysis are at the centre point of current science and business.

2.2 MapReduce as a Programming Model

MapReduce intends to perform flexible information processing in the cloud [9]. Many Programming models have been proposed under the name process models such as generic processing model, graph processing model and stream processing model to solve domain-specific applications. These models are used to improve the performance of NoSQL databases. MapReduce programming model comes under generic processing model that used to address the general application problems. MapReduce programs can be seen in two phases, map phase and reduce phase which consist of map function and reduce function, respectively, and input to each

Table 1 Classification of MapReduce algorithms

Class	Description	Example
First	Algorithms which are fundamentally based on single execution of MapReduce model	Factoring integers
Second	Algorithms which are adapted as a sequential execution of a constant number of MapReduce model	Clustering LARge Application (CLARA) [28]
Third	Algorithms which are iterative in nature and where each iteration requires execution of single MapReduce model only	Partitioning around medoids (PAM) [28]
Fourth	Algorithms which are iterative in nature and content of single iteration is significantly denoted as an execution of multiple MapReduce models	Conjugate gradient (CG) [29]

function are key-value pairs. MapReduce algorithms can be categorised into four classes, as shown in Table 1.

3 Parallel Apriori Algorithm

3.1 Parallel Apriori Algorithm on MapReduce

First and foremost, it is required to write parallel Apriori algorithm code in terms of map and reduce functions to run the application on MapReduce model. These two main functions of MapReduce model get the inputs in key-value pairs and generate the output in the key-value form also. The key step in parallel Apriori algorithm is to find out the frequent itemsets. Figure 1 shows the work flow of generation of frequent 1-itemsets.

First, HDFS divides the transactional database into data-chunks (default size of data-chunk is 64 MB) and distributes them among different machines in key-value form where key represents the Transactional ID (TID) and value denotes the list of items. Each mapper running on different machines fed by this key-value pairs and generates the output (key-value) pairs after reading one transaction at a time where key is further refined to represent each item and value is frequency of occurrence of item in the database. These outputs of mapper functions also are known as intermediate values, because these values are fed to combiner before to submit to reducers. Combiner has the task to shuffle and exchange the values using shuffle sort algorithm and consequently prepares a list having values linked with the same key. Here, key represents the item and value represents the support count $\geq$ minimum support of that item.

Reducer function has the main task to aggregate all key-value pairs and generates final output [17]. Here, frequent 1-itemsets are generated at the end into HDFS (storage unit) as output. Frequent k-itemsets are generated by each mapper after reading frequent itemsets from previous iteration and generate candidate itemsets on that basis. This process is done in iterative fashion to get frequent k-itemsets where each iterative step is same as generation of frequent 1-itemsets [7, 18].

3.2 Various Proposed Implementations of Parallel Apriori Algorithm on MapReduce

To reduce the time and space complexity of parallel Apriori algorithm, various Apriori-like algorithms have been proposed which execute on MapReduce framework. Broadly, these algorithms can be further classified based on 1-phase of MapReduce and combiner and k-phase of MapReduce approach which is used to

