DESIGN AND IMPLEMENTATION OF GLOBAL FREQUENT PATTERN MINING ALGORITHM USING MAP REDUCE

Enrolment No.	122211
Name of Student	Lucky Rajpoot
Name of supervisor	Dr. Pardeep Kumar

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

juit

विद्या तत्व ज्योतिसमः

MAY-2014

Submitted in partial fulfilment of the Degree of

Master of Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT, DIST. SOLAN, (H.P.), INDIA

Table of content

CERTIFICATE

This is to certify that the work titled "**DESIGN AND IMPLEMENTATION OF GLOBAL FREQUENT PATTERN MINING ALGORITHM USING MAP REDUCE**" submitted by "**LUCKY RAJPOOT**" in partial fulfilment for the award of degree of M. Tech of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor ……………………..

Name of Supervisor	Dr. Pardeep Kumar
Designation	Assistant Professor, Department of CSE and ICT
Date	……………………..

<u>ACKNOWLEDGEMENT</u>

I take this opportunity to offer my honour and profound greetings on the success of my M.Tech research work to my supervisor Dr. Pardeep Kumar, Assistant Professor-Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat, for his generous help and great support. I felt motivated from his incredible guidance and this research work would not have been possible without his invaluable contribution.

I would also like to express my heartfelt thanks to my family and friends who have been my inspiration. Their ideas and motivation have been my driving force throughout my research tenure. Lastly, I would like to thank my friend Anubhav who has helped me in simulating my research work.

Signature of the student	……………………..
Name of Student	……………………..
Date	……………………..

SUMMARY

With the proliferation of computing in our daily lives, more and more amount of data is being collected. Along with the traditional computing techniques, novel technologies like pervasive computing, cyber physical systems and Internet of Things (IoT) have come out of their incubation period and together they are generating highly complex and unstructured data which is sometimes also termed as Big Data. In order to take intelligent decisions and create applications that leverage from the treasure trove of information available we need to analyze and discover new facts and subtle yet important relationships. We need to mine the data to extract even the last bit of useful knowledge. In order to tackle this challenge we have developed a frequent item set mining algorithm that operate on real world data. Our proposed technique uses Map Reduce technique which is the key to exploit the inexhaustible and boundless potential of the omnipresent cloud. Preliminary evaluation and analysis indicates that the proposed approach has comparable, if not better, performance to other state of the art frequent item set mining algorithms.

Signature of Student	Signature of Supervisor
Name: Lucky Rajpoot	Name: Dr. Pardeep Kumar
Date:	Date:

LIST OF FIGURES

Chapter 1

Introduction

1.1 Data mining and knowledge discovery [1]:

Data mining is the process of extracting interesting (implicit, nontrivial, earlier unknown and potentially useful) information or patterns from large data repositories such as: data warehouses, relational database, XML repository etc. Data mining is one of the core processes of Knowledge Discovery in Database (KDD) which is the extracted knowledge from the large datasets used for generating patterns and decision making.

There three main tasks in data mining:

- **Pre-processing** - This is the process which is executed before data mining techniques are applied to the right data. The pre-processing includes data cleaning, integration, selection and transformation.
- **Data mining process** – In this process hidden knowledge is produced by applying different algorithm.
- **Post-processing** – After having the mining results, the mining result is evaluated according to user's requirements and domain knowledge.

The data sources may come from different databases, which may have some inconsistence and duplicate data. So first we need to clean and integrate the databases by removing noises from the data source. The second task is to select related data from the integrated resources and transform them into a format that is ready to be mined. After selection of relevant data, the database in which our data mining techniques are to applied will be much smaller, consequently the whole process will be more efficient.

