Coverage Pattern Mining Based on Map Reduce

by

ralla akhil, Shadaab Siddiqie, P Krishna Reddy, Anirban Mondal

in

CODS-COMAD

Report No: IIIT/TR/2020/-1

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
January 2020
Abhijit Jana, Prosenjit Bhattacharyya, and Anirban Mondal
International Institute of Information Technology
Hyderabad, Telangana, India

ABSTRACT

Pattern mining is an important task of data mining and involves the extraction of interesting associations from large databases. However, developing fast and efficient parallel algorithms for handling large volumes of data is a challenging task. The MapReduce framework enables the distributed processing of huge amounts of data in large-scale distributed environment with robust fault-tolerance. In this paper, we propose a parallel algorithm for extracting coverage patterns. The results of our performance evaluation with real-world and synthetic datasets demonstrate that it is indeed feasible to extract coverage patterns effectively under the MapReduce framework.

KEYWORDS

Data Mining, Knowledge Discovery, Coverage Patterns, MapReduce

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CAMSOMAD 2020, January 5-7, 2020, Hyderabad, India
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-9999-9/18/06...$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Pattern mining [1, 15] is an important task of data mining and involves the extraction of interesting associations from large databases. It has significant applications in market basket analysis, recommendation systems, and internet advertising. In pattern mining based applications, databases are typically huge; this necessitates fast and scalable pattern mining algorithms. This problem can be addressed by the development of parallel algorithms in large-scale distributed environments. In the literature, the MapReduce framework [7] has been introduced for enabling the distributed processing of huge amounts of data on a large number of machines in geographically distributed environments with robust fault-tolerance. Computations in the MapReduce framework are distributed among worker machines and are described by the map and reduce functions. The map function processes key-value pairs and the reduce function merges all the values associated with the same key.

Another useful type of pattern is the coverage pattern [15], which has several important and diverse applications in areas such as banner advertising [17], search engine advertising [4, 5] and visibility mining [8]. Given a transactional database and a set of data items, coverage pattern (CP) is a set of items covering a certain percentage of transactions by minimizing overlap among the transactions covered by each item of the pattern. In the literature, a level-wise CP mining algorithm, designated as CMine [15], and a pattern growth approach called CPPG [16] have been proposed to extract CPs from transactional databases.

Incidentally, MapReduce-based pattern mining approaches have been proposed for extracting frequent patterns [11, 18, 19], periodic frequent patterns [3], utility patterns [12, 14, 23] and sequential patterns [6, 10, 21]. MapReduce-based pattern mining was first studied in the context of frequent patterns by means of an iteration-based apriori MapReduce algorithm [1, 20]. In this paper, we propose a new algorithm, designated as CMineMR, for the parallelization of the CMine coverage pattern mining algorithm under the MapReduce framework. The results of our performance evaluation with real-world and synthetic datasets demonstrate that it is indeed feasible to extract coverage patterns effectively by using our proposed MapReduce-based CMineMR algorithm.

The remainder of this paper is organized as follows. In Section 2, we discuss background information concerning coverage patterns. In Section 3, we present the proposed approach. In Section 4, we report the performance evaluation. Finally, we conclude in Section 5 with directions for future work.

2 BACKGROUND INFORMATION

This section discusses background information concerning coverage patterns.

2.1 Model of Coverage Patterns

Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of items, $DB$ be the transactional database. Each transaction $T$ in $DB$ comprises a set of items i.e., $T \subseteq I$. |$DB$| represents the total number of transactions in database $DB$. $T^{T^p}$ represents the set of transactions, which contains the item $i_p$. |$T^{T^p}$| represents the number of transactions containing $i_p$. 
The concept of coverage patterns incorporates the following notions: relative frequency, coverage set, coverage support and overlap ratio. We shall now discuss each of these notions.

**Definition 1. Relative frequency of item** \( i_p \): The fraction of transactions containing a item \( i_p \) is called the Relative Frequency (RF) of \( i_p \) and is computed as \( RF(i_p) = |T^{i_p}| / |DB| \).

An item is considered to be frequent if its RF \( \geq \text{minRF} \), where \( \text{minRF} \) is a user-specified threshold.

**Definition 2. Coverage set** \( CSet(X) \) of a pattern \( X \): Given a pattern \( X = \{i_p, \ldots, i_q, i_r\} \), \((1 \leq p, q, r \leq n)\), \( CSet(X) \) is the set of all transactions containing at least one item of pattern \( X \), i.e., \( CSet(X) = T^{i_p} \cup T^{i_q} \cup \ldots \cup T^{i_r} \).

