Data Mining Research Paper
Frequent Pattern Analysis and Association rule: Apriori Algorithm on Hadoop MapReduce Framework

Joseph D. Yun
Georgia State University
Undergraduate, Computer Science
jyun10@student.gsu.edu

ABSTRACT
Identifying associations and frequently appearing subsets from a dataset is one of the most important field of study in both Data Mining and Big Data. Often times dealing with such large dataset, mining association rules from the dataset can take a very long time. Consequently one of the solution proposed was Apriori algorithm [1]. Originate from latin root, the term refers to denoting reasoning or knowledge that proceeds from theoretical deduction rather than from observation or experience. Therefore the Apriori Algorithm deduces the running time by sequentially scanning the dataset and deducing the size of our target dataset. Nonetheless, this very solution can also take long periods of time to run in the cases of huge datasets that oftentimes exists in today's world. This paper proposes the possibility of improving the efficiency of Apriori algorithm by implementing MapReduce framework.

Categories and Subject Descriptors
D.3.2 [**Java**]: MapReduce (Apache Hadoop): Mapper, Reducer, Combiner.

K.3.2 [**Computer and Information Science Education**]: computer science education, curriculum.

General Terms
Algorithms, Measurement, Performance, Design, Experimentation, Languages, Verification.

Keywords
Apriori algorithm, Association Rules, Data Mining, MapReduce, Machine Learning, MapReduce.

1. INTRODUCTION
Data mining is the extraction of interesting patterns or knowledge from given data, whether if that is non-trivial, implicit, previously unknown, or potentially useful. This very appealing idea has attracted many attention throughout large spectrum of organizations, such as research, commercial institutions, businesses, government. One of the important subject in data mining is pattern recognition and association rule mining [2]. It refers to the task of finding all subsets of the superset that frequently occur and identifying the relationship or association among the subsets by using two main steps: identifying the frequent itemsets and generating association rules [2].

The apriori algorithm [1] is used to find frequent sets from the given dataset. The algorithm operates by using *downward closure* property of frequent patterns which traverse the dataset based on initial frequency and deduce our growing sets sequentially until it satisfies our minimum requirements. However this very methods of sequentially iterating through the dataset and generating candidate itemsets exponential. Therefore when the dataset size is huge, both memory use and the computational cost can be extremely expensive.

For example,

Let, N = number of sets or *number of scans* = 100.
Then, *k, number of candidate subset,* is exponential: $2^{100} - 1$, which equivalates to
1.27×10^{30}= 1,267,650,600,228,229,401,496,703,205,375

In addition, the resources such as the memory and CPU resource are very limited to the single processor. This additional limitation furthermore puts the burden onto the computational cost of the algorithm. Eventually it becomes nearly impossible to process the dataset on a single sequential machines. One of the solution for such problem is parallel and distributed computing. If an efficient and scalable parallel and distributed algorithm can be implemented, parallel and distributed computing [3] can offer a solution for the computing limitation from problem above,.

The implementation can be achieved by using Hadoop-MapReduce framework which is writing applications that has the ability to process vast amount of data in-parallel on a large computer clusters in reliable, fault-tolerance manner [4]. More details about Hadoop-MapReduce is explained in future sections.

Finally, the rest of this paper is organized by:

Section 2: Background of MapReduce, Hadoop framework, Association Rule, and Apriori Algorithm, Dataset

Section 3: Previous work, Related work

Section 4: Explanation of the dataset

Section 5: Apriori algorithm in Parallel and distributed computing implementation.

Section 5.1: Conclusion and Performance Evaluation,

2. BACKGROUND

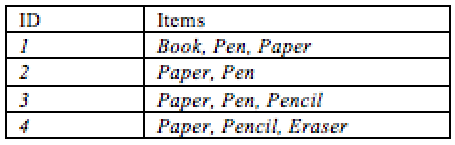
2.1 Association

Association rules [2] are used to find the relationship between elements in a large dataset. It can be used to find the normality and relationship between two or more elements. Frequent pattern matching can be utilized in many different ways. The example of primary use are data analysis, marketing, web analysis, and gene analysis.

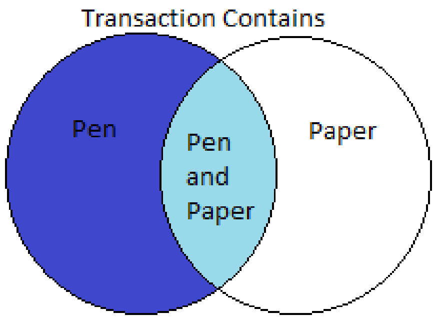
Furthermore, many businesses such as Amazon collects user data and analyze the collected data using various methods. One of the prominent technique is known as association rules which oftentimes is used to create a profitable business model and business analysis.

For example we have the following dataset.

Table 1.



