Frequent Itemset Mining Algorithms: A Survey

[1]Abdul Rahman,[2]Arati Manjaramkar,

Department of Information Technology,
SGGSIE&T, Nanded, Maharashtra, India

Abstract: Feature extraction have been commonly recognized as a dominant method to discover additional information from huge scale database. One of the feature extraction method is general interdependence rules extraction through grouping. This method is used to determine extra valuable information other than normal regular interdependence rules by taking user related knowledge into consideration. Frequent Itemset Mining (FIM) play a fundamental role in interdependence rule elements extraction, so as to serve in various feature extraction jobs. It supports to recognize collection of elements, features, signs etc. that usually occur in our databases. Ever since its beginning, a numeral important frequent item sets mining algorithm has been established to accelerate extraction performance. In this paper, we have discussed different algorithms for both sequential extraction as well as parallel extraction of repeated elements. So different algorithms to do have a variety of compromises among communication and computation, utilization of memory, synchronization, and the consumption of problem specific knowledge.

***Index Terms* - Information Mining, Association Rules, Frequent Itemset Mining, Hadoop Map Reduce**

1. INTRODUCTION

Extraction of hidden knowledge is a method of mining earlier unknown and possibly valuable knowledge (for example information rule, limitations, symmetries) from databases [1]. Through information determining in databases, relevant information, sequence of elements, or complex knowledge can be mined from important collections of databases. This information then can be examined from variety of viewpoints. Therefore, huge databases can be considered as ironic and consistent bases for information production and confirmation.

Extraction of knowledge and information from huge database has been identified by numerous scholars as a crucial topic in database systems and machine learning. Several business companies have been using extraction methods as a vital field with a chance of main revenues. The determined data can be used for data management, question processing, decision support system, procedure controller, and numerous further areas. Scholars in various dissimilar areas, containing database systems, information system, artificial intelligence, machine learning, and information gaining have revealed great curiosity in feature extraction.

2. BACKGROUND

2.1 Mining Association Rules

Consider an example of auctions transaction database. It is required to determine the vital interdependences amongst the elements such that the occurrence of few elements in an operation will indicate the occurrence of another elements in the similar operation. A scientific prototype was suggested in [2] to discuss the issues of extracting interdependence rules. Consider $I = i_1; i_2; \ldots .. i_m$ be a collection of literal, called as elements. Let D be a collection of operations, where every operation T is a collection of elements such as $T \subset I$. Notice that the number of elements bought by a customer in an operation are not well thought-out, it means that every element is a twofold variable denoting if an element was purchased. Every operation is related with an identifier, representing as T_{ID}. Consider X be a collection of elements. An operation T is said to have X as long as X is a subset of T. An interdependence rules is an inference of the structure such as $X \Rightarrow Y$, where $X \in I$, $Y \in I$, and $X \cap Y = \Phi$. The rules $X \Rightarrow Y$ hold in the operation collection D with confidence c if C% of operations in D that have X also have Y. The rules $X \Rightarrow Y$ have support s in the operation collection D if S% of operations in D have $X \cup Y$.

As per [2 - 4], the difficulty of extraction of interdependence rule is disintegrated into the succeeding two stages:

1. Determine the huge elements, i.e., the collections of elements that contain operation support beyond a predefined least support s.
2. Use of the huge elements to produce the interdependence rule for the databases.

2.2 Frequent Itemset Mining

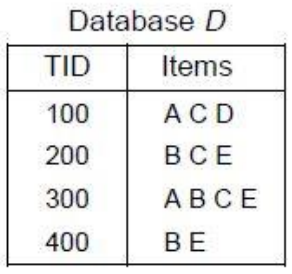
Repeated elements extraction is an essential problem in association rule mining (ARM), sequence extraction, and alike. Accelerating the method of frequent itemset mining (FIM) is serious and essential, since FIM feeding accounts for a major portion of extraction time because of its high calculation and input/output (I/O) intensity. Repeated elements extraction algorithms can be distributed into two groups namely, sequential extraction and parallel extraction.

3. SEENTILA MINING ALGORITHMS

There are two kinds of techniques that can be used for the extraction of repeated elements. First technique is based on producing the candidate elements. The examples of this technique include the Apriori algorithm and Direct Hashing with Efficient Pruning (DHP) algorithm. Second technique is based on without producing the candidate elements, which includes the FP-growth algorithm and frequent items ultrametric tree (FUIT). The following section gives a description on each of the techniques.

3.1 Apriori Algorithm

Agrawal et al. [3] proposed a fast algorithm for mining association rules in large databases, known as Apriori algorithm. Take an example of operation databases as shown in Figure 1. In every step, Apriori builds a candidate collection of huge elements, sums the amount of incidents of each candidate elements, and then calculates huge elements founded on a predefined least support. In the initial step, Apriori only searches all the operations to calculate the amount of incidents for every element. The collection of candidate 1-elements, C_1, found is presented in Figure 2.