**Definition 3. Coverage support** \( CS(X) \) of a pattern \( X \): Given \( X = \{i_p, \ldots, i_q, i_r\} \), \((1 \leq p, q, r \leq n)\), \( CS(X) \) is the ratio of the size of \( CSet(X) \) to \( |DB| \), i.e., \( CS(X) = |CSet(X)| / |DB| \).

**Definition 4. Overlap ratio** \( OR(X) \) of a pattern \( X \): Given \( X = \{i_p, \ldots, i_q, i_r\} \), \((1 \leq p, q, r \leq n)\) and \( |T^{i_p}| \geq \ldots \geq |T^{i_q}| \geq |T^{i_r}| \), \( OR(X) \) is the ratio of the number of common transactions between \( CSet(X) = \{i_p\} \) and \( T^{i_r} \) to the number of transactions having item \( i_r \), i.e., \( OR(X) = \frac{|CSet(X-i_r) \cap T^{i_r}|}{|T^{i_r}|} \).

A pattern is interesting if it has high CS since it covers more number of transactions. Suppose we want to increase the coverage by adding a new item \( i_k \) to the pattern \( X \). The addition of item \( i_k \) will be more interesting if it adds more number of transactions for the coverage set \( CSet(X) \) of pattern \( X \). In essence, adding a new item \( i_k \) to pattern \( X \) could be interesting if there is a minimal overlap. Thus, a pattern having less OR could be more interesting.

**Definition 5. Coverage pattern** (CP): A pattern \( X = \{i_p, \ldots, i_q, i_r\} \), \((1 \leq p, q, r \leq n)\) and \( |T^{i_p}| \geq \ldots \geq |T^{i_q}| \geq |T^{i_r}| \) is called a coverage pattern if \( OR(X) \leq \text{maxOR} \), \( CS(X) \geq \text{minCS} \) and \( RF(i_k) \geq \text{minRF} \) \( \forall i_k \in X \), where \( \text{maxOR} \) and \( \text{minCS} \) are user-specified threshold values of maximum overlap ratio and minimum coverage support respectively.

Given a set \( I \) of items, transactional database \( DB \), \( \text{minRF} \), \( \text{minCS} \) and \( \text{maxOR} \), the problem of mining CPs is to discover the complete set of CPs.

**About sorted closure property:** The overlap ratio satisfies downward closure property if the items are ordered in descending order of their frequencies respectively. Such a property is called the sorted closure property [13].

**Sorted closure property:** Let \( X = \{i_p, \ldots, i_q, i_r\} \), \((1 \leq p, q, r \leq n)\) be a pattern such that \( |T^{i_p}| \geq \ldots \geq |T^{i_q}| \geq |T^{i_r}| \). If \( OR(X) \leq \text{maxOR} \), all of the non-empty subsets of \( X \) containing \( i_r \) and having size \( \geq 2 \) will also have overlap ratio less than or equal to maxOR.

An item \( a \) is said to be a non-overlap item w.r.t. a pattern \( X \) if \( OR(X, a) \leq \text{maxOR} \) and \( RF(i_k) \geq \text{minRF} \) \( \forall i_k \in \{X, a\} \). The notion of non-overlap pattern (NOP) is defined as follows.

**Definition 6. Non-overlap pattern** (NOP): A pattern \( X = \{i_p, \ldots, i_q, i_r\} \), \((1 \leq p, q, r \leq n)\) and \( |T^{i_p}| \geq \ldots \geq |T^{i_q}| \geq |T^{i_r}| \) is called a non-overlap pattern if \( OR(X) \leq \text{maxOR} \) and \( RF(i_k) \geq \text{minRF} \) \( \forall i_k \in X \).

## 2.2 CMine Algorithm

Similar to the apriori algorithm [1], CMine [15] is an iterative multi-pass algorithm for extracting CPs from a given transactional database. In case of CMine, NOPs of size \( k \) are used to explore size \( k+1 \) patterns. As NOPs satisfy sorted closure property, we extract NOPs, which satisfy the maxOR constraint. Next, CPs are extracted by applying the minCS constraint.

Let \( C_k \), \( NOP_k \) and \( CP_k \) denote the candidate, non-overlap and coverage patterns of size \( k \) respectively. At the \( k^{th} \) iteration, NOPs and CPs of size \( k \) are computed. Given \( \text{minCS} \), \( \text{maxOR} \), and \( \text{minRF} \) values, the steps of CMine algorithm for extracting CPs from the transactional database \( DB \) can be summarized as follows:

1. **First iteration:** The frequency of each item is computed by scanning \( DB \). After scanning, \( CP_1 \) and \( NOP_1 \) are computed by checking relative frequency. Item is added to \( NOP_1 \) if \( RF \geq \text{minRF} \), and added to \( CP_1 \) if \( RF \geq \text{minCS} \). The items in \( NOP_1 \) are sorted in descending order of their frequencies.