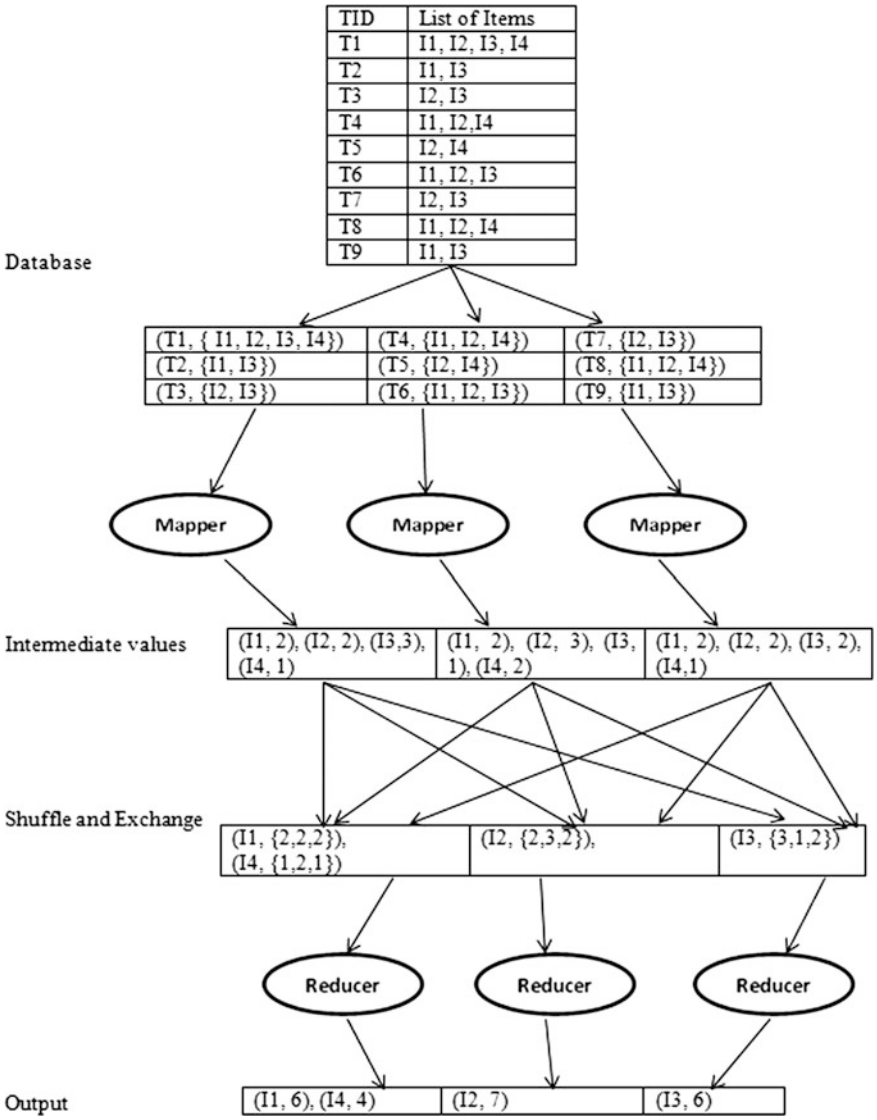

Fig. 1 Finding of frequent 1-itemsets

develop them. Algorithms having 1-phase of MapReduce approach execute single iteration of MapReduce job to extract all frequent itemsets. On the other hand, algorithms having *k*-phase of MapReduce approach execute multiple iterations of MapReduce job [19]. As a result of continuous research, an improved Apriori algorithm [20] comes into existence which further minimises the time complexity of parallel Apriori algorithm from $O(|L_k|^2)$ to $O(|V_{key}|^2/q)$ where L_k is the set of large

k-itemsets, V_{key} is the value list of ith key and q is the number of reducers. Further, pruning step of this algorithm is improved that leads to Improved Pruning Apriori (IP-Apriori) [21].

4 MapReduce Open Issues and Extensions

4.1 Performance Issues

MapReduce platform provides some key features such as scalability, fault tolerance to handle the data at large scale, but overall performance of this platform highly depends on the nature of application that is executed in distributed computational environment. To make MapReduce framework more suitable for Big Data handling and to improve the performance, various Hadoop extensions are suggested over the period such as index creation [22], data co-location, reuse the previously computed results and mechanisms dealing with computational skew.

4.2 Programming Model and Query Processing Issues

To code MapReduce applications, understanding of both system architecture and programming skills is required. The programming model of MapReduce has the limitation under its "batch" nature where data is needed to upload into the file system even when the same data set needs to be analysed many times. This programming model is also inappropriate for many classes of algorithm where results of one MapReduce job serving as the input for the next in case of complex queries analysis process. Consequently, a set of domain-specific systems have been emerged to extend the MapReduce programming model where high-level languages such as Java, Ruby, Python and various abstractions have been built to support MapReduce application development environment. Researchers proposed some model to implement iterative algorithms using MapReduce framework such as Hadoop, iHadoop [23], iMapReduce [24], Twister [25] and CloudClustering [26]. Apart from that, users have to spend more time in writing programs in the absence of expressiveness just like SQL. Therefore, it is required to enhance the MapReduce query capabilities [27].