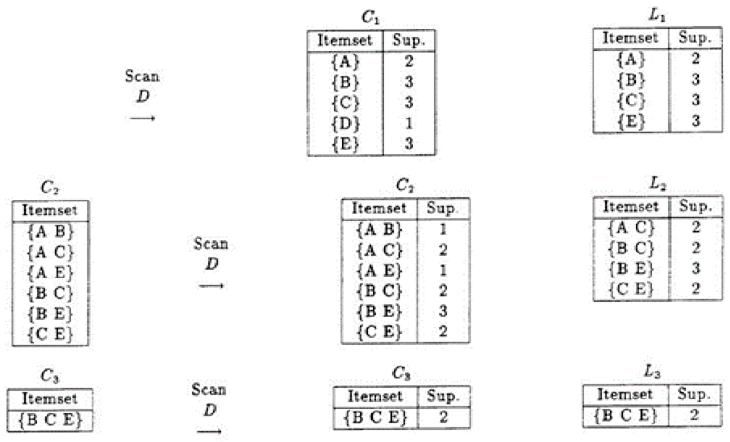
Database *D*

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Figure 1. A Transaction Database.

Suppose that the least operation support needed is 2 (such as, s = 40%). The collection of huge 1-elements, L_1, consists of candidate 1-elements with the least support mandatory, can then be calculated. To determine the collection of huge 2-elements, in sight of the point that any subclass of a huge elements essential have least support, Apriori algorithm requires $L_1 * L_1$ to produce a candidate collection of elements C_2 where * is an action for concatenation in this situation. C_2 contains of 2-elements. Then, the four operations in D are searched and the support of every candidate elements in C_2 is calculated. The second row of mid table in Figure 2 denotes the outcome from such calculating in C_2. The collection of huge 2-elements, L_2, is consequently calculated which relies on the support of every candidate 2-elements in C_2.

The collection of candidate elements, C_3, is produced from L_2 as below. From L_2, two huge 2-elements with the similar first element, such as (BC) and (BE), are recognized first. After that, Apriori checks whether the 2-elements (CE), which contains of their next elements, creates a huge 2-elements or not. As (CE) is a huge element by its-own, we see that all the subclasses of (BCE) are huge and then (BCE) converts to a candidate 3-elements. There is no other candidate 3-elements from L_2. Then Apriori searches all the operations and identify the huge 3-elements L_3 in Figure 2. As close by, there is no candidate 4-elements to be created from L_3, Apriori stops the procedure of determining huge elements.



C_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A B}
{A C}
{A E}
{B C}
{B E}
{C E}

C_2

Itemset	Sup.
{A B}	1
{A C}	2
{A E}	1
{B C}	2
{B E}	3
{C E}	2

L_2

Itemset	Sup.
{A C}	2
{B C}	2
{B E}	3
{C E}	2

C_3

Itemset
{B C E}

C_3

Itemset	Sup.
{B C E}	2

L_3

Itemset	Sup.
{B C E}	2

Figure 2. Candidate Set Generation in Apriori.

The Apriori Algorithm has a decent performance increased by minimizing the dimension of candidate collections. But, in circumstances with a huge amount of repeated sequence of elements, long sequence of elements, or pretty low least support, an Apriori-like algorithm may have the subsequent two nontrivial expenses:

1. It is expensive to support a large amount of candidate collections. For instance, if there are 10^4 repeated 1-

elements, the Apriori algorithm will require generating greater than 10^7 length 2-candidates. Then it needs to gather and checks their occurring regularities. Furthermore, to determine a repeated sequence of element of size 100, such as $a_1, \ldots, a_{100}$, it needs to produce $2^{100} - 2 = 10^{30}$ candidates in overall. This is the natural cost for candidate production, it does not matter what execution method is used.

2. It is boring to constantly search the databases and test a huge collection of candidates by sequence of element comparing, which is particularly right for extraction of large sequence of elements.

3.2 Apriori on Hadoop MapReduce

Kovacs et al. [19] presented parallel frequent itemset mining on hadoop. Hadoop is a framework which uses the MapReduce programming. To implement an algorithm on MapReduce the main tasks are to design two independent maps and reduce functions for the algorithm and to convert the datasets in the structure of the form (key, 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 phase of map-reduce to get the final result.

Apriori algorithm is an iterative process and its two main components are candidate elements generation and repeated elements generation. In each search of database, mapper generates native candidates and reducer sums up the native counting and generates repeated elements. The Count Distribution [See Section 4.1] parallel version of Apriori is best suited on Hadoop where as to implement Data Distribution [See Section 4.2] algorithm we have to control the distribution of feature which is automatically controlled by Hadoop [20].