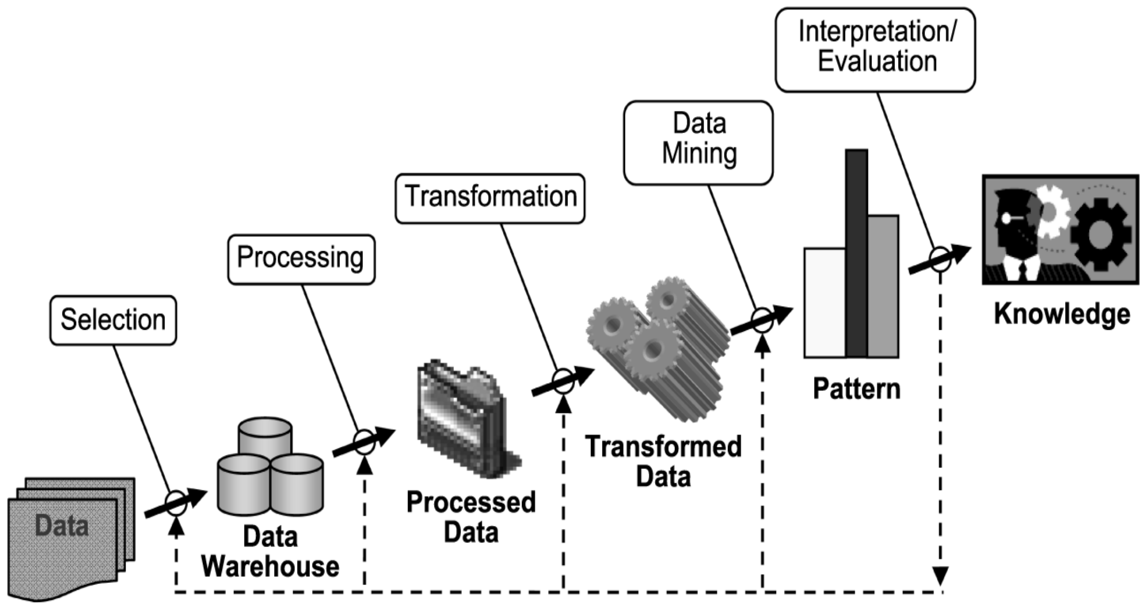


Fig 1.1 Steps of KDD process

Fig 1.1 shows the KDD process of extracting knowledge from data. KDD has evolved from interaction and cooperation among such different fields as machine learning, pattern recognition, database, statistics, artificial Intelligence, knowledge representation, and knowledge acquisition for intelligent systems. The main idea in KDD is to discover a high level knowledge (abstract knowledge) from lower levels of relatively raw data, or to discover a higher level of interpretation and abstraction than those previously known.

1.2 Frequent pattern mining[2,3]:

Frequent pattern mining is the extraction of those items from the databases whose frequency is more than a specified threshold or support. Frequent pattern mining is a focused theme in researches of data mining over a decade. Large number of researches has been dedicated to it and remarkable progress is achieved. Frequent pattern mining plays a necessary role in several data mining tasks, like mining associations, sequential patterns, classification, and clustering, max-patterns and frequent closed patterns.

The discovery of frequent patterns is one of the most prominent research problems in data mining. Now a day in each and every field data is generated with a very high speed. To extract the useful hidden information from the large amount of databases we use data mining. In transaction databases there are frequent patterns which are important problem in data mining. To solve this problem this problem many researches are going on. The databases of each organisation are increasing at rapid rate, with the increment in size the required memory and computation size also increases. User behaviour also becomes more complex due to the increase in number of items. Many algorithms are applied to discover frequent patterns from large databases.

Discovering all frequent itemset from large databases is quite challenging task as the search space required is exponential to the number of items to the database. The output is limited to reasonable space because of the support threshold. Support counting can be a tough problem in the large databases which contains millions of transactions. A very large number of frequent itemsets and rules are generated from a frequent pattern mining. It reduces not only efficiency but also effectiveness of mining.

Frequent pattern mining of transaction databases is one of the important problems in data mining as the size of databases is continuously increasing because of which the computation time as well the memory required is also increasing. Many researches are going to solve this problem. The best way to solve it, apply parallel and distributed computing techniques to discover the frequent patterns from massive databases

In many of the data mining tasks, such as association rule, clusters and classifiers, correlation sequences and episodes frequent set plays an essential role. Among these association rule is one of the most popular problem of data mining. Many algorithms of finding frequent pattern have been discovered on static databases as well as streaming datasets.

1.2.1 Problem Statement[2]

Let I be a set of items. A set $X = \{i_1, . . . , i_k\} \subseteq I$ is called an itemset, or a k-itemset if it contains k items.

A transaction over I is a couple $T = (tid, I)$ where tid is the transaction identifier and I is an itemset. A transaction $T = (tid, I)$ is said to support an itemset $X \subseteq I$, if $X \subseteq I$.

A transaction database D over I is a set of transactions over *I*. We omit *I* whenever it is clear from the context.

The cover of an itemset X in D consists of the set of transaction identifiers of transactions in D that support X:

$$cover(X, D) := \{tid \mid (tid, I) \in D, X \subseteq I\} \tag{1.1}$$

The support of an itemset X in D is the number of transactions in the cover of X in D:

$$support(X, D) := |cover(X, D)|. \tag{1.2}$$

The frequency of an itemset X in D is the probability of X occurring in a transaction T ∈ D:

$$frequency(X, D) := P(X) = \frac{support(X, D)}{|D|} \tag{1.3}$$

Note that |D| = support({}, D). We omit D whenever it is clear from the context.

An itemset is called frequent if its support is no less than a given absolute minimal *support threshold* $\sigma_{abs,}$ with $0 \leq \sigma_{abs} \leq |D|$. When working with frequencies of itemsets instead of their supports, we use a relative minimal frequency threshold σrel, with $0 \leq \sigma_{rel} \leq 1$. Obviously, $\sigma_{abs} = \sigma_{rel} \cdot |D|$. In this thesis, we will only work with the absolute minimal support threshold for itemsets and omit the subscript abs unless explicitly stated otherwise.

Definition: Let D be a transaction database over a set of items I, and σ a minimal support threshold. The collection of frequent itemsets in D with respect to σ is denoted by

$$F(D, \sigma) := \{X \subseteq I \mid support(X, D) \geq \sigma\}, \tag{1.4}$$

or simply F if D and σ are clear from the context.

Problem: (Itemset Mining) Given a set of items I, a transaction database D over I, and minimal support threshold σ, find *F(D, σ).*

1.3 Data mining task based on Frequent set

1.3.1 Association rules[4,5]

Formally, an association rule is an implication relation in the form $X \rightarrow Y$ between two disjunctive sets of items X and Y. A typical example of an association rule on "market basket data" is that "80% of customers who purchase bread also purchase butter ". Each rule has two quality measurements, support and confidence.

The rule $X \rightarrow Y$ has confidence c if c% of transactions in the set of transactions D that contains X also contains Y. The rule has a support S in the transaction set D if S% of transactions in D contain $X \cup Y$. The problem of mining association rules is to find all association rules that have a support and a confidence exceeding the user-specified threshold of minimum support (called MinSup) and threshold of minimum confidence (called MinConf) respectively.

The support of an itemset X denoted by S (X) is the ratio of the number of transactions that contains the itemset X ($|T_x|$) to the total number of transactions (|D|). S(X) is defined by the following formula:

$$S(X) = \frac{|T_X|}{|D|} \qquad (1.5)$$

The support of an association rule denoted by $S(X \Longrightarrow Y)$ is the ratio of the number of transactions containing both X and Y ($|T_x \cap T_y|$) to the total number of transactions, |D|. If the support of an association rule is 20% this means that 20% of the analyzed transactions contain $X \cup Y$ is defined by the following formula:

$$S(X \Longrightarrow Y) = \frac{|T_X \cap T_Y|}{|D|} \qquad (1.6)$$

The confidence of an association rule indicates the degree of correlation between x and y in the database. It is used as a measure of a rule's strength. The confidence of an association rule $X \rightarrow Y$ denoted by C($X \rightarrow Y$) is the ratio of the number of transactions that contain X ∪ Y (S($X \rightarrow Y$)) to the number of transactions that contain X (S(X)). Consequently, if we say an association rule has a confidence of 87%, it means that 87% of the transactions containing X also contain Y C($X \rightarrow Y$) is defined by the following formula:

$$C(X \Longrightarrow Y) = \frac{S(X \Longrightarrow Y)}{S(X)} = \frac{|T_X \cap T_Y|}{|T_X|} \qquad (1.7)$$

Association rule mining is described as a two-step process as follows:

Step 1: extraction of all frequent itemsets.

Step 2: Strong association rules extractions from the obtained frequent itemsets.

In general, association rules are considered interesting (frequent) if they satisfy both a minimum support threshold and a minimum confidence threshold defined by users or domain experts.

If the support and the confidence of an association rule $X \rightarrow Y$ is greater than or equal to the user specified minimum support, minsupp and minimum confidence value, minconf this rule is said to be frequent (interesting). A frequent rule is characterized by the following properties:

$$S(X \Longrightarrow Y) \geq minsupp \qquad (1.8)$$

And

$$C(X \Longrightarrow Y) \geq minconf \qquad (1.9)$$

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories. Association rules are widely used in various areas such as telecommunication networks, market and risk management, inventory control etc.

Generally, an association rules mining algorithm contains the following steps:

1. The set of candidate k-itemsets is generated by 1-extensions of the large (k -1)-itemsets generated in the previous iteration.

2. Supports for the candidate k-itemsets are generated by a pass over the database.

3. Itemsets that do not have the minimum support are discarded and the remaining itemsets are called large k-itemsets.

This process is repeated until no more large itemsets are found.

Suppose the support of an item is 0.1%, it means only 0.1 percent of the transaction contain purchasing of this item. Confidence of an association rule is defined as the percentage/fraction of the number of transactions that contain $X \cup Y$ to the total number of records that contain X. Confidence is a measure of strength of the association rules, suppose the confidence of the association rule $X \Rightarrow Y$ is 80%, it means that 80% of the transactions that contain X also contain Y together.

1.3.2 Clusters[6]

Now, how the frequent itemsets help in forming cluster? First we need to know what are clusters? Clusters are sets of something which have same characteristics or properties.

Clustering is the process of grouping physical or abstract objects into similar ones.

It is the unstructured process of learning in which the number of classes is not prior defined. Each and every transaction is clustered on the basis of characteristics and features.

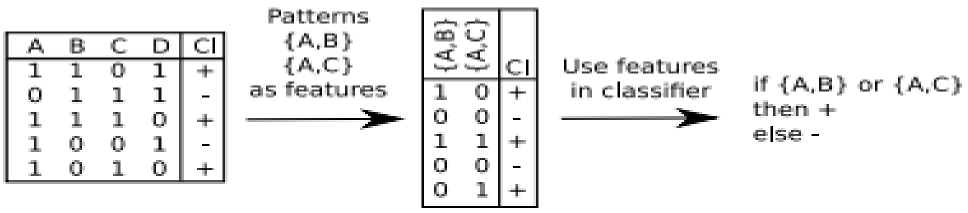
The frequent itemset sets are maintained via clusters as the centre point of each cluster is the transaction which is most frequent and every transaction is categorised by comparing that transaction with the cluster centre. In this way each and every transaction is classified.

1.3.3 Classifiers[7,8]

Classifiers are just like clusters only, the only difference is that it is structured learning i.e. the number of classes are prior known. In this we randomly select the transaction whose Euclidean distance is high as a centre point. We select as many transactions as much we need the classes as a centre point. Now to classify the transaction into the classes we follow the same process as in clustering.

The idea to use frequent patterns is to define features or can be used as rules; classification models which make use of these features or rules may be more accurate or simpler to understand. Frequent pattern mining can be considered a propositionalization approach which enables the use of propositional data mining and machine learning algorithms.

The main idea of pattern-based classification is that patterns define new features, which can be used in a classification model. A simple example is provided in the figure below:



A	B	C	D	Cl
1	1	0	1	+
0	1	1	1	-
1	1	1	0	+
1	0	0	1	-
1	0	1	0	+

{A,B}	{A,C}	Cl
1	0	+
0	0	-
1	1	+
0	0	-
0	1	+

Fig 1.2: Example of classifying dataset using pattern extraction

Essentially, a pattern is a regularity that is observed in a number of examples, in this example {A, B} is a pattern that occurs in the first example and third example. Whether this regularity is present or not in an example can be seen as a feature of each example. A prediction can be based on this, for instance, if an example includes items A and B we may predict the example to be positive.

1.3.4 Correlation Sequences[9]

Correlation is up to how much one transaction is related to other. This can be calculated by using Pearson correlation or Euclidean distance. Most of the database uses correlation as a basic function for many analytic tasks. If we are having similar queries then it is required to set the length and there is no proper way to define the proper length based on the needs of different application.

Now a days in many application (such as network analysis, sensor network analysis, financial data analysis and image processing) sequence data can be found. To analyse data without prior knowledge of the query length Longest- lasting correlated sub sequences may be useful. If we take the example of stock analysis, a stock analysis wants to find stock whose price variance is similar to its competitor then he can find its answer easily by subsequence matching.

1.3.5 Frequent Episode[10]

Episode can be defined as a partially ordered collection of events occurring together. Informally, an episode is a partially ordered collection of events occurring together. These ordered collections of events may carry useful information regarding correlations among events types. Episodes can be described by directed acyclic graphs. An *episode* is said to occur in an event sequence if there are events of appropriate event types in the data sequence with a time ordering that conforms to the partial order specified by the episode.

For instance consider the episodes α, β, γ.

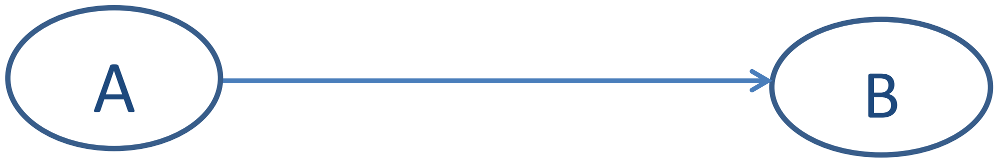


Fig 1.3.: Episode α

Episode α is a serial episode: this episode occurs in a sequence only if there are event of types A and B that occur in this order in a sequence. This means that the event B is followed by the event A in the sequence.

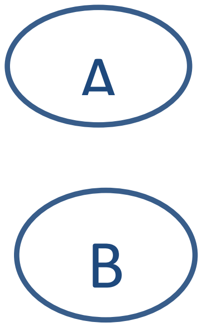


Fig 1.4: Episode β

Episode β is a *parallel episode*: no constraints on the relative order of *A* and *B* are given. These events are independent of each other and occur independently.

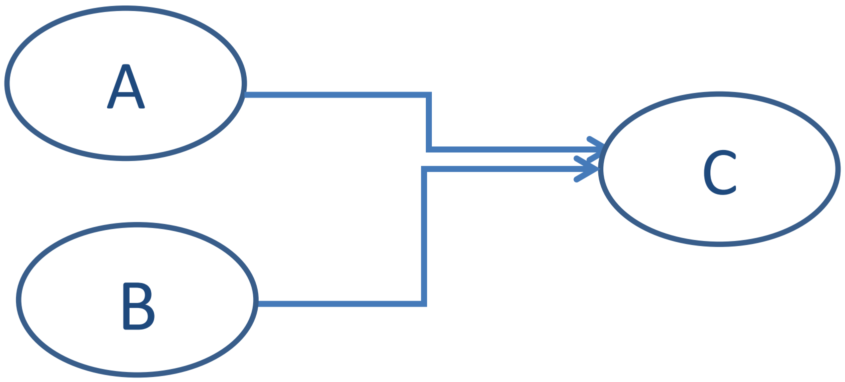


Fig 1.5: Episode γ

Episode γ is non-serial and non-parallel episode: In this the sequence of occurrence of events A and B. they can occur in parallel or may occur one after another but the occurrence of C will depend on event A and B.

A frequent episode is one whose frequency exceeds a user-specified threshold and based on the same general idea as the Apriori algorithm. This method exploits the fact that a necessary condition for an N-node episode to be frequent is that all its (N-1) nodes sub-episodes should be frequent.

1.4 Types of Data[11]

I. **Static Data:** this data is passive and randomly accessed or it is a static data which has been generated in past. In this training data is selected randomly and the result accuracy does not depend on response time, the data is persistent and update speed is minimal and respond in passive mode. In this the querying process is one time.

II. **Stream Data:** it contains transient data relation that is the data in the dataset is temporary and changes continuously with time; in this the process of querying is continuous. The data is continuously updated. The performance and result accuracy is influenced with time. The data is accessed sequentially. The speed of updating is high therefore the response type should be active. It is of two types: online data and offline data. The streaming dataset mining has lead to study frequent pattern mining on online data, which helps in many of the emerging applications, such as network traffic analysis, web log and click stream mining, business and stock market analysis, trend analysis and fraud detection in telecommunications data and sensor network. With emergence of these applications, it is becoming difficult to conduct advanced analysis and data mining over fast-arriving and large data streams in order to capture interesting patterns, trends and exceptions.

1.5 Kinds of Frequent Itemsets Mining [12]

We can mine the complete set of frequent itemsets, based on the completeness of patterns to be mined: we can distinguish the following types of frequent itemset mining, given a minimum support threshold:

1.5.1 Frequent Closed: if we have an itemset *X* then the closure of *X* is the set of all items that appear in all transaction where *X* appears in a database *r*. Mathematically it can be given as:

$$\boldsymbol{cl(x)} = \cap \{\boldsymbol{t} \in \boldsymbol{r} | \boldsymbol{X} \subseteq \boldsymbol{t}\} \tag{1.10}$$

If *cl(X) =X* then *X* is said to be closed.

Closed frequent itemset can be find by first identifying all frequent itemsets then from this group find those that are closed by checking to see if there exists a superset that has the same support as the frequent itemset, if there is, the itemset is disqualified, but if none can be found, the itemset is closed.

1.5.2 Maximal Frequent Itemsets: An itemset X is a maximal frequent itemset (or max-itemset) in set S if X is frequent, and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in S.
first examine the frequent itemsets that appear at the border between the infrequent and frequent itemsets then Identify all of its immediate supersets. If none of the immediate supersets are frequent, the itemset is maximal frequent.

1.5.3 Constrained Frequent Itemset: An itemset X is a constrained frequent itemset in set S if X satisfy a set of user-defined constraints.

1.5.4 Approximate Frequent Itemset: An itemset X is an approximate frequent itemset in set S if X derive only approximate support counts for the mined frequent itemsets.

1.5.5 Near-match Frequent Itemsets: An itemset X is a near-match frequent itemset if X tally the support count of the near or almost matching itemsets.

1.5.6 Top-k Frequent Itemset: An itemset X is a top-k frequent itemset in set S if X is the k most frequent itemset for a user-specified value, k

1.5.7 Frequent Free: in a given dataset r, if there is no exact rule of the form *X1* $\rightarrow$ *X2 where X1* and *X2* are distinct subsets of *X* then an itemset X is said to be free. We can find weather an itemset is free or not efficiently by the following property:

$$\boldsymbol{X \text{ is free} \Longleftrightarrow \forall x \in X, sup(X) < sup(X - x)} \quad (1.11)$$

1.5.8 Frequent Essential Itemