ISSN 2581 – 3781

AJCSE
ASIAN JOURNAL OF COMPUTER SCIENCE ENGINEERING

Available Online at www.ajcse.info
Asian Journal of Computer Science Engineering 2018;3(2):11-18

RESEARCH ARTICLE

Generating Frequent Itemsets by RElim on Hadoop Clusters

B. Usharani

Department of CSE, KL University, Andhra Pradesh, India

Received: 02-01-2018; Revised: 25-02-2018; Accepted: 12-03-2018

ABSTRACT

Data mining is considered as the process of extracting the useful information by finding the hidden information out of large chunks of dataset. Frequent itemset mining is the popular data mining methods. MapReduce has turn out to be an important distributed processing model for large-scale data-intensive applications like data mining. MapReduce is an efficient, scalable, and easy programming model for large-scale distributed data processing on a huge cluster of commodity computers. In this paper, RElim algorithm is implemented on MapReduce framework.

Key words: Association rule mining, frequent pattern mining, frequent itemset mining, Hadoop, RElim

INTRODUCTION

To handle storage resources across the cluster, Hadoop uses a distributed user-level file system. The file system Hadoop distributed file system (HDFS) is written in Java and measured for portability across heterogeneous hardware and software platforms. An important characteristic of Hadoop is the partitioning of data and computation across thousands of hosts and executing application computations in parallel. MapReduce is an easy programming model for large-scale distributed data processing and also used in cloud computing. Association rule mining (ARM) is an essential component of data mining. Data mining and knowledge discovery have appeared to mine useful, hidden and unknown patterns, and knowledge from large database. ARM is one of the mainly essential and accepted procedures of data mining which locates interesting correlation or association between set of items or attributes and also frequent patterns in large database.[1] The mainly usual application of ARM is in market basket analysis which examines the purchasing behavior of customers by discovering the frequent items purchased together. In addition to the many business application, it is also appropriate to bi-informatics, medical diagnosis, and text analysis.[2] Various ARM algorithms have been developed that diverges from each other in the way the approach they used. These approaches are based on candidate generation, without candidate generation, based on equivalence class clustering, maximal hypergraph clique clustering, and lattice traversal scheme.[3-5] When it comes to mine vast volume of data, these algorithms failed to verify scalability and efficiency. The major reasons behind this are the processing capacity, storage capacity, and RAM of a single machine.[6] For this reason, parallel and distributed algorithms are developed to present large-scale computing in ARM on several processors. These parallel and distributed algorithms progress the mining performance but also include some overheads such as partition of input data, workloads balancing, reduction in communication costs, and aggregation of information at local nodes to form the global information. There are a variety of such algorithms developed that deals with these issues in homogeneous computing environment.[7-10] These usual parallel and distributed algorithms are not suitable for heterogeneous environment such as heterogeneous cluster and grid environment.[11-18]

APACHE HADOOP MAPREDUCE FRAMEWORK

The word "MapReduce" originally referred to the proprietary Google technology.[44] MapReduce was first described Dean and Ghemawat[43]research paper from Google, by Jeffrey Dean and Sanjay Ghemawat, researchers in Google. MapReduce

Address for correspondence:
B. Usharani
E-mail: ushareddy.vja@gmail.com

is a patented software framework introduced by Google to support distributed computing on large datasets on clusters of computers. MapReduce is a functional programming model. It runs in the Hadoop background to provide scalability, simplicity, speed, recovery, and easy solutions for data processing. MapReduce is a parallel and distributed solution approach developed by Google for processing large datasets. MapReduce is used by Google and Yahoo to power their web search. Hadoop is a large-scale distributed batch processing infrastructure for parallel processing of big data on large cluster of commodity computers.[30] Hadoop is an open source project of Apache[23] which implemented Google's File System[31] as HDFS and Google's MapReduce[21] as Hadoop MapReduce programming model.

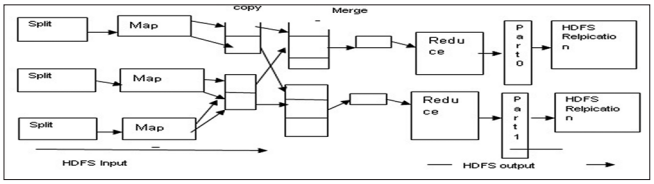

Figure 1: Overall view of Hadoop distributed file system

Hadoop MapReduce

MapReduce is a programming model considered for parallel processing of vast volumes of data by separating the job into independent tasks across a bulky number of machines. It uses two concepts:

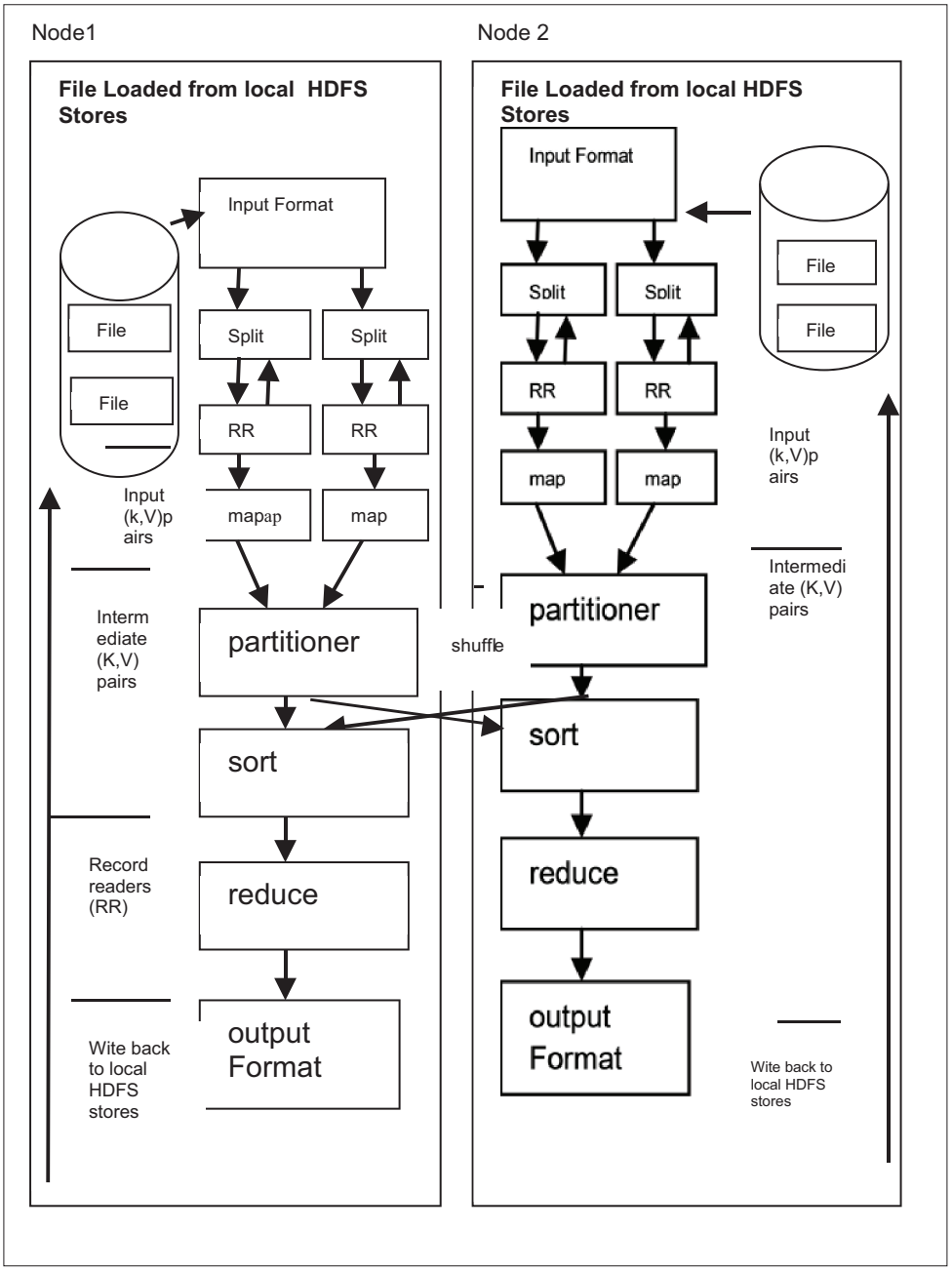

Figure 2: :Detailed Hadoop MapReduce data flow

Map and reduce. Based on it, a MapReduce program consists of two functions mapper and reducer which run on all machines in a Hadoop cluster. The input and output of these functions are in the form of (key and value) pairs.[30] MapReduce has two key components: Map and reduce. A map is a function which is used on a set of input values and computes a set of key/value pairs. Reduce is a function which takes these results and applies another function to the result of the map function. A reducer gets all the data for an individual "key" from all the mappers [Figure 1].

MapReduce programs are designed to compute huge volumes of data in a parallel fashion. This requires separating the workload across a large number of machines. This model would not scale to large clusters [Figure 2].[30]

All data elements in MapReduce cannot be updated. If in a mapping task you change an input (key and value) pair, it does not get reflected back in the input files; communication occurs by generating new output (key and value) pairs which are then promoted by the Hadoop system into the next phase of execution.[30]

Input and output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3> (Output). The mapper takes the input (k_1, v_1) pairs from HDFS and generates a list of intermediate (k_2, v_2) pairs. An optional combiner function is applied to decrease communication cost of transferring intermediate outputs of mappers to reducers. Output pairs of mapper are locally sorted and grouped on same key and provide for to the combiner to make local sum. The intermediate output pairs of combiners are shuffled and exchanged between machines to cluster all the pairs with the same key to a single reducer. This is the only one communication step takes place and handle by the Hadoop MapReduce platform. There is no other communication between mappers and reducers take place. The reducer takes (k_2, list [v_2]) values as input, put together sum of the values in list (v_2) and produce new pairs (k_3, v_3).[30,32] Figure 2 illustrates the workflow of MapReduce.

MapReduce incorporates a framework which supports MapReduce operations. A master controls the whole MapReduce process. The MapReduce framework is responsible for load balancing, reissuing task if node as failed or is to slow, etc. The master divides the input data into separate units and sends the individual chunks of data to the mapper machines and collects the information once a mapper is finished. If the mapper is finished, then the reducer machines will be allocated work. All key/value pairs with the same key will be transporting to the same reducer. Table 1 data are represented in diagrammatic form as shown in Figure 3.

MapReduce is a programming model since all the parallelization, intermachine communication and fault tolerance are held by run-time system.[21]

ARM

It is proposed by Agrawal *et al*., in 1993.[41] It is an important data mining model studied widely by the database and data mining. Initially ARM is used for market basket analysis to discover how items are purchased by customers.

Definition

ARM is a procedure which is meant to find frequent patterns, correlations, associations, or causal structures from datasets found in various kinds of databases such as relational databases, transactional databases, and other forms of data repositories.[42]

The major challenge found in frequent pattern mining is a number of result patterns. If the

Table 1: Input and output for map and reduce

Category	Input	Output
Map	<k1, v1>	list < k2, v2>
Reduce	<k2, list (v2)>	(k3, v3)

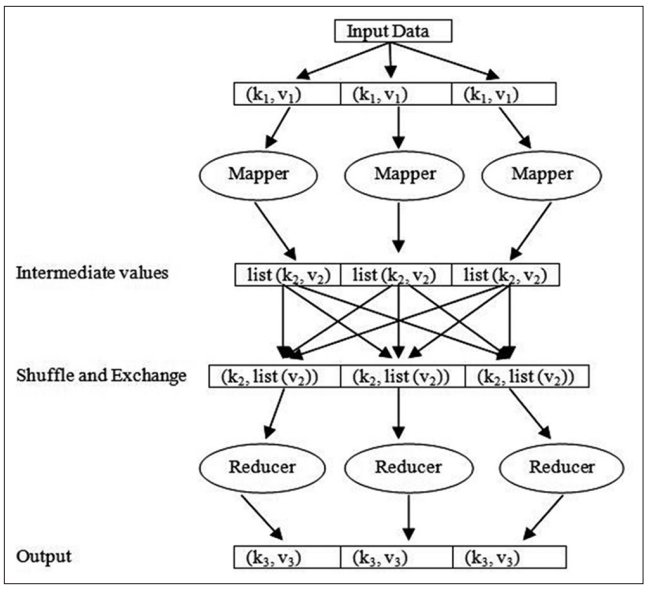

Figure 3: MapReduce model with input and output shown for each phase

minimum support (threshold) value becomes lower, then an exponentially large number of itemsets are generated. Hence, pruning unnecessary patterns can be done effectively. The main aim is to minimize the process of finding patterns which are supposed to be efficient, scalable, and detect the important patterns.

Disadvantages of other popular frequent mining algorithms

1. Apriori reduces the size of the candidate set, but it scans the database many times. The performance is affected by scanning the database multiple times. Apriori is efficient only for market basket analysis.
2. Frequent pattern (FP)-growth overcomes the limitations of Apriori. It scans only the database only twice. The performance is improved when compared to Apriori. FP suffers from memory requirement problem.

```
function RElim (a: array of transaction lists,   (* conditional database to process *)
                p: set of items,                 (* prefix of the conditional database a *)
                s_min: int) : int                (* minimum support of an item set *)
var i, k: int;                                   (* buffer for the current item *)
    s: int;                                      (* support of the current item *)
    n: int;                                      (* number of found frequent item sets *)
    b: array of transaction lists;               (* conditional database for current item *)
    t, u: transaction list element;              (* to traverse the transaction lists *)
begin                                            (* — recursive elimination — *)
  n := 0;                                        (* initialize the number of found item sets *)
  while a is not empty do                        (* while conditional database is not empty *)
    i := length(a) − 1; s := a[i].wgt;           (* get the next item to process *)
    if s ≥ s_min then                            (* if the current item is frequent: *)
      p := p ∪ {item(i)};                        (* extend the prefix item set and *)
      report p with support s;                   (* report the found frequent item set *)
      b := array [0..i−1] of transaction lists;  (* create an empty list array *)
      t := a[i].head;                            (* get the list associated with the item *)
      while t ≠ nil do                           (* while not at the end of the list *)
        u := copy of t; t := t.succ;             (* copy the transaction list element, *)
        k := u.items[0];                         (* go to the next list element, and *)
        remove k from u.items;                   (* remove the leading item from the copy *)
        if u.items is not empty                  (* add the copy to the conditional database *)
        then u.succ = b[k].head; b[k].head = u; end;
        b[k].wgt := b[k].wgt +u.wgt;             (* sum the transaction weight *)
      end;                                       (* in the list weight/transaction counter *)
      n := n + 1 + RElim(b, p, s_min);           (* process the created database recursively *)
      p := p − {item(i)};                        (* and sum the found frequent item sets, *)
    end;                                         (* then restore the original item set prefix *)
    t := a[i].head;                              (* get the list associated with the item *)
    while t ≠ nil do                             (* while not at the end of the list *)
      u := t; t := t.succ;                       (* note the current list element, *)
      k := u.items[0];                           (* go to the next list element, and *)
      remove k from u.items;                     (* remove the leading item from current *)
      if u.items is not empty                    (* reassign the noted list element *)
      then u.succ = a[k].head; a[k].head = u; end;
      a[k].wgt := a[k].wgt +u.wgt;               (* sum the transaction weight *)
    end;                                         (* in the list weight/transaction counter *)
    remove a[i] from a;                          (* remove the processed list *)
  end;
  return n;                                      (* return the number of frequent item sets *)
end;   (* function RElim() *)
```