ID	Items
1	*Book, Pen, Paper*
2	*Paper, Pen*
3	*Paper, Pen, Pencil*
4	*Paper, Pencil, Eraser*



In *Table 1*, first column contains the transaction ID and second column contains item in each transaction. Looking at our given dataset, it is fairly easy to discover that there is relationship between pen and paper.

Pen → Paper

Retailers can use this rule to suggest paper item to the prospect customer who are looking to buy pen. There are two major measurement of an association rules - support and confident. Support measure the frequency of data in a dataset. Confidence measure the percentage of item A also contain item B.

A→ *B*

Support: Number of transaction contain A∪ *B*
Confidence: Possibility that transaction A also contain transaction B

Example of support and confidence from Table 1

Pen → Paper

Support: Pen and Paper is 3 and there are 5 transaction. The support is ⅗ which is 60%.

Confidence: We divide support count of pen and paper with the support count of pen. The support count for pen and paper is 3 and support count for pen is 3. The confident count of pen → paper is 3/3 which is 100%.

Support are use the measures if the rule happens randomly. We might want to exclude small support data. In the cross-market example, weak support count relations won't benefit the sales since it contains items set bought in a limited number of transaction compare to total transaction.

Confidence measures how often the supporting dataset contains the rule. Low confidence means the relationship between two set rarely happens on the other hand high confidence means relation. in this case store items, often purchased together.

2.1.1 Serial Association Rule Implementation

For each set must contain support and confidence.
For the following given example in *Table 1:*

$\{Book\}, \{Pen\}, \{Paper\},\{Book, Pen\},\{Book, Paper\},$
$\{Paper, Pen\}, \{Book, Pen, Paper\}$

That is the total of 7 subset for a transaction with 3 datasets. The following equation can be used to calculate the amount of subset in a dataset.

$$R = 2^N - 1$$

The subset exponentially increase as a number of transaction increases. This is a very extensive tasks and as result, computation time can be wasted. Although many of the rules calculated are discard when there is minimum support and minimum confidence; However, given a large dataset, the problem is still prominent.

To solve this issue, we can prefilter out rule with low support and confidence using frequent subset generation and rule generation. Transaction ID 1, transaction ID 2, and transaction ID 3 would have similar support because it contains the same item. We can keep or discard multiple transaction by just compute similar rule. With this we can preprocess the dataset and eliminate majority of low support data,

2.2 Apriori Algorithm

Apriori algorithm is used for frequent item set mining and association rule learning over transactional datasets. The algorithm was proposed by Agrawal and Srikant in 1994 [5]. Algorithm Implementation and Candidate Generation is explained in next section.

2.2.1 Implementation

Figure 1 gives the Apriori algorithm. The rst pass of the algorithm simply counts item occurrences to determine the large 1-itemsets. A subsequent pass, say pass *k*, consists of two phases. First, the large itemsets L_{k1} found in the $(k_1)th$ pass are used to generate the candidate itemsets C_k , using the apriori-gen function described in Section 2.2.1. Next, the database is scanned and the support of candidates in C_k is counted. For fast counting, we need to efficiently determine the candidates in $C_k$that are contained in a given transaction *t*. Figure 1 illustrates the steps.

Figure 1: Apriori Algorithm

```
1). $C_k$: Candidate itemset of size k
2). $L_k$ : frequent itemset of size k
3). $L_1$ = {frequent items};
4). for (k = 1; $L_k$ != ∅; k++) do begin
5).         $C_{k+1}$ = candidates generated from $L_k$;
6).         for each transaction t in database do
7).                 increment the count of all candidates in $C_{k+1}$
                    that are contained in t
8).         $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
9).         end
10). return È$_k$ $L_k$;
```

2.2.2 Apriori Candidate Generation

The *apriori-gen* function takes an argument L_{k-1}, the set of *all large (k - 1)* - itemsets. It returns a superset of the set of all large *k*-itemsets. The function works as follows.

Step 1 : **Self Joining L_k**

Insert into C_k
Select p.item$_1$, p.item$_2$, …, p.item$_{k-1}$, q.item$_{k-1}$ **From** L_{k-1} p, L_{k-1} q
Where p.item$_1$=q.item$_1$, …, p.item$_{k-2}$=q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$

Step 2 : **Pruning**

```
forall itemsets c in $C_k$ do
        forall (k-1)-subsets s of c do
                if (s is not in $L_{k-1}$) then delete c from $C_k$
```

2.2.3 Apriori Example

Dataset

Transaction Id	*Items*
10	*A, C, D*
20	*B, C, E*
30	*A, B, C, E*
40	*B, E*

* $Supp_{min} = 2$

Scan 1*: C_1& Remove infrequent itemset (D)*

Itemset	*Support*
{A}	*2*
{B}	*3*
{C}	*3*
{D}	*1*
{E}	*3*

Scan 1*: Output; L_1*

Itemset	*Support*
{A}	*2*
{B}	*3*
{C}	*3*
{E}	*3*

Scan 2*: Input (Based on Scan 1 deduction) & Remove infrequent set;*

C_2

Itemset	*Support*
{A, B}	*1*
{A, C}	*2*
{A, E}	*1*
{B,C}	*2*
{B, E}	*3*
{C, E}	*2*