2. **Second iteration and beyond:** Starting from \( k=2 \), the following step is repeated until \( C_k \neq \emptyset \). \( C_k \) is generated by computing \( NOP_{k-1} \) \( \bowtie \) \( NOP_{k-1} \) (self-join). After scanning \( DB \), \( NOP_k \) and \( CP_k \) are computed by checking OR and CS of candidate patterns in \( C_k \) accordingly.

## 3 PROPOSED APPROACH

This section presents the proposed approach.

### 3.1 Basic Idea

We distribute \( DB \) across \( N \) machines and extract the CPs in a distributed manner. Let \( X = \{i_1, i_2, \ldots, i_n\} \) be a pattern, \( N \) be the number of machines and \( DB_i \) represent the \( i^{th} \) partition of \( DB \). The basic idea is to extract the CPs by checking the values of CS and OR by accessing the partitions of \( DB \).

The main issue is to compute the OR value of a candidate pattern in a distributed manner. Notably, as the value of \( OR(X) \) is a fraction, i.e., \( OR(X) = \frac{|CSet(X-i_k) \cap T^{i_k}|}{|T^{i_k}|} \), it cannot be computed by adding the OR values from the partitions of \( DB \). However, \( OR(X) \) can be computed efficiently by computing the numerator and the denominator of \( OR(X) \) independently under the MapReduce framework. It can be observed that the denominator \( |T^{i_k}| \) of \( OR \) is the frequency of a item. Hence, it is possible to compute the respective frequencies of all of the items in the first phase of MapReduce and store these frequencies in each of the \( N \) machines. Moreover, the value \( CSet(X) \) can be computed by aggregating the corresponding coverage sets from the partitions of \( DB \) stored in each machine in a distributed manner. Once the frequency of the \( n^{th} \) item \( i_n \) is with every machine, the value of \( OR(X) = \frac{|CSet(X-i_n) \cap T^{i_n}|}{|T^{i_n}|} \) can be computed in a distributed manner. Notably, the value of \( CS(X) \) in \( DB \) can be computed in a distributed environment by adding the coverage support values in each partition of the \( DB \).

The overview of the proposed approach under MapReduce is as follows. We distribute \( DB \) into \( N \) machines. In the first iteration, we compute relative frequency values of all items using one phase of MapReduce. We broadcast frequencies of all items to all machines. In the second iteration, we compute the CPs of size two by using one phase of MapReduce. From the third iteration onwards, we employ two phases of MapReduce; one phase is for generating candidate patterns, while the other phase is for computing CPs.

210
3.2 CMineMR Algorithm

Similar to the CMine algorithm, our proposed MapReduce algorithm is also an iterative algorithm. At the k\textsuperscript{th} iteration of the proposed CMineMR algorithm, NOP\textsubscript{k} and CP\textsubscript{k} of size k are computed. The input to the algorithm consists of a transactional database DB, minRF, minCS and maxOR. First, DB is segmented into multiple partitions and each partition is loaded onto each machine.

(i) First iteration: In this iteration we explain the generation of NOP\textsubscript{1} and CP\textsubscript{1}. Each mapper reads each transaction of the data partition and maps each item to 1. Reducer groups all the item counts of each item into a list, which we designate as count-list. Then item frequencies are computed by adding counts in count-list of an each item. Algorithm 1 depicts the procedure to compute frequencies of size one itemsets. The NOP\textsubscript{1} and CP\textsubscript{1} are computed by comparing the relative frequencies with minRF and minCS respectively. The frequencies of NOP\textsubscript{1} are broadcast among all machines; this is used in the subsequent iterations.

Algorithm 1 First iteration-Computing CP\textsubscript{1}, NOP\textsubscript{1} (DB)

\begin{algorithm}
\begin{algorithmic}
\Procedure{Map}{key = null, value = DB\textsubscript{1}}
\ForEach {t\textsubscript{i} in DB\textsubscript{1}}
\ForEach {i\textsubscript{k} in t\textsubscript{i}}
output < i\textsubscript{k}, 1 >
\EndFor
\EndFor
\EndProcedure
\Procedure{Reduce}{key = ik, value = count-list(i\textsubscript{k})}
\ForEach {count in count-list(i\textsubscript{k})}
i\textsubscript{k}.freq += count
\EndFor
\EndProcedure
\end{algorithmic}
\end{algorithm}

(ii) Second iteration: In this iteration, we explain the generation of NOP\textsubscript{2}, CP\textsubscript{2}. In the second iteration, the candidate patterns are computed by joining the non-overlap patterns of the first iteration. The CP\textsubscript{2} are broadcast across all machines. The OR, CS of CP\textsubscript{2} are computed using one MapReduce phase.