Figure 4: Relim algorithm

Traditional RElim

This algorithm was proposed by Christian Borgelt, in 2005.[40] RElim stands for "Recursive Elimination." RElim tries to find all frequent itemsets with a given prefix by recursively renewing support at the same time. The approach used in the RElim is "pattern-growth method." RElim is based on H-mine and FP-growth algorithms. RElim uses linked list as a data structure. RElim uses horizontal layout. RElim algorithm is free from candidate generation.

RElim algorithm [Figure 4][40]

The preprocessing of RElim is demonstrated in Figure 5, which shows an example transaction database on the left. The frequencies of the items in this database, sorted ascending, are shown in Figure 5 in the middle. If we are given a user-specified minimal support of three transactions, items f and g can be discarded. After doing so and sorting the items in each transaction ascending with respect to their frequencies, we obtain the reduced database shown in Figure 5 on the right.

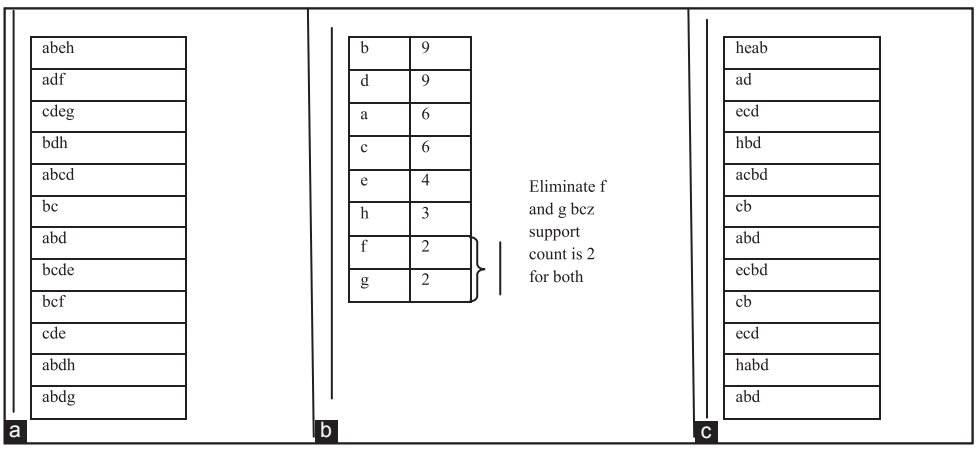

Figure 5: Transaction database (left), item frequencies (middle), and reduced transaction database with items in transactions sorted ascending with respect to their frequency. (a) Initial database. (b) Calculating item frequencies (support = 3). (c) Sorted with respect to frequencies

Recursive procedure of RElim

The lists are grouped according to their leading item. The leading item of each transaction has been removed from all transactions, as it is implicitly represented by which list a transaction is contained. Each transaction list contains in its header a counter for the number of transactions. For the rightmost list, this count states the support of the associated item in the represented dataset and the left list represented the list of items followed by the leading item [Figure 6].

Reassignments are made to lists that lie to the right of the currently processed one. For each list element, the leading item of its transaction is retrieved and used as an index into the list array; then, the element is added at the head of the corresponding list. Copy of the list element is inserted in the same way into an initially second array of transaction lists. In this particular example, remove h and makes its support as zero and the list assigned to the remaining lists, i.e. h contains three lists of itemsets (1) (e,a,b), (2) (b,d), and (3) (a,b,d). The first list (eab) is added to list bcz the leading element is e (ab is added to list e) and the support is incremented by one bcz only one is added from the previous list. For the second list (b,d), the second list item d is added to list b and increment support of b by 1. For the third list (abd), the leading item is a, the remaining list (bd) is added to the a, and increment support of a by one. The procedure is shown right as prefix h [Figure 7].

Next, eliminate e by reassigning the list items of e to the corresponding list. In the above example, e has three sublists, namely (bd), (cbd), and (ab). Reassign the lists one by one, i.e. first d to b, bd to c, and b to a and increment the support of b, c, and a by one [Figure 8].

The next step is eliminate a by reassigning the list items of to the corresponding list. In the above example, a has four sublists, namely (b), (cbd), (bd), and (b). The first list b, as the first list contains only one item directly increment support of b, for (cbd) sublist add item (bd) to c and increment support of c, for list (bd) add item d to b and increment support of b and the last list

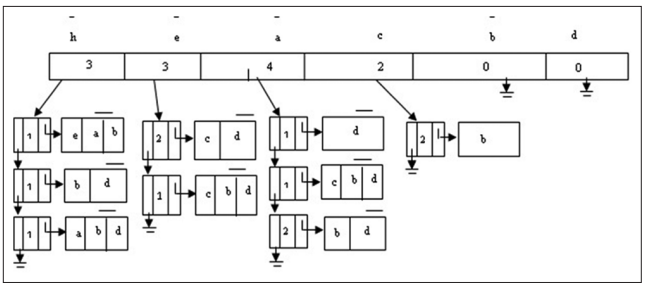

Figure 6: Initial database in RElim

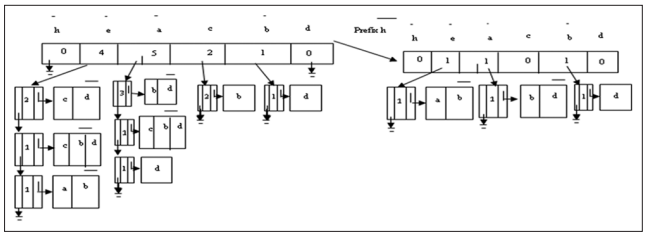

Figure 7: Eliminate h and prefix for h is also shown

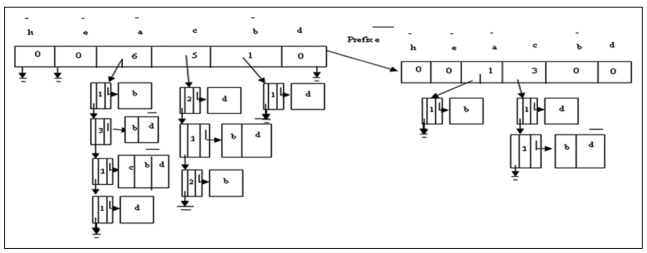

Figure 8: Eliminate e

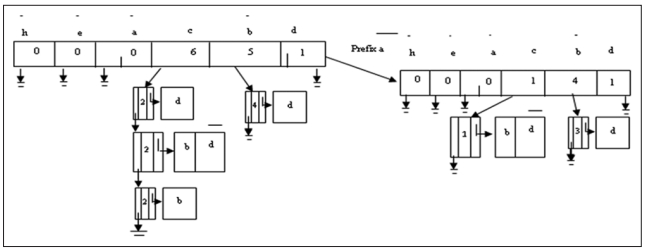

Figure 9: Eliminate a

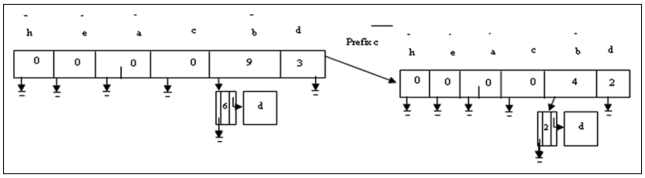

Figure 10: Eliminate c

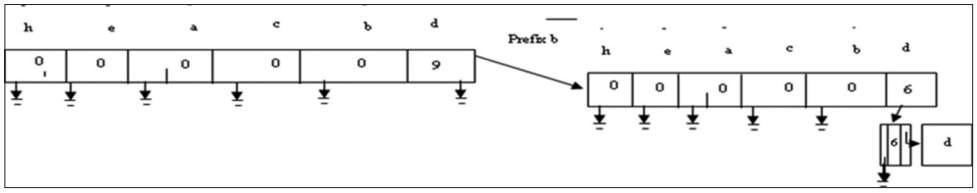

Figure 11: Eliminate b

contains only one item d so directly increment support of d [Figure 9].

The above discussed same recursive procedure is applied to eliminate c [Figure 10].

All transaction lists have been processed and the lists have become empty. The list for the last element (referring to item d) is always empty because there are no items left that could be in a transaction and thus all transactions are represented in the counter [Figure 11].

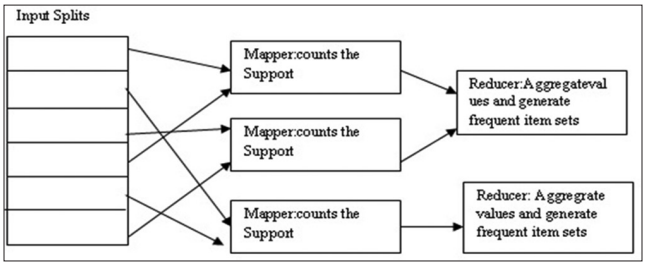

Figure 12: Design of the proposed system

RELIM ALGORITHM ON HADOOP MAPREDUCE

To implement an algorithm on MapReduce framework, the main tasks are to design two independent map and reduce functions for the algorithm and to convert the datasets in the form of (key and value) pairs. In MapReduce programming, all the mapper and reducer on different machines execute in parallel fashion, but the final result is obtained only after the completion of reducer. If algorithm is recursive, then we have to execute multiple phases of MapReduce to get the final result [Figure 12].[33]

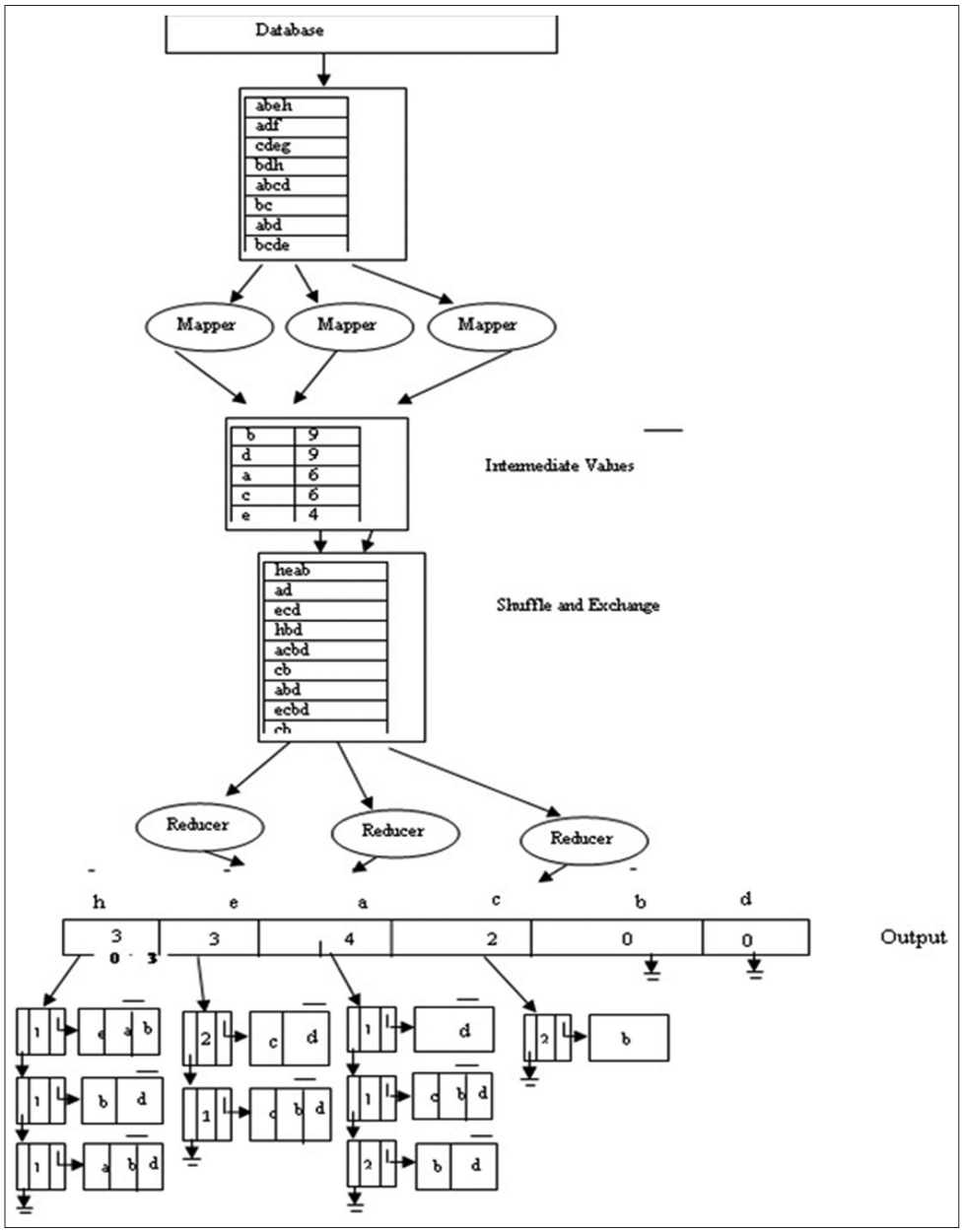

Figure 13: Generating frequent itemsets by RElim on MapReduce

Table 2: Algorithm: Mapper, combiner, and reducer

Mapper (key, value)	Combiner (key, value)	Reducer (key, value)
//key: ID	//key: items	//key: list (itemset)
//value: itemsets in transaction Ti	//value: count	//value: count
for each transaction Ti assigned to Mapper do	for each itemset do	for each list do
for each itemset in Ck do	for each item in list (itemset) of corresponding itemset do	for each itemset in leading item of its transaction is retrieved and used as an index into the list array
if itemset ∈ Ti	compare items with their frequencies and min support count	
output (itemset, count);	end for	end for
end if	sort (itemset);	Increment count if list is repeated
end for	end for	output (count, list (itemset);
end for		end for

Traditional RElim to MapReduce

HDFS breaks the transactional database into blocks and distributes to all mappers running on machines. Each transaction is converted to (key and value) pairs where key is the ID and value is the list of items. Mapper reads one transaction at time and output (key' and value') pairs where key' is each item in transaction and value is support or count. The combiner combines the pairs with same key' and makes the local sum of the values for each key. The output pairs of all combiners are shuffled and exchanged to make the list of values associated with same key and give output in sorted order of items for each transaction by removing the items that have support less than user support. Reducers take these item lists and output the linked list of items separated by their leading items. Final frequent itemsets are obtained by merging the output of all reducers [Figure 13].

Table 2 summarizes the algorithms corresponding to mapper, combiner, and reducer for RElim algorithm.

CONCLUSION

MapReduce is very beneficial for parallel processing of big data on large cluster of commodity computers. In this paper, there is focus on the RElim algorithm on MapReduce framework. The MapReduce computing model is similar to the computation of frequent itemsets in RElim algorithm. RElim performs excellently on sparse datasets. For artificial datasets, RElim has the best performance when compared to other frequent itemset mining algorithms. For BMS-webView-1, RElim performance is almost equivalent to FP growth.

AQ1

REFERENCES

1. Hipp J, Ulrich G, Gholamreza N. Algorithms for association rule mining-a general survey and comparision. ACM 2000;2:58-64.
2. Han J, Kamber M. Data Mining: Concepts and Techniques. San Francisco: Morgan Kaufmann Publishers; 2006.
3. Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In; Proceedings Twentieth International Conference on Very Large Databases; 1994. p. 487-99.
4. Han J, Pei J, Yin Y. Mining frequent pattern s without candidate generation. In: ACM SIGMOD International Conference on Management of Data. Vol. 29. New York: ACM; 2000. p. 1-12.
5. Zaki M, Parthasarathy S, Ogihara M, Li W. New algorithms for fast discovery of association rules. In: Proceedings 3rd Int'l Conferences Knowledge Discovery and Data Mining. Menlo Park: AAAI Press; 1997. p. 283-6.
6. Oruganti S, Ding Q, Tabrizi N. Exploring HAD OOP as a platform for distributed association rule mining. In: FUTURE COMPUTING 2013 The Fifth International Conference on Future Computational Technologies and Applications; 2013. p. 62-7.
7. Agrawal R, Shafer JC. Parallel mining of association rules. In: IEEE Transactions on Knowledge and Data Engineering. Vol. 8; 1996. p. 962-9.
8. Zaki MJ. Parallel and distributed association mining: A survey. Concurr IEEE 1999;7:14-25.
9. Zaki MJ. Parallel and distributed data mining: An introduction. In: LNAI 1759 Large-Scale Parallel Data Mining. Heidelberg: Springer; 2000. p. 1-23.
10. Bhaduri K, Das K, Liu K, Kargupta H, Ryan J. Distributed data mining bibliography; 2008:1-46.
11. Fiolet V, Olejnik R, Lefait G, Toursel B. Optimal grid exploitation algorithms for data mining. In: Proceedings 5th IEEE International Symposium on Parallel and Distributed Computing; 2006. p. 246-52.
12. Luo C, Pereira AL, Chung SM. Distributed mining of maximal frequent itemsets on a data grid system. J Supercomput 2006;37:71-90.
13. Aouad LM, Le-Khac NA, Kechadi TM. Distributed frequent item sets mining in heterogeneous platforms. J Eng Comput Archit 2007;1:1-12.
14. Yang CT, Shih WC, Tseng SS. A heuristic data distribution scheme for data mining applications on grid environments in fuzzy systems. FUZZ-IEEE (IEEE World Congress on Computational Intelligence). Hong Kong: IEEE Press; 2008. p. 1-6.

15. Tlili R, Slimani Y. A novel data partitioning approach for association rule mining on grids. Int J Grid Distrib Comput 2012;5:1-20.
16. Tlili R, Slimani Y. Executing association rule mining algorithms under a grid computing environment. In: Proceedings Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD '11), ACM; 2011. p. 53-61.
17. Tlili R, Slimani Y. Mining association rules on grid platforms. In: LNCS Euro-Par 2011 Workshops, Springer; 7155. p. 201-10.
18. Tlili R, Slimani Y. Dynamic load balancing of large-scale distributed association rule mining. In: Proceedings IEEE International Conference on Computer Application and Industrial Electronics (ICCAIE 2011); 2011. p. 553-8.
19. Lin MY, Lee PY, Hsueh SC. Apriori Based Frequent Itemset Mining Algorithms on Map Reduce. In: Proceedings 6th International Conference on Ubiquitous Information Management and Communication, ACM; 2012. p. 1-8.
20. Moens S, Aksehirli E, Goethals B. Frequent Item Setmining for Big Data. In: Proceedings IEEE International Conference on Big Data; 2013. p. 1-8.
21. Dean J, Ghemawat S. MapReduce: Simplified da ta processing on large clusters. ACM Commun 2008;51:107-13.
22. Chen Y, Archanan G, Rean G, Randy K. The Case for Evaluating Map Reduce Performance using Workload Suites. IEEE 19 International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems; 2011. p. 390-9.
23. Apache Hadoop. Available from: http://www.hadoop.apache.org.
24. Tan PN, Steinbach M, Kumar V. Chapter 5-Association Analysis: Basic Concepts and Algorithms, Introduction to Data Mining. Boston: Addison-Wesley; 2005. p. 357-449.
25. Park JS, Ming-Syan MS, Yu PS. Using a hash-B ased method with transaction trimming for mining association rules. IEEE Trans Knowl Data Eng 1997;9:813-25.
26. Park JS, Ming-Syan MS, Yu PS. An effective H ash-based algorithm for mining association rules. SIGMOD ACM 1995;24:175-86.
27. Zaki MS, Parthasarathy S, Li W, Ogihara M. Evaluation of Sampling for Data Mining of Association Rules. In: Proceedings IEEE 7th International Workshop on Research Issues in Data Engineering; 1997. p. 42-50.
28. Savasere A, Omiecinski E, Navathe S. An Efficient Algorithm for Mining Association Rules in Large Databases. In: Proceedings 21st VLDB Conference, Switzerland; 1995. p. 432-44.
29. Brin S, Motwani R, Ullman JD, Tsur S. Dynamic item set counting and implication rules for market basket data. ACM SIGMOD Record 1997;26:255-64.
30. Yahoo! Hadoop Tutorial. Available from: http://www.developer.yahoo.com/hadoop/tutorial/index.html.
31. Ghemawat S, Gobioff H, Leung S. The google file system. ACM SIGOPS Oper Syst Rev 2003;37:29-43.
32. Lee KH, Lee YJ, Choi H, Chung YD, Moon B. Parallel data processing with mapreduce: A survey. ACM SIGMOD Record 2011;40:11-20.
33. Kovacs F, Illes J. Frequent item set mining on Hadoop. In: Proceedings IEEE 9th International Conference on Computational Cybernetics (ICCC); 2013. p. 241-5.
34. Li N, Zeng L, He Q, Shi Z. Parallel Implementation of RE lim Algorithm based on MapReduce. In: Proceedings 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel and Distributed Computing, IEEE; 2012. p. 236-41.
35. Li J, Roy P, Khan SU, Wang L, Bai Y. Data Mining Using Clouds: An Experimental Implementation of Apriority over Map Reduce. IEEE, Processing 12th International Conference on Scalable Computing and Communications; 2012. p. 1-8.
36. Yang XY, Liu Z, Fu Y. Map reduce as a programming model for association rules algorithm on hadoop. Proc Int Conf Inform Sci Interact Sci (ICIS) 2010;99:23-5.
37. Li L, Zhang M. The Strategy of Mining Association Rule Based on Cloud Computing. In: Proceedings IEEE International Conference on Business Computing and Global Informatization (BCGIN); 2011. p. 29-31.
38. Margaret HD, Xiao Y. A Survey of Association Rules, Technical Report, Southern Methodist University, Department of Computer Science, Technical Report TR 00-CSE-8; 2008. p. 1-65.
39. Yaha O, Hegazy O, Ezat E. An efficient implementation of RElim algorithm based on hadoop-mapreduce model. Int J Rev Computing 2012;12:59-67.
40. Borgelt C. Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination Opensource Data Mining Workshop, OSDM, II, ACM Press; 2005. p. 66-70.
41. Agarwal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. In: Proceeding SIGMOD 93 Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data; 1993. p. 207-16.
42. Available from: https://www.techopedia.com/definition/30306/association-rule-mining.
43. Dean J, Ghemawat S. MapReduce: Simplified da ta processing on large clusters. ACM Commun 2008;51:107-13.
44. Available from: https://www.en.wikipedia.org/wiki/MapReduce.

Author Queries???

AQ1:Please note reference number 19, 20, 22, 24-29, 34-39 are not cited in the text part. Kindly check and advise.

AQ2:Kindly provide last accessed details