Scan 2*: Output; L_2*

Itemset	*Support*
{A, C}	*2*
{B, C}	*2*
{B, E}	*3*
{C, E}	*2*

Scan 3*: Input (Based on Scan 2 deduction);C_3*

Itemset	*Support*
{B, C, E}	*2*

Scan 3*: Final output; L_3*
Algorithm Terminates

Itemset	*Support*
{B, C, E}	*2*

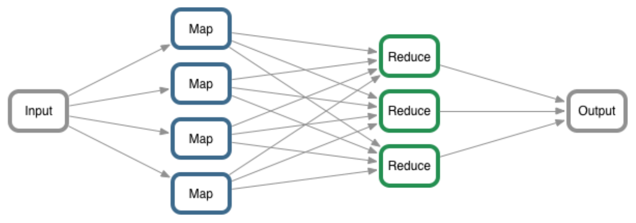
2.3 MapReduce

One of the prominent and well known parallel and distributed (cluster) computing, MapReduce Model is created by Google in 2004. It is a programming model based on C++ language which provides associated implementation for processing and generating large data sets in a massively parallel and distributed manner [6]. MapReduce program is mainly composed of *Map* routine that performs mapping procedure that outputs <*Key, Value*> pair based on given dataset, and *Reduce* routine that performs merging of these <*Key, Value*> pairs with same intermediate key. There are also intermediate routine such as *Combiner* that reduces processing time by combining intermediate <*Key, Value*> pairs, provided in map routine, which otherwise *Reducer* would have to process.

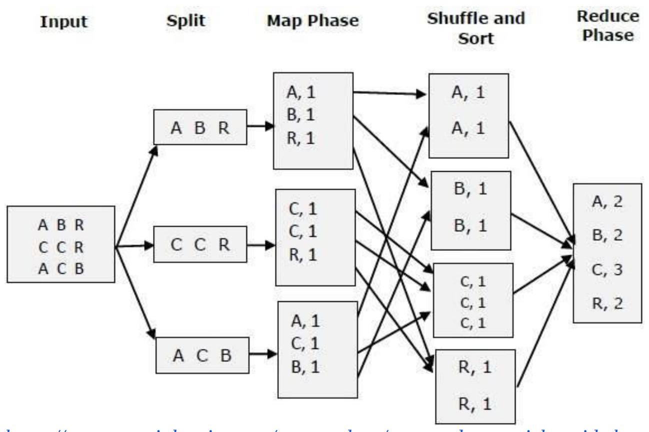
Due to the simplicity of this modeling, the task of the developers is diminished to implementing map function that processes a <*Key, Value*> pairs, and the reduce function that merges all intermediate values associated with the same intermediate key. Many data mining areas such as: association rule, clustering and classification algorithms can and have been implemented based on this MapReduce model.

Figure 2.3.1 provides an example of the data flow.

2.3.1 MapReduce Models



http://www.kdnuggets.com/2016/05/hadoop-key-terms-explained.html [7]



https://www.tutorialspoint.com/map_reduce/map_reduce_quick_guide.htm [8]

2.4 Hadoop Framework

Hadoop (*Apache Hadoop*) is an open-source software framework is used for distributed storage and processing of big data sets using the MapReduce programming model. It is composed of computer clusters built from commodity computing hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework [9].

The core of *Hadoop* is consisted of a distributed storage part, known as *Hadoop Distributed File System (HDFS)*, and a processing part which is a MapReduce implementation. *Hadoop* distributes files into large blocks and sends them across nodes in a cluster system. Then transfers packaged code into nodes to process the data in parallel and distributed manner. This approach takes advantage of data locality [10] - nodes manipulating the data they have access to - to allow the dataset to be processed faster and more efficiently than it would be in a conventional *multi-core* computer architecture that relies on a parallel file system where computation and tat are distributed via high-speed networking.

The term *Hadoop* has become not only the framework itself, but also an *environment* [11], or collection of additional software packages that can be installed on top of or alongside with *Hadoop*. Such as *Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache Spark, Apache ZooKeeper, Cloudera Impala, Apache Flue, Apache Sqoop, Apache Oozie, Apache Storm.*

The *Hadoop* framework is mostly written in *Java*, along with native code in *C* and command line utilities written as *shell scripts*.

3. RELATED WORKS

3.1 Fast Algorithm for Mining Association Rule [14]

In this proposed paper, Rakesh and Ramakrish proposes improved algorithm for reducing running time of original Apriori algorithm. The proposed idea argues the property of Apriori algorithm in which:

If the set a => (1 - a) does not hold, neither does ~a => (l - ~a) for any ~$a \subset a$.

Therefore, in order for the rule (1 - c) => c to hold, all rules of the form (1- ~c) => ~c must also hold, where ~c *is* a non-empty subset of c.

For example, if the rule *AB* => *CD* holds, then the rules *ABC* => *D* and *ABC* => *C* must also hold.

Finally, with the understanding of this property of original algorithm, paper proposes the idea in which:

```
// Faster Algorithm
1) forall large k-itemsets lk, k  2 do begin
2)      H1 = {consequents of rules derived from $l_k$ with one
        item in the consequent};
3)      call ap-genrules($l_k$ ,$H_1$);
4) end

procedure ap-genrules:
($l_k$: large k-itemset, $H_m$: set of m-item consequents)
        if (k > m + 1) then begin
           $H_{m=1}$ = apriori-gen(Hm);
           forall $h_{m=1} \in H_{m=1}$ do begin
              conf = support ($l_k$) / support ($l_k$ - $h_{m=1}$);
              if (conf ≥ minconf) then
                  output the rule ($l_k$ - $h_{m=1}$ ) => $h_{m=1}$ with
                  confidence = conf and support = support ($l_k$);
              else
                  delete $h_{m=1}$ from $H_{m=1}$;
              end
              call ap-genrules($l_k$ - $h_{m=1}$);
        end
```

3.2 A Fast Distributed Algorithm for Mining Association Rule [15]

3.2.1 FDM-LP: FDM with Local Pruning

This paper describes alternative to known *Association Rule* mining techniques by using distributed or parallel algorithm.

The first proposed method is the following:

Iteratively execute the following program fragment (for the k-th iteration) distributively at each site S_i. The algorithm terminates when either L_k= ∅or the set of candidate sets $CG_{(k)}$= ∅

```
(1)  if $k = 1$ then
(2)      $T_{i(1)} = get\_local\_count(DB_i, \emptyset, 1)$
(3)  else {
(4)      $CG_{(k)} = \cup_{i=1}^{n} CG_{i(k)}$
             $= \cup_{i=1}^{n} Apriori\_gen(GL_{i(k-1)})$;
(5)      $T_{i(k)} = get\_local\_count(DB_i, CG_{(k)}, i)$ ; }
(6)  for_all $X \in T_{i(k)}$ do
(7)      if $X.sup_i \geq s \times D_i$ then
(8)          for $j = 1$ to $n$ do
(9)              if $polling\_site(X) = S_j$ then
                     insert $\langle X, X.sup_i \rangle$ into $LL_{i,j(k)}$;
(10) for $j = 1, \ldots, n$ do send $LL_{i,j(k)}$ to site $S_j$;
(11) for $j = 1, \ldots, n$ do {
(12)     receive $LL_{j,i(k)}$;
(13)     for_all $X \in LL_{j,i(k)}$ do {
(14)         if $X \notin LP_{i(k)}$ then
                 insert $X$ into $LP_{i(k)}$;
(15)         update $X.large\_sites$; } }
(16) for_all $X \in LP_{i(k)}$ do
(17)         $send\_polling\_request(X)$;
(18) $reply\_polling\_request(T_{i(k)})$;
(19) for_all $X \in LP_{i(k)}$ do {
(20)     receive $X.sup_j$ from the sites $S_j$,
         where $S_j \notin X.large\_sites$;
(21)     $X.sup = \sum_{i=1}^{n} X.sup_i$;
(22)     if $X.sup \geq s \times D$ then
             insert $X$ into $G_{i(k)}$; }
(23) broadcast $G_{i(k)}$;
(24) receive $G_{j(k)}$ from all other sites $S_j$, $(j \neq i)$;
(25) $L_{(k)} = \cup_{i=1}^{n} G_{i(k)}$.
(26) divide $L_{(k)}$ into $GL_{i(k)}$, $(i = 1, \ldots, n)$;
(27) return $L_{(k)}$.
```

3.2.2 FDM-LP: FDM with Local Pruning

The second proposed method is the following:

Program fragment of FDM-LUP is obtained from FDM-LP by inserting the following condition in line 7.1 after line 7 of the first algorithm.

```
(7.1)    if $g\_upper\_bound(X) \geq s \times D$ then
```

The function *g_upper_bound* computes an upper bound for a candidate set *X*. In other words, *g_upper_bound* returns an upper bound of *X* as the sum:

$$X.sup_i + \sum_{j=1, j \neq i}^{n} maxsup_j(X).$$

X.sup is computed already in the local prning step, and the values of *maxsupp(X)*, can be computed from the local support counts from the (k - 1) iteration. If upper bound is smaller than the global support threshold, it is used to prune away *X*. FDM-LUP should usually have a smaller number for candidate sets for count exchange in comparison with FDM-LP.