The first step of the algorithm is to generate repeated 1-elements L_1 which is illustrated in Figure 3 by an example. HDFS breaks the operational database into blocks and distribute to all mappers running on machines. Each operation is translated into (key, value) pair, where key is the T_{ID} and value is the list of elements i.e. operation. Mapper reads an operation at any given time and produces an output (key', value') pair where key' is each element in operation and value' is 1. The combiner combines the pairs with same key' and makes the native sum of the values for each key'. The output pairs of all combiners are shuffled and exchanged to make the list of values related with same key', as (key', list (value')) pairs.

Reducers take these pairs and sum up the values of respective keys. Reducers output (key', value') pairs where key' is element and value' is the support count least support, of that element [21 - 23]. Finally, repeated 1-elements L_1 is obtained by merging the output of all reducers. To generate repeated k-elements L_k, each mapper reads repeated elements L_{k-1} from previous iteration and generates candidate elements C_k from L_{k-1} as in traditional algorithm. A

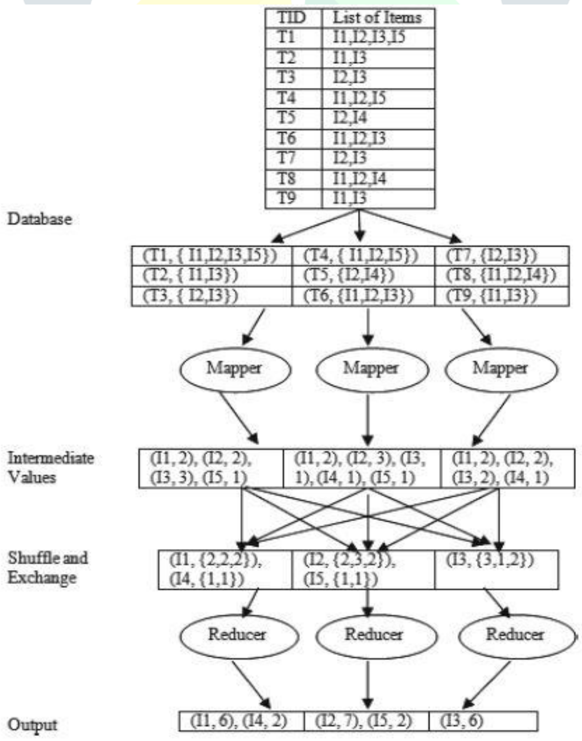

TID | List of Items
T1 | I1,I2,I3,I5
T2 | I1,I3
T3 | I2,I3
T4 | I1,I2,I5
T5 | I2,I4
T6 | I1,I2,I3
T7 | I2,I3
T8 | I1,I2,I4
T9 | I1,I3
Database
(T1, { I1,I2,I3,I5}) (T2, { I1,I3}) (T3, { I2,I3})
(T4, { I1,I2,I5}) (T5, {I2,I4}) (T6, {I1,I2,I3})
(T7, {I2,I3}) (T8, {I1,I2,I4}) (T9, {I1,I3})
Mapper
Mapper
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.

candidate element collection in C_k is selected as key and assigned a value 1, if it is present in the operation assigned to the mapper. Now we have (key, value) pairs where key is k-elements and value is 1. All the remaining procedures is same as generation of L_1.

3.3 Direct Hashing with Efficient Pruning (DHP) Algorithm

Park et al. [4] proposed an effective hash-based algorithm for mining association rules, called Direct Hashing with Efficient Pruning. Like Apriori, DHP as well produces candidate k-elements from L_{k-1}. But, it consists of a hash table, which is constructed in the earlier iteration, to examine the suitability of a k-elements. For all k-elements from $L_{k-1} * L_{k-1}$ to C_k, it augments a k-elements to C_k on a condition that k-elements is hashed to a hash access whose value is larger than or identical to the least operation support needed. By doing this, the candidate collection C_k has considerably amount of size. Such a cleaning method is mainly influential in decreasing the dimension of C_2. It also minimizes the database dimensions gradually by not only pruning every individual operation dimensions as well as trimming the amount of operations in the database.

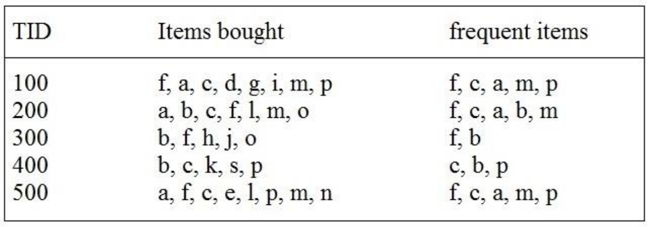
Notice that either DHP or Apriori are recursive procedures on the huge elements dimensions in the logic that the huge k-elements are resulting from the huge (k-1)-elements. These huge elements counting methods are in fact appropriate to handle with other feature extraction issues [7 - 8].