For each transaction (t\textsubscript{i}) and candidate pattern (P), mapper maps the P to [x:y] of the form: <P,[x:y]> . The first component x is 1 if t\textsubscript{i} has at least one item of the P. The second component y is 1 if t\textsubscript{i} has the least frequent item of P and at least one item among the remaining items. Reducer groups all the counts of each pattern into a list, which we designate as counts-list. Then the corresponding integers of each P are added. Algorithm 2 depicts the procedure to compute the size of coverage set and the numerator of overlap ratio of candidate patterns of size k (k > 1). After reduction, the CS of P is computed by dividing the first component with the total number of transactions. The OR of P is computed by dividing the second component with the frequency of the least frequent item (broadcast in the first iteration).

(iii) Third iteration and beyond: In this iteration, we explain the generation of NOP\textsubscript{k}, CP\textsubscript{k} (k > 2). From the third iteration onwards, CP\textsubscript{k} are generated using one MapReduce phase. The OR and CS of CP\textsubscript{k} are computed using another MapReduce phase. This procedure of two MapReduce phases is repeated until no new candidate patterns are generated.

For each pattern P in NOP\textsubscript{k-1}, mapper maps the pattern having the first k-2 items of P to the least frequent item of P. Reducer groups all the least frequent items based on the key into a list, which we designate as item-list. For each size k-2 pattern, CP\textsubscript{k} are generated by iterating over the item-list as shown in Algorithm 3. Algorithm 3 depicts the procedure to compute CP\textsubscript{k}. The value of CP\textsubscript{k} is broadcast across all machines. The CS and OR of CP\textsubscript{k} are computed by another MapReduce operation, which is similar to the Second iteration.

Algorithm 2 k\textsuperscript{th} iteration-Computing CP\textsubscript{k}, NOP\textsubscript{k} (DB, C\textsubscript{k})

\begin{algorithm}
\begin{algorithmic}
\Procedure{Map}{key = null, value = DB\textsubscript{i}}
\ForEach {t\textsubscript{i} in DB\textsubscript{i}}
\ForEach {i\textsubscript{k} in t\textsubscript{i}}
output < i\textsubscript{k}, 1 >
\EndFor
\EndFor
\EndProcedure
\Procedure{Reduce}{key = ik, value = count-list(i\textsubscript{k})}
\ForEach {count in count-list(i\textsubscript{k})}
i\textsubscript{k}.freq += count
\EndFor
\EndProcedure
\end{algorithmic}
\end{algorithm}

by iterating over the item-list as shown in Algorithm 3. Algorithm 3 depicts the procedure to compute CP\textsubscript{k}. The value of CP\textsubscript{k} is broadcast across all machines. The CS and OR of CP\textsubscript{k} are computed by another MapReduce operation, which is similar to the Second iteration.

Algorithm 3 k\textsuperscript{th}(k > 2) iteration-Computing CP\textsubscript{k} (NOP\textsubscript{k-1})

\begin{algorithm}
\begin{algorithmic}
\Procedure{Map}{key = null, value = NOP\textsubscript{k-1}}
\ForEach {X = [i\textsubscript{1}, i\textsubscript{2}, ..., i\textsubscript{k-1}] in NOP\textsubscript{k-1}}
output < [i\textsubscript{1}, i\textsubscript{2}, ..., i\textsubscript{k-2}], i\textsubscript{k-1} >
\EndFor
\EndProcedure
\Procedure{Reduce}{key = X, value = item-list(X)}
\ForEach {i\textsubscript{n} in item-list(X)}
\ForEach {i\textsubscript{q} in item-list(X)}
if Freq(i\textsubscript{n}) < Freq(i\textsubscript{q}) then
\{i\textsubscript{1}, i\textsubscript{2}, ..., i\textsubscript{k-2}, i\textsubscript{n}, i\textsubscript{q}\}
\EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

4 PERFORMANCE EVALUATION

We have conducted experiments by implementing our proposed CMineMR algorithm as well as the reference CMine algorithm in Python 2.7. The CMineMR algorithm is written using Apache Spark architecture [22] and it is performed in a cluster of 24 machines, with 2 GB memory each. The experiments on the reference CMine algorithm [15] are performed in one machine of the cluster.

Table 1: Parameters used in our experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Parameter</th>
<th>Default value</th>
<th>Variations</th>
<th>step-size</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS-POS</td>
<td>N/of Machines</td>
<td>8</td>
<td>[4,6,8,10, 12,16,20,24]</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>(N/M)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DB</td>
<td>515,596</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>minRF</td>
<td>0.065</td>
<td>[0.065, 0.095]</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>minCS</td>
<td>0.5</td>
<td>[0.1, 0.9]</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>maxOR</td>
<td>0.6</td>
<td>[0.1, 0.9]</td>
<td>0.1</td>
</tr>
<tr>
<td>Synthet</td>
<td>N/of Machines</td>
<td>8</td>
<td>[4,6,8,10, 12,16,20,24]</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>(N/M)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DB</td>
<td>100,000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>minRF</td>
<td>0.045</td>
<td>[0.045, 0.06]</td>
<td>0.0025</td>
</tr>
<tr>
<td></td>
<td>minCS</td>
<td>0.3</td>
<td>[0.1, 0.9]</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>maxOR</td>
<td>0.5</td>
<td>[0.05, 0.5]</td>
<td>0.05</td>
</tr>
</tbody>
</table>
The experiments were conducted on two datasets. The first dataset is BMS-POS [9] dataset, which is a click-stream dataset of an e-commerce company; this dataset has 515,596 transactions and 1656 distinct items. The second dataset is the T104D100K, which is a synthetic dataset [2] generated by a dataset generator. This dataset has 100,000 transactions and 870 distinct items.

Our experiments are conducted by varying the number of machines (NM), maxOR, minCS and minRF. Table 1 summarizes the parameters used in our experiments. As the performance metric, we use execution time (ET), which is the total processing time (in seconds) for extracting CPs during the course of the experiment.

Figure 1: Effect of variations in NM

Figure 2: Effect of variations in maxOR

Figure 3: Effect of variations in minCS

(i) Effect of variations in NM: Figure 1 depicts the effect of variations in NM. For BMS-POS, the results are shown in Figure 1(a). The ET of CMineMR decreased rapidly till NM=8 due to a large amount of parallel computation in extracting CPs. However, the change in ET decreases with increase in NM and reaches saturation when 16 machines are used due to increase in the communication cost. The proposed CMineMR algorithm is 3.2 times faster than CMine algorithm when NM is 8. Similar trend is observed in Synthetic dataset as shown in Figure 1(b).

(ii) Effect of variations in maxOR: Figure 2 depicts the effect of variations in maxOR. For BMS-POS, the results are shown in Figure 2(a). The ET of CMine and CMineMR increases with the increase in maxOR, as the number of non-overlap patterns generated increases, thereby eventually increasing the runtime of the algorithms. The ET of CMineMR is 2.1 times faster than that of CMine algorithm when maxOR is 0.9 due to a significant amount of parallel computation in extracting CPs. The results for Synthetic dataset are shown in Figure 2(b).

(iii) Effect of variations in minCS: Figure 3 depicts the effect of variations in minCS. For BMS-POS, the results are shown in Figure 3(a). Notably, the CPs in each iteration are generated by checking minCS of NOPs, thereby leading to no significant changes in ET for CMine and CMineMR due to variations in minCS, as shown in Figure 3(a). The results for Synthetic dataset are depicted in Figure 3(b).

(iv) Effect of variations in minRF: Figure 4 depicts the effect of variations in minRF. The results for BMS-POS, Synthetic datasets are depicted in Figures 4(a) and 4(b) respectively. The decrease in ET for CMine and CMineMR with increase in minRF represents the decrease in the number of size-one frequent itemsets (items satisfying minRF). For BMS-POS, the gradual decrease in ET indicates that there are small changes in the number of size-one frequent itemsets with increase in minRF. However, for Synthetic dataset, there is a sudden fall in ET, which indicates that most of the items are having comparable frequencies.

5 CONCLUSION

In pattern mining, developing fast and efficient parallel algorithms handling large volumes of data becomes a challenging task. In this paper, we have introduced the problem of parallel mining in the context of coverage patterns and proposed the CMineMR algorithm for efficiently extracting the knowledge of coverage patterns. The results of our performance evaluation with real-world and Synthetic dataset demonstrate that it is indeed feasible to extract coverage patterns effectively using our proposed CMineMR algorithm under the MapReduce framework. As part of future work, we plan to develop parallel algorithms for pattern growth approach towards extracting coverage patterns. Furthermore, we plan to investigate the parallel coverage pattern extraction by considering issues such as skew in transactional databases and load-balancing.

ACKNOWLEDGEMENTS

This work is supported by Kohli Center on Intelligent Systems, IIT-Hyderabad. The authors are thankful to the anonymous referees for their useful comments.
REFERENCES