3.2.3 FDM-LPP: FDM with Local Pruning and Polling-Site-Pruning

The algorithm method is the following:

The program fragment of FDM-LPP is obtained from algorithm 1 by replacing its line 17 with the following two lines:

```
(16.1)   if $p\_upper\_bound(X) \geq s \times D$ then
(17)         $send\_polling\_request(X)$;
```

The new step in FDM-LPP is the one for polling-site-pruning. At that stage, $S_i$is a polling site and has received requests from the other sites to perform polling. Each request contains a locally large itemset *X* and its local support count *X.supp* where $S_j$is a site from which *X* is sent to S_i. *X.large_sites* is the set of all the originating sites from which the requests for polling *X* are being sent to the polling site (line 15). For every site $S_i \in$ *X.large_sites* the local support count *X.supp_j* has been sent to S_i already. For a site,

$S_q \notin$*X.large_sites* since *X* is not locally large at S_q, its local support count X.supp_q must be smaller than the local threshold $s \times D_q$. Previously mentioned, $X.supp_q$is bounded by the value $min\ (maxsupp_q(X), s \times D_q - 1)$. Hence an upper bound of $X.supp$can be computed by the sum:

$$\sum_{j \in X.large\_sites} X.sup_j + \sum_{q=1, q \notin X.large\_sites}^{n} min(maxsup_q(X), s \times D_q - 1).$$

In FDM-LPP, $S_i$calls *p_uppoer_bound* to compute an upper bound for *X.supp* according to the above formula. This upper bound can be used to prune away *X* if it is smaller than the global support threshold.

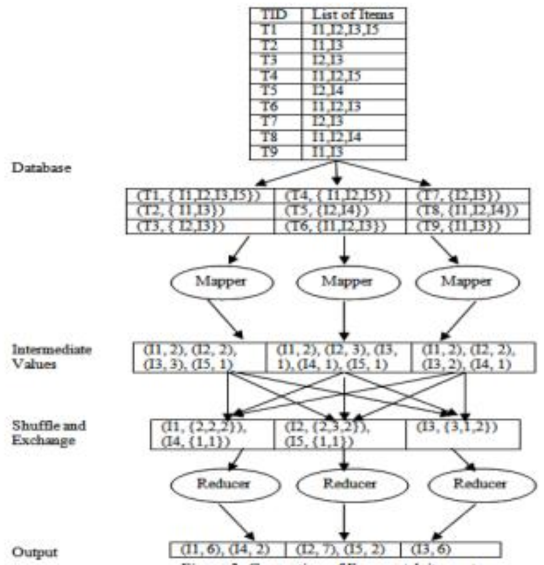
3.3 Review of Apriori Algorithm on Hadoop MapReduce [16]

The following paper introduces various of previous implementation of Apriori Algorithm on Hadoop MapReduce. Paper argues that the first step of the algorithm is to generate frequent 1-itemsets L_1 . HDFS breaks the transactional database into blocks and distribute to all mappers running on machines.

Each transaction is converted to (key, value) pairs where key is the TID and value is the list of items. Mapper reads one transaction at time and output (key2,value2) pairs where key2 is each item in transaction and value2 is 1.

The combiner combines the pairs with same key2 and makes the local sum of the values for each key2. The output pairs of all combiners are shuffled & exchanged to make the list of values associated with same key2, as (key2, list value3) pairs where key2 is item and value3 is the support count ≥ min. supp. of corresponding item.

Apriori algorithm to MapReduce Based Apriori using following diagram:


TID
List of Items
T1 I1,I2,I3,I5
T2 I1,I3
T3 I2,I3
T4 I1,I2,I5
T5 I2,I4
T6 I1,I2,I3
T7 I2,I3
T8 I1,I2,I4
T9 I1,I3
Database
(T1, { I1,I2,I3,I5}) (T4, { I1,I2,I5}) (T7, {I2,I3})
(T2, { I1,I3}) (T5, {I2,I4}) (T8, {I1,I2,I4})
(T3, { I2,I3}) (T6, {I1,I2,I3}) (T9, {I1,I3})
Mapper
Mapper
Mapper
Intermediate Values
(I1, 2), (I2, 2), (I3, 3), (I5, 1)
(I1, 2), (I2, 3), (I3, 1), (I4, 1), (I5, 1)
(I1, 2), (I2, 2), (I3, 2), (I4, 1)
Shuffle and Exchange
(I1, {2,2,2}), (I4, {1,1})
(I2, {2,3,2}), (I5, {1,1})
(I3, {3,1,2})
Reducer
Reducer
Reducer
Output
(I1, 6), (I4, 2)
(I2, 7), (I5, 2)
(I3, 6)
Figure 3: Generation of Frequent 1-itemsets.

The following table given, depicts the algorithms corresponding to mapper, combiner, and reducer for the Apriori algorithm:

Table 2: Algorithm: Mapper, Combiner and Reducer [19], [34], [37]

Mapper (key, value)	*Combiner (key, value)*	*Reducer (key, value)*

// key: TID
// value: itemsets in transaction Ti
for each transaction Ti assigned to Mapper *do*
  for each itemset in C_k *do*
    if itemset ∈ Ti
      output (itemset, 1);
    end if
  end for
end for
	// key: itemset
	// value: list (1)
	for each itemset *do*
	  for each 1 in list (1) of corresponding itemset *do*
	    itemset.local_sup + = 1;
	  end for
	  output (itemset, itemset.local_sup);
	end for
		// key: itemset
		// value: list (local_sup)
		for each itemset *do*
		  for each local_sup in list (local_sup) of corresponding itemset *do*
		    itemset.sup + = local_sup;
		  end for
		  if itemset.sup ≥ minimum support;
		  output (itemset, itemset.sup);
		end for

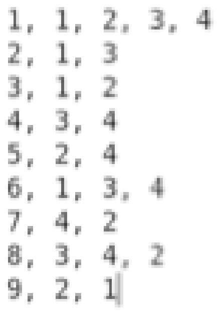
Finally, paper illustrates comparison of different proposed implementation with the table:

Algorithm Proposed by	***Map-Reduce Phase***	***Mapper (key, value)***	***Combiner (key, value)***	***Reducer (key, value)***
X. Y. Yang *et al.*, 2010 [36]	k-phase	Input: key = line no.; value = one row of data Output: key = candidate itemset; value = 1	Not used	Input: key = candidate itemset; value = list (1) Output: key = itemset; value = support
L. Li and M. Zhang, 2011 [37]	1-phase	Input: key = TID; value = transaction Output: key = itemset; value = 1	Input: key = itemset; value = list (1) Output: key = itemset; value = local support	Input: key = itemset; value = list (local support) Output: key = itemset; value = support
N. Li *et al.*, 2012 [34]	k-phase	Input: key = offset in byte of record; value = string of the content of record Output: key = candidate itemset; value = 1	Not used	Input: key = candidate itemset; value = list (1) Output: key = candidate itemset; value = support
J. Li *et al.*, 2012 [35]	k-phase	Input: key = line no.; value = one line of data Output: key = itemset; value = 1	Input: key = itemset; value = list (1) Output: key = itemset; value = local support	Input: key = itemset; value = list (local support) Output: key = itemset; value = support
M-Y Lin *et al.*, 2012 [19]	k-phase	Input: key = TID; value = itemset in transaction with TID Output: key = itemset; value = 1	Input: key = itemset; value = list (1) Output: key = itemset; value = local support	Input: key = itemset; value = list (local support) Output: key = itemset; value = support
S. Oruganti *et al.*, 2013 [6]	k-phase	Input: key and value are not specified Output: key = candidate itemset; value = 1	Not used	Input: key = candidate itemset; value = list (1) Output: key = candidate itemset; value = support
F. Kovacs and J. Illes, 2013 [33]	k-phase	Input: key = TID; value = itemset in transaction with TID Output: key = itemset; value = local support	Not used	Input: key = itemset; value = list (local support) Output: key = itemset; value = support

4. DATASET

Experimental dataset is in the form of Comma-Separated Values or cvs for short. Each row contains the transaction ID, and items purchase. One transaction can span into multiple row based on how many items are on that transaction. The dataset contains up to 75,000 records; the dataset is large enough to apply our algorithm and to identify associations between these sets. To implement apriori algorithm in parallel, Java function will able to parse the content of the csv file into Java Mapper class. Hadoop Mapper function maps intermediate in multiple clusters thereby reducing the running time of extracting data process. We will follow the same format as Extended bakery dataset provided by trac integrated SCM & project management

https://wiki.csc.calpoly.edu/datasets/wiki/apriori [12].



```
1, 1, 2, 3, 4
2, 1, 3
3, 1, 2
4, 3, 4
5, 2, 4
6, 1, 3, 4
7, 4, 2
8, 3, 4, 2
9, 2, 1
```

test.csv - Testing dataset with nine transactions and four items

First column contains receipt ID similar to unique key in SQL database and following by item id purchase on that receipt.

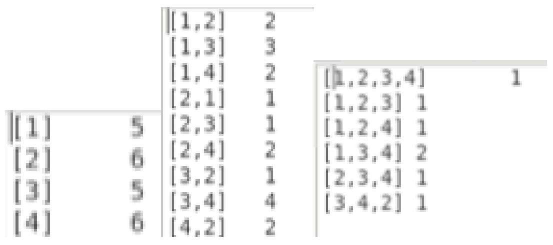
5. IMPLEMENTATION

5.1 Itemset Mapper and Reducer

Our goal for this mapper and reducer is to create an intermediate dataset. It will create output file contains all single itemset of receipt transactions and support count of each itemset. This step will not randomly select a random chunk of input file where this step should take place before this step. The mapper will input the entire Itemset from the file.