4.3 MapReduce Extensions

To eliminate the limitations of MapReduce framework, researchers try to integrate the key features of parallel database and database to MapReduce programming

Table 2 MapReduce extensions and advantages

MapReduce extensions	Advantages
Hadoop++ [30]	Enhanced performance by injecting joining and indexing capabilities into Hadoop
Manimal [31]	Enables the MapReduce to analyse MapReduce programs automatically
CoHadoop [32]	Supports data storage of linked data at the same compute nodes
SkewReduce [33]	Handles workload by equally dividing input data to mitigate computational skew
SkewTune [34]	Reduces skew using both map and reduce phases at run-time
MapReduce Online [35]	Helps in online aggregation and stream processing to improve resource utilisation
EARL [36]	Allows incremental computations for early results using bootstrapping technique that is used to estimate the error in sampling data [36]
HAIL [37]	Binary PAX representation [38] is used in HAIL to maintain each physical block copy in a different sort order and preserves Hadoop's fault-tolerance properties
MRShare [39]	Provides the optimal grouping of queries to support sharing opportunities
ReStore [40]	Stores and reuses intermediate results of script after completion of tasks or sub-tasks

model which results in MapReduce extensions. Various MapReduce extensions with key advantages are listed in Table 2.

5 Conclusion and Future Research Direction

Based on our survey, both Apriori (traditional Apriori) and parallel Apriori algorithm versions are suffering from the problem of scanning the database multiple times, specially those based on *k*-phase of MapReduce approach which incurs high processing cost and generation of candidate itemsets that needs more memory space. We also focused on MapReduce capabilities, limitations as open issues and various proposed extensions. Open issues lead to various extensions or enhancements, and major enhancements are the result of integration of database with MapReduce, integration of indexing capabilities to MapReduce, integration of MapReduce with data warehouse capabilities and adding skew management in MapReduce.

Future research can be carried out in two dimensions to enhance the performance of parallel Apriori algorithm. One dimension leads to modification in joining and pruning steps of existing algorithm to enable it to support pipelining or use of alternative Apriori-like algorithms which are free from the problem of multiple times scanning of database. Second dimension of research leads to use of advanced

MapReduce framework such as i^2 MapReduce model which supports incremental problem-based algorithm or hybrid algorithms also to enhance the overall throughput of system.

References

1. Assunção, M.D., Calheiros, R.N., Bianchi, S., Netto, M.A., Buyya, R.: Big data computing and clouds: trends and future directions. J. Parallel Distrib. Comput. **79**, 3–15 (2015)
2. Agrawal, R., Ramakrishnan, S.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, vol. 1215, pp. 487–499 (1994)
3. Agrawal, R., Imieliński, T., Swami, A. Mining association rules between sets of items in large databases. ACM SIGMOD Rec. **22**(2), 207–216 (1993)
4. Brin, S., Motwani, R., Ullman, J.D., Tsur, S.: Dynamic itemset counting and implication rules for market basket data. ACM SIGMOD Rec. **26**(2), 255–264 (1997)
5. Dunham, M.H., Xiao, Y. Gruenwald, L., Hossain, Z.: A survey of association rules (2008). Retrieved 5 Jan 2001
6. Oruganti, S., Ding, Q., Tabrizi, N.: Exploring HADOOP as a platform for distributed association rule mining. In: FUTURE COMPUTING 2013, The Fifth International Conference on Future Computational Technologies and Applications, pp. 62–67 (2013)
7. Lin, M.-Y., Lee, P.-Y., Hsueh, S.-C.: Apriori-based frequent itemset mining algorithms on MapReduce. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, p. 76. ACM (2012)
8. Agrawal, R., Shafer, J.C.: Parallel mining of association rules. IEEE Trans. Knowl. Data Eng. **6**, 962–969 (1996)
9. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of the Operating Systems Design and Implementation (OSDI), pp 137–150 (2004)
10. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
11. He, H., Du, Z., Zhang, W., Chen, A.: Optimization strategy of Hadoop small file storage for big data in healthcare. J. Supercomput. 1–12 (2015)
12. Schneider, R.D.: Hadoop for Dummies®. Special edn. Wiley, Canada (2012)
13. Pavlo, A., Paulson, E. Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, pp. 165–178. ACM (2009)
14. Lee, K.-H., Lee, Y.-J., Choi, H. Chung, Y.D., Bongki, M.: Parallel data processing with MapReduce: a survey. ACM SIGMOD Rec. **40**(4), 11–20 (2012)
15. Jiang, D., Ooi, B.C., Shi, L., Wu, S.: The performance of Mapreduce: an in-depth study. Proc. VLDB Endowment **3**(1-2 (2010): 472-483
16. Goyal, A., Dadizadeh, S.: A survey on cloud computing. In: University of British Columbia Technical Report for CS 508, 55–58 (2009)
17. Kovacs, F., Illes, J: Frequent itemset mining on hadoop. In: 2013 IEEE 9th International Conference on Computational Cybernetics (ICCC), pp. 241–245. IEEE (2013)
18. Li, N., Zeng, L., He, Q., Shi, Z.: Parallel implementation of Apriori algorithm based on MapReduce. In: 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), pp. 236–241. IEEE (2012)

19. Li, L., Zhang, M.: The strategy of mining association rule based on cloud computing. In: 2011 International Conference on Business Computing and Global Informatization (BCGIN), pp. 475–478. IEEE (2011)
20. Al-Maolegi, M., Arkok, B. An improved Apriori algorithm for association rules. arXiv preprint arXiv:1403.3948 (2014)
21. Sequeira, J.V., Ansari, Z.: Analysis on improved pruning in Apriori algorithm. Int. J. Adv. Res. Comput. Sci. Softw. Eng. (IJARCSSE) **5**, 894–902 (2015)
22. Kang, W.L., Kim, H.G., Lee, Y.J.: Efficient indexing for OLAP query processing with MapReduce. In: Computer Science and Its Applications, pp. 783–788. Springer, Berlin, Heidelberg (2015)
23. Song, J., Guo, C., Zhang, Y., Zhu, Z., Yu, G.: Research on MapReduce based incremental iterative model and framework. IETE J. Res. **61**(1), 32–40 (2015)
24. Zhang, Y., Gao, Q., Gao, L., Wang, C.: Imapreduce: a distributed computing framework for iterative computation. J. Grid Comput. **10**(1), 47–68 (2012)
25. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J., Fox, G.: Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 810–818. ACM (2010)
26. Dave, A., Lu, W., Jackson, J., Barga, R.: CloudClustering: toward an iterative data processing pattern on the cloud. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp. 1132–1137. IEEE (2011)
27. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., Murthy, R.: Hive—a petabyte scale data warehouse using Hadoop. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE), pp. 996–1005. IEEE (2010)
28. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis, vol. 344. Wiley (2009)
29. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain (1994)
30. Dittrich, J., Quiané-Ruiz, J.-A., Jindal, A., Kargin, Y., Setty, V., Schad, J.: Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). Proc. VLDB Endowment **3**(1–2), 515–529 (2010)
31. Jahani, E., Cafarella, M.J., Ré, C.: Automatic optimization for MapReduce programs. Proc. VLDB Endowment **4**(6), 385–396 (2011)
32. Eltabakh, M.Y., Tian, Y., Özcan, F., Gemulla, R., Krettek, A., McPherson, J.: CoHadoop: flexible data placement and its exploitation in Hadoop. Proc. VLDB Endowment **4**(9), 575–585 (2011)
33. Kwon, Y.C., Balazinska, M., Howe, B., Rolia, J.: Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 75–86. ACM (2010)
34. Kwon, Y.C., Ren, K., Balazinska, M., Howe, B., Rolia, J.: Managing skew in Hadoop. IEEE Data Eng. Bull. **36**(1), 24–33 (2013)
35. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R.: MapReduce Online. NSDI **10**(4), 20 (2010)
36. Laptev, N., Zeng, K., Zaniolo, C.: Early accurate results for advanced analytics on mapreduce. Proc. VLDB Endowment **5**(10), 1028–1039 (2012)
37. Dittrich, J., Quiané-Ruiz, J.-A., Richter, S., Schuh, S., Jindal, A., Schad, J.: Only aggressive elephants are fast elephants. Proc. VLDB Endowment **5**(11), 1591–1602 (2012)
38. Ailamaki, A., DeWitt, D.J., Hill, M.D., Skounakis, M.: Weaving relations for cache performance. VLDB **1**, 169–180 (2001)
39. Nykiel, T., Potamias, M., Mishra, C., Kollios, G., Koudas, N.: MRShare: sharing across multiple queries in MapReduce. Proc. VLDB Endowment **3**(1–2), 494–505 (2010)
40. Elghandour, I., Aboulnaga, A.: ReStore: reusing results of MapReduce jobs. Proc. VLDB Endowment **5**(6), 586–597 (2012)