3.4 Frequent Pattern (FP) Growth Algorithm

Han et al. [9] presented a new algorithm for mining frequent pattern without candidate generation, called FP-Growth. FP-Growth calculation utilizes a tree structure called FP-tree [6]. There are two techniques to go over the FP-tree to remove the rehashed grouping of components which will be utilized to create the reliance rules.

3.4.1 Construction of the tree structure

The creation of the FP-tree needs two searches of the operation databases. The initial search stores the support of every element and then picks elements that fulfil least support, i.e. repeated 1-elements. The supports of the descendants of every element are also stored. These elements are arranged based on occurrence in descendant order to create F-list. The next search builds the FP-tree.



TID	Items bought	frequent items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Figure 4. A Transaction Database

Initially, add the descendants of every element in the operation. After that the operations are rearranged based on the F-list, whereas non-repeated elements are uncovered. Finally, the rearranged operations are injected onto the FP-tree. In this method insert fp-tree, if the node conforming to the elements in operation be present the counting of the node is augmented, or else a fresh node is produced and the sum is combined to one. The identical arrangements of the elements play vital part for the optimization of the database as mutual prefaces can be shared between lots of operations. Figure 4 and 5 depicts the FP-tree construction.

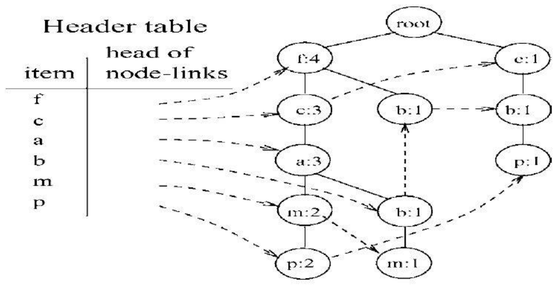


Figure 5. Construction of FP-Tree.

3.4.2 Finding Frequent Items

Bottom Up Traversal Method: In Bottom up fp-tree (BU-FPtree) the nodes are traverses beginning from the smallest repeated element in F-list. Whereas staying at every node, BU-FPtree as well gathers the affix trail of the node, which is the collection of elements on the trail from the suffix node to the root of the tree. BU-FPtree similarly keeps the value on the node as the count of the affix trail. The affix trail produced is so called restricted sequence of element of that element. A small database of repeated sequence of elements that co-occur with the element is called the restricted sequence of element. After that BU-FPtree construct a small fp-tree from the restricted sequence of element called restricted fp-tree. Throughout every pass, a fresh repeated element is produced by augmenting the suffix to the elements from the preceding pass. BU-FPtree also take care of a list of descendants anc-list for the elements in the present element.

When BU-FPtree come upon such situations, the pass is stopped and BU-FPtree go for the ensuing element. The procedure is recursively repeated till no restricted sequence of element can be produced and all repeated sequence of elements that contain the element are determined.

Top-Down Traversal Method: The Top-down algorithm is stimulated by algorithms Top-Down FP-tree(TD-FPtree) [10]. The benefit of using this technique is no necessity to build the restricted sequence of element and the sub-trees. The cleaning techniques alike with Bottom-Up FP-tree are as well hired to evade the redundant searching of elements which happen with their descendants.

TD-FPtree removes the generation of restricted sequence of element by straight re-joining the node relations of the fp-tree. At the time of re-joining, TD-FPtree as well adjusts the value information in the fp-tree nodes. For every different suffix extension lead, for example y, a different header table H-y is build, where H represent Header table and y represents path from the node to the root node. The header table comprises of elements earlier to the element y in the preceding header table. After that going with the node links of y, the affix trail to the root node are searched. While traversing the fp-tree, the node links of the fp-tree nodes are re-joined to header table H-y. The values in header table and in the fp-tree node are as well improved comparative to its co-occurrences with y. This procedure is recursively repeated.

3.5 Frequent Items Ultrametric Tree

The FP-growth consists of three major features: (1) It consists the fp-tree, which is typically lesser than the original databases, therefore is storage proficient. (2) It executes a sequence of element growing technique to evade the expensive candidate producing step by merging repeated elements in the fp-tree. (3) a partitioned-based divide-and-conquer finding method is suggested in extracting repeated elements. Nevertheless, the blockage of FP-growth is the lessening of searching and building expense of the fp-tree and its restricted fp-trees. Therefore, the two main drawbacks of the FP-growth are, Initial, it is not possible to construct a main-memory based FP-tree if the database is huge and large. In addition, its requirement is to build a restricted sequence of element and restricted sub-fp-trees for individually smaller sequence of element. In order to traverse for lengthier sequence of elements, the gaining of outcomes is very period and space taking processes as the amount of iteration and sequence of elements rises.