The mapper will parse the input file and call get itemSet() method which will get all the set and it subset if applicable such that transaction with item 1, 2, 3 will result in mapping of itemset {1}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}.

The Reducer will aggregate the result from the mapper and output the support count into an intermediate folder. Each reducer have a set of key:



```
[1]    5
[2]    6
[3]    5
[4]    6
```

```
[1,2]    2
[1,3]    3
[1,4]    2
[2,1]    1
[2,3]    1
[2,4]    2
[3,2]    1
[3,4]    4
[4,2]    2
```

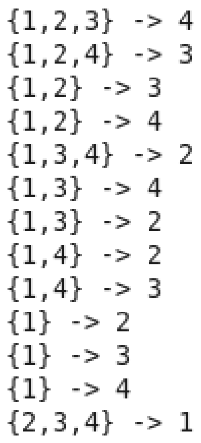
```
[1,2,3,4]    1
[1,2,3] 1
[1,2,4] 1
[1,3,4] 2
[2,3,4] 1
[3,4,2] 1
```

The figure shows the output of Itemset Mapper and Reducer on test.csv. The output contains itemset and frequency of the set.

5.2 Association Rule Mapper and Reducer

The second mapper takes all the output from the Itemset reducer (applicant Itemset). It then parse each itemset into the mapper. The support threshold is set to two, meaning that input set with one support will be excluded from the mapper.

The reducer will produce a key for each itemset for which the actual value of itemset will be ignored. With the key Reducer will use the Apriori algorithm and output the association rule into the final result file.



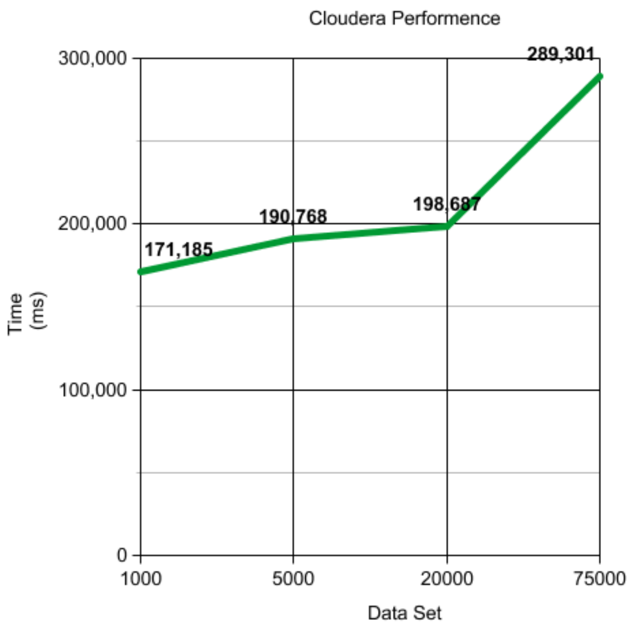
```
{1,2,3} -> 4
{1,2,4} -> 3
{1,2} -> 3
{1,2} -> 4
{1,3,4} -> 2
{1,3} -> 4
{1,3} -> 2
{1,4} -> 2
{1,4} -> 3
{1} -> 2
{1} -> 3
{1} -> 4
{2,3,4} -> 1
```

figure shows fragment of output from Association Rule reducer. Showing itemset and it association item.

5.3 Performance Comparison

Performance comparison of map tasks (ms) and reduce tasks (ms) on four Trac-dataset containing 1000, 5000, 20000, and 75000 transactions. Each dataset runs five times, and the average is recorded. The environment we are using is single-node Cloudera machine, and GSU owned DICE cluster.

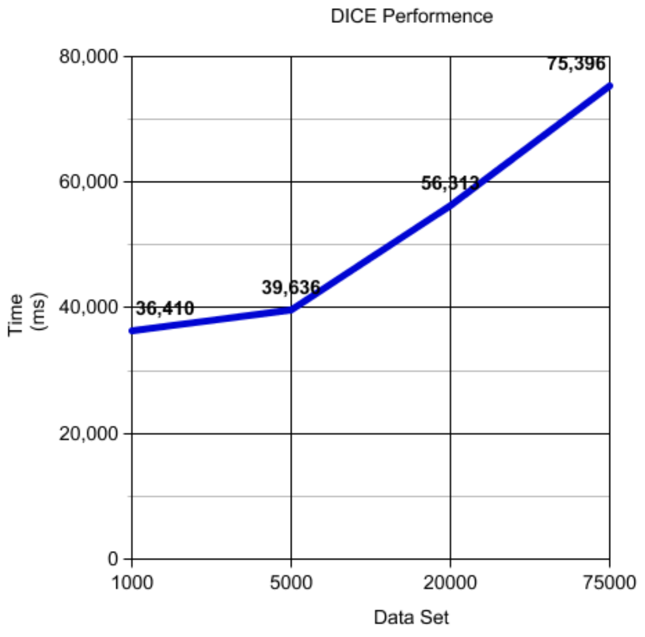
First, we run our test out Map/Reduce on Cloudera. The performance test will be on QuickStarts version of Cloudera CDH 5. Cloudera, Company based in California founded in 2008, is an enterprise level platform build for big data analyze. It includes Hadoop running in single node pseudo-distributed mode. Hadoop is distributed to all the core in a single machine. The file can be accessed via HDFS on local connection socket for communication between processes [13]. We will run Cloudera via virtual box with 5120 MB allocated memory and 70 GB of SSD storage.