Tsay et al. [11] proposed a novel approach for mining frequent itemsets, Frequent Items Ultrametric Tree (FIUT). The FIUT technique works on well-organized ultrametric tree format, also called as the FIU tree, to show repeated elements for extracting repeated elements. The suggested FIUT contains two important stages in two searches of database D. Stage 1 starts by calculating the support for all elements that occur in the operation records. Then, a trimming method is developed to eliminate all non-repeated elements; departure only repeated elements to produce the k-elements, where the amount of repeated elements of a operation is k in a database. In the meantime, all the repeated 1-elements are produced. Stage 2 is the iterative building of shorter ultrametric trees, the real extracting of these trees, and their issue. The method for extracting repeated k-elements, as recommended in this paper, primary construct an independent, comparative k-FIU-tree for all k-elements, where k from M down to 2, and M represents the huge value of k amongst the operations in the database. Then, every tree is extracted independently, without producing candidate k-elements. The k-FIU-tree is removed effectively after repeated k-elements are extracted. Only one k-FIU-tree will be there in the main memory at any given time.

4. PARALLEL MINING ALGORITHMS

Sequential FIM techniques undergoes from performance degradation when it is applied on massive amount of data on a single machine. To deal with this issue parallel FIM techniques are used for extraction of repeated elements. Therefore, different parallel algorithms for extraction of interdependence rules are discussed below and the issue of extracting interdependence rule is considered on a shared nothing multiprocessor.

4.1 Count Distribution

Agrawal et al. [12] presented a parallel mining algorithm. They have proposed three algorithms (Count Distribution, Data Distribution, and Candidate Distribution) for parallel mining of association rules. Count Distribution procedure uses a basic norm of letting redundant computations in parallel otherwise idle CPUs to avoid communication. The primary step is distinctive. For every other step k-1, the procedure functions as below:

1. Every CPU P_i produces the whole C_k, using the whole repeated elements L_{k-1} produced at the end of step k-1. Note that ever since every CPU has the same L_{k-1}, they will be producing alike C_k.
2. CPU P_i take a step above its feature dividing (D_i) and grows native support count for candidates in C_k.
3. CPU P_i interchanges native C_k count with all other CPUs to produce global C_k count. CPUs are required to coordinate in this step.
4. Each CPU P_i now calculates L_k from C_k.

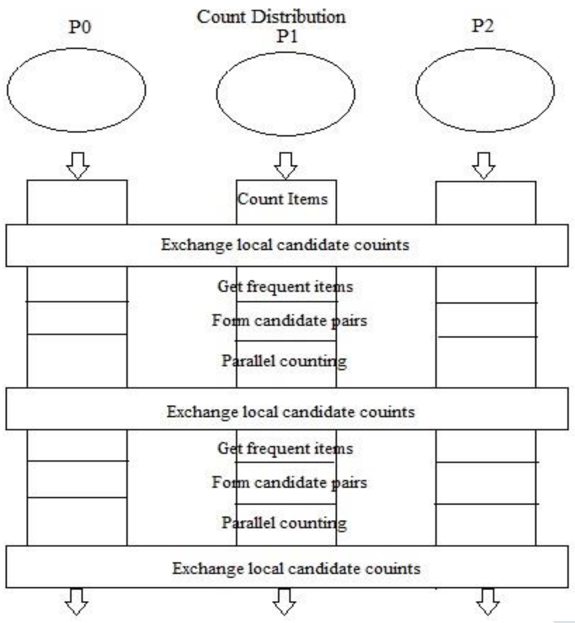


Figure 5. Overview of Count Distribution Algorithm.

5. Each CPU P_i individually take a choice to dismiss or continue to the next step. The choice will be same as the CPUs all have equal L_k.

In the initial pass, every CPU P_i with dynamism produces its native candidate elements C_i based on the elements essentially exists in its native feature fragments D_i. Therefore, the candidates calculated by dissimilar CPUs might not be equal and attention must have been occupied in swapping the native counts to decide global C_1.

Therefore, in all step, CPUs can search for the native feature asynchronously in a parallel manner. But, they need to coordinate at the completion of every step to produce over-all count.

4.2 Data Distribution

The striking characteristic of the Count Distribution procedure is that no feature tuples are replaced amongst CPUs, only counts are replaced. Therefore, CPUs can work individually and asynchronously and analyses the features in the database. On the other hand, the drawback is that this algorithm does not abuse the cumulative storage of the systems proficiently. Assume that individual CPU consumes storage of M. The amount of candidates that can be calculated in single step is calculated by M. As we grow the amount of CPUs from 1 to N, the computer have N x M overall storage, but still we can calculate the identical amount of candidates in single step, as individual CPU is including equal candidates. The Count Distribution procedure count no further candidates per step than the sequential procedure.

The Data Distribution procedure [12] is aimed to achieve better the entire systems storage as the amount of CPUs grow. In this particular process, individual CPU count commonly candidates. Therefore, as the number of CPUs grows, a more amount of candidates can be calculated in a step. On an N-CPU arrangement, feature will be allowed to sum in a one-step a candidate collection but Count Distribution would need N steps. The problem of this procedure is that each CPU must forecast its native feature to another CPUs in each step. Thus, this procedure can grow into worthwhile only on a device with actually fast messaging.

Pass 1: Identical as the Count Distribution algorithm.
Pass k-1:

1. CPU P_i produces C_k from L_{k-1}. It recollects merely 1/Nth of the elements making the candidates sub-class C_k that it will sum. Which 1/N elements are reserved is calculated by the CPU ID and can be calculated deprived of interactivity with another CPUs.
2. CPU P_i grows support values for the elements in its native candidate collection C_k using native feature page as well as feature page obtained from another CPUs.
3. At the end of the above step, every CPU P_i computes L_k using the native C_k. Once more, all L_k collections are separate, and the combination of all L_k collections is L_k.
4. CPUs rearrange L_k accordingly to each CPU has the whole L_k for producing C_{k+1} for the next step. This phase needs CPUs to coordinate. After obtaining the whole L_k, individual CPU can individually choose whether to dismiss or remain on to the next step.

The exciting phase is Step 2 in which CPUs grows the support values for native candidates C_k asynchronously. For the duration of this step, CPUs are forecasting their native feature in addition to getting the native feature of another CPUs. We need to be suspicious to evade network traffic and use non-synchronous messages to overlap messages time with the calculating of support. Complete description is given in [16].

4.3 Candidate Distribution

One restriction of Count Distribution as well as Data Distribution algorithm is that as any databases operation might maintain any candidate elements, individual operation need to be matched alongside the whole candidate collection. This is what necessitates count to replicate the candidate collection on each CPU and feature to forecast each databases operation. Moreover, Count Distribution as well as Data Distribution algorithms needs CPUs to coordinate at the end of every step to interchange values or repeated elements correspondingly. If the amount of work is not seamlessly balanced, this can be a problem to all the CPUs to wait for either CPU completes last in each step. These issues are because of the point that neither Count nor Data Distribution exploits issue related information; feature tuples and candidate elements are divided simply to evenly partition the load. All CPUs need to be referred and all knowledge collected beforehand they can be used to the subsequent step.

The Candidate Distribution [12] procedure tries to do away with this reliance by dividing the feature as well as the candidates in such a manner that individual CPU might well execute independently. In step 1, this procedure splits the repeated elements L_{k-1} among CPUs in a manner that a CPU P' can produce a distinctive C_k, not dependent of all another CPUs. Meanwhile, feature is re-partitioned so as to a CPU can sum the candidates in C_k not dependent of all another CPU. Notice that upon resting on the excellence of the elements dividing, fragments of the databases might have to be duplicated on numerous CPUs. The elements dividing procedure contemplates this feature by recognizing parts of L_{k-1} that are probably maintained by dissimilar databases operations. The option of the relocation step is a compromise amongst decoupled CPU dependency the moment possible and waiting till the elements turn out to be more simply and fairly partitionable.

Afterward this Candidate Distribution, individual CPU proceed independently, counts only its parts of the global candidate collection by means of only native feature. No messaging of values or feature tuple is eternally needed. The only requirement a CPU has on another CPUs is for trimming the native candidate collection throughout the trim phase of candidate production. Nevertheless, this data is send non-synchronously, and computers do not wait for the whole trimming data to reach from all another CPUs. For the duration of the trim phase of candidate production, it trims the candidate collection as much as probable by means of whatever data has reached, and resourcefully begins calculating the candidates. Then later trimming data can instead be used in succeeding steps [12].

4.4 PFP: Parallel FP-Growth

Li et al. [17] proposes a parallel FP-growth algorithm for query recommendation, they have called it PFP (Parallel FP-growth). Assumed an operation database, PFP works on three Map Reduce jobs to parallelize FP-Growth. Following are the phases used in PFP.

Stages of PFP:

Stage 1: Sharding: Partitioning database into consecutive fragments and putting away the fragments on P dissimilar PCs. Such a splitting and distribution of feature is known Sharding, and every fragment is known as a shard.

Stage 2: Parallel Counting: Performing a Map Reduce job to calculate the support count of every elements that occur in database. Individual mapper reads one piece of database. This phase indirectly determines the elements vocabulary I, which is typically unidentified for a massive database. The outcome is kept in F-list.

Stage 3: Gathering Items: Partitioning completely the components of F-List to Q-gatherings. The rundown of gathering is named gathering list (G-list), where each gathering has an unmistakable gathering id (gid). According to F-rundown and G-list are both minor and the running time is O(I), this stage can complete on one machine in little moments.

Stage 4: Parallel FP-Growth: The crucial phase of PFP. In this stage, PFP works on single Map Reduce job, where the map instance and reduce instance do dissimilar vital purposes.

Mapper - Producing group dependent operations: Every mapper job takes a piece of database produced in phase 1. Beforehand it computes operations in the piece one at a time, it analyses the G-list. With the mapper procedure it yields one or extra key value pair, where every key is a group-id and its equivalent value is a produced group-dependent operation.

Reducer - FP-Growth on group-dependent pieces: When every mapper instance has completed their function, for every group-id, the Map Reduce infrastructure routinely collects all equivalent group-dependent operations to a piece of group-dependent operations. Every reducer job is allotted to process one or extra group-dependent piece one at a time. For every piece, the reducer job forms a native fp-tree and its restricted fp-trees iteratively. Throughout the iterative procedure, it might have output a determined sequence of elements.

Stage 5: Aggregating: Combining the outcomes produced in Phase 4 as our last outcome.

4.5 BPFP: Balanced Parallel FP-Growth

Zhou et al. [18] presented a balanced parallel FP-growth (BPFP) with Map Reduce. Assume an operation database and the least support value, BPFP requires 2 phases of Map Reduce to run FP-Growth in parallel. Associated with PFP, the two main variations are, first one is the combining phase has a stable approach, which splits the whole extracting job to a comparatively uniform subclass to increase parallelism. The another is it does not have the "Aggregating" phase, as its purpose is to determine altogether repeated elements.

Stages of BPFP:

Stage 1: Sharding: Sharding is distributing database to consecutive parts and keeping the parts on P different computers. When using Hadoop Map Reduce, we impartially require replicating the database into Hadoop Distributed File System. Hadoop will do the fragmentation.

Stage 2: Parallel Counting: This phase works on one Map Reduce task to calculate every element. One shard of database is given to each mapper as input. The Input is in the key and value format like (key, value = T_i), where T_i is a database operation. For every element, say a_jT_i, the mapper emits a key and value structure (key' = a_j, value' = 1). Once each mapper instance has completed their work, the Reducer calculate the summation of all the counts conforming to the similar key' and emits (key', sum (value')) pair. The outcome of this phase is the F-List, a list of repeated elements arranged by occurrence in downward manner.

Stage 3: Balanced Grouping: Balanced Grouping properly splits the elements in the F-List to Q groups, stabilizing work between entire groups. Such a grouping is to increase parallelism of the entire extracting procedure. Work of the entire FP-Growth equal to summation of works of sub-FP-Growth, which are FP-Growth on restricted sequence of element of each repeated elements. Balanced Grouping can be allocated into two stages. The initial stage is calculating load component, which is the volume of efforts of executing FP-Growth on restricted sequence of element of every repeated element. The next stage is properly splits entire load component to a numerous group.

Table. 1. Comparing references based on different parameters like automatic parallelization, load balancing, data mining efficiency, data distribution, scanning the database, storage.

References	Advantages	Disadvantages
Agrawal et al. [3]	Sequential mining on dataset, efficient.	It lacks in load balancing, need large database for storage, cannot be used on large database.
Tsay et al. [11]	It requires less storage space for database, no need to scan the database again and again for mining, parallel mining on database.	Lacks in automatic parallelization and load balancing.
Han et al. [15]	Distribution of data over data nodes, parallel mining.	Not efficient, require more time for mining.
Yu et al. [26]	Parallel mining on database, workload balance.	Lack in automatic parallelization and its expensive data mining.
Zhou et al. [18]	Parallel FP-Growth, use MapReduce programming model for large database.	Lacks in load balancing and its expensive for mining.

4.5.1 Mining Load Estimation

In the meantime, data presented in this stage is not sufficient for calculating exact load of every element; few predictions are required to be made. Two predictions are made in this step. Earliest one, amount of recursive repetitions for the duration of FP-Growth execution on restricted sequence of element of each element is predicted as load of FP-Growth on restricted sequence of element of each element. The additional one is that every element in F-List is predicted to be the size of the lengthiest repeated trail in the restricted sequence of element. The amount of recursive repetitions is exponentially relative with lengthiest repeated trail in the restricted sequence of element. Predicted load of element i, say T_i, can be calculated by locality of element i in F-List, say L_i, as $T_i' = \log L_i$.

4.5.2 Balanced Partition

Primarily, entire elements in F-List are organized by load in downward manner, making L-List. Then and there, the anterior Q elements create the primary Q groups, single element for each group. Load of every group is adjusted with the load of the element it holds. After that, reiterate the subsequent two stages till entire elements in the L-List are collected:

1. Augment the resulting non-grouped element in L-List to the group with the least load.
2. Rises the load of that group by the load of the novel element.

5. CONCLUSIONS

Researchers in various fields have denoted to the state of the art of the feature extraction [23 - 25]. As a result, it is a stimulating job to make available a complete summary of the feature extraction approaches in this paper. This paper is an effort to offer a sensible review, from a researcher's viewpoint, on the feature extraction methods designed in recent times.

We have discussed about various repeated sequence of elements extraction algorithms. In repeated sequence of element extraction, it is recognized that few features of the datasets, for example the closeness, can disturb the selection of extraction algorithm. This paper gives the overview of algorithms designed for parallel extraction of repeated elements, and also focuses on each of the algorithms advantages, disadvantages and limitations for finding sequence of elements among huge database systems.

References

[1] Piateski, Gregory, and William Frawley. *Knowledge discovery in databases*. MIT press, 1991.

[2] Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases." *Acm sigmod record*. Vol. 22. No. 2. ACM, 1993.

[3] Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994.

[4] Park, Jong Soo, Ming-Syan Chen, and Philip S. Yu. *An effective hash-based algorithm for mining association rules*. Vol. 24. No. 2. ACM, 1995.

[5] Zaki, Mohammed J. "Parallel and distributed association mining: A survey." *IEEE concurrency* 7.4 (1999): 14-25.

[6] Pramudiono, Iko, and Masaru Kitsuregawa. "FP-tax: Tree structure based generalized association rule mining." *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. ACM, 2004.

[7] Agrawal, Rakesh, and Ramakrishnan Srikant. "Mining sequential patterns." *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*. IEEE, 1995.

[8] Chen, Ming-Syan, Jong Soo Park, and Philip S. Yu. "Data mining for path traversal patterns in a web environment." *Distributed Computing Systems, 1996., Proceedings of the 16th International Conference on*. IEEE, 1996.

[9] Han, Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." *ACM sigmod record*. Vol. 29. No. 2. ACM, 2000.

[10] Wang, Ke, et al. "Top down fp-growth for association rule mining." *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, Berlin, Heidelberg, 2002.

[11] Tsay, Yuh-Jiuan, Tain-Jung Hsu, and Jing-Rung Yu. "FIUT: A new method for mining frequent itemsets." *Information Sciences* 179.11 (2009): 1724-1737.

[12] Agrawal, Rakesh, and John C. Shafer. "Parallel mining of association rules." *IEEE Transactions on knowledge and Data Engineering* 8.6 (1996): 962-969.

[13] Cheung, David W., and Yongqiao Xiao. "Effect of data skewness in parallel mining of association rules." *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, Berlin, Heidelberg, 1998.

[14] Shintani, Takahiko, and Masaru Kitsuregawa. "Hash based parallel algorithms for mining association rules." *Parallel and Distributed Information Systems, 1996., Fourth International Conference on*. IEEE, 1996.

[15] Han, Eui-Hong, George Karypis, and Vipin Kumar. *Scalable parallel data mining for association rules*. Vol. 26. No. 2. ACM, 1997.

[16] Agrawal, Rakesh, and John C. Shafer. "Parallel mining of association rules." *IEEE Transactions on knowledge and Data Engineering* 8.6 (1996): 962-969.

[17] Li, Haoyuan, et al. "Pfp: parallel fp-growth for query recommendation." *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 2008.

[18] Zhou, Le, et al. "Balanced parallel fp-growth with mapreduce." *Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on*. IEEE, 2010.

[19] Kovacs, Ferenc, and János Illés. "Frequent itemset mining on hadoop." *Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on*. IEEE, 2013.

[20] Oruganti, Shravanth, Qin Ding, and Nasseh Tabrizi. "Exploring Hadoop as a platform for distributed association rule mining." *FUTURE COMPUTING 2013 the Fifth International Conference on Future Computational Technologies and Applications*. 2013.

[21] Lin, Ming-Yen, Pei-Yu Lee, and Sue-Chen Hsueh. "Apriori-based frequent itemset mining algorithms on MapReduce." *Proceedings of the 6th international conference on ubiquitous information management and communication*. ACM, 2012.

[22] Li, Ning, et al. "Parallel implementation of apriori algorithm based on mapreduce." *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*. IEEE, 2012.

[23] Li, Lingjuan, and Min Zhang. "The strategy of mining association rule based on cloud computing." *Business Computing and Global Informatization (BCGIN), 2011 International Conference on*. IEEE, 2011.

[24] Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From data mining to knowledge discovery in databases." *AI magazine* 17.3 (1996): 37.

[25] Piatetsky-Shapiro, Gregory. "Discovery, analysis, and presentation of strong rules." *Knowledge discovery in databases* (1991): 229-238.

[26] Yu, Kun-Ming, et al. "A load-balanced distributed parallel mining algorithm." *Expert Systems with Applications* 37.3 (2010): 2459-2464.