Graph showing dataset size vs time (ms) running on cloudera.

DICE is a high performance computer cluster for big data analytics runs by Georgia State University's Department of Computer Science. DICE have 128TB distributed storage on HDFS, 100 Gbps communication speed between nodes, and 3.7 teraflop Intel Broadwell processor.

For this comparison, DICE job profile is set to 12 cores and 128 GB.



As we can see from the graph DICE perform about 4-5 times faster than that of Cloudera. Also data set from range 1000 to 5000 see no performance changes. This may due to the Hadoop takes more time to spawn reducer and mapper.

6. REFERENCES

[1] *Mining Frequent Itemsets – Apriori Algorithm. Digital image. Http://software.ucv.ro/~cmihaescu/ro/teaching/AIR/docs/Lab8-Apriori.pdf. N.p., n.d. Web..*

[2] *Association Analysis*. Digital image. *Https://www-users.cs.umn.edu/~kumar/dmbook/ch6.pdf*. N.p., n.d. Web.

[3] *Journal of Parallel and Distributed Computing*. Digital image. *Https://www.elsevier.com/journals/journal-of-parallel-and-distributed-computing/0743-7315?generatepdf=true*. N.p., n.d. Web.

[4] MapReduce Tutorial. (n.d.). Retrieved February 11, 2017, from https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

[5] Agrawal, Rakehs, and Ramakrishnan Srikant. *Fast algorithms for mining association rules*. Digital image. *Http://rakesh.agrawal-family.com/papers/vldb94apriori.pdf*. N.p., n.d. Web.

[6] Dean J. & Ghemawat S. (2004). *MapReduce: Simplified Data Processing on Large Clusters. Proc. of the 6th Symposium on Operation Systems Designing and Implementation (OSDI '04)*. San Francisco, CA, Google Inc.: 1- 13.

[7] "KDnuggets." *KDnuggets Analytics Big Data Data Mining and Data Science*. N.p., n.d. Web. 12 Feb. 2017. <http://www.kdnuggets.com/2016/05/hadoop-key-terms-explained.html>.

[8] Tutorialspoint.com. "*MapReduce Quick Guide.*" Www.tutorialspoint.com. N.p., n.d. Web. 12 Feb. 2017. <https://www.tutorialspoint.com/map_reduce/map_reduce_quick_guide.htm>.

[9] Hadoop.apache.org "Welcome to Apache Hadoop!". hadoop.apache.org. Retrieved 2017-02-12.

[10] "What Is the Hadoop Distributed File System (HDFS)?" IBM - Hadoop Distributed File System (HDFS) - United States. N.p., n.d. Web. 12 Feb. 2017. <http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/>

[11] "Continuuity Raises $10 Million Series A Round to Ignite Big Data Application Development Within the Hadoop Ecosystem." Yahoo! News. Yahoo!, 14 Nov. 2012. Web. 12 Feb. 2017. <https://finance.yahoo.com/news/continuuity-raises-10-million-series-120500471.html>.

[12] EXTENDED BAKERY Dataset for Mining Association Rule <https://wiki.csc.calpoly.edu/datasets/wiki/apriori>

[13] Cloudera <https://www.cloudera.com>

[14] Agrawal, Rakesh, and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules (n.d.): n. pag. Fast Algorithms. Web. <https://www.it.uu.se/edu/course/homepage/infoutv/ht08/vldb94_rj.pdf>

[15] Cheung, D., Han, J., Ng, V., Fu, A., & Fu, Y. (n.d.). A Fast Distributed Algorithm for Mining Association Rule. Retrieved from <http://hanj.cs.illinois.edu/pdf/FDM96.pdf>

[16] Singh, Sudharkar, Rahki Garg, and P K Mishra. "Review of Apriori Algorithm on Apache Hadoop MapReduce." N.p., n.d. Web. https://arxiv.org/ftp/arxiv/papers/1702/1702.06284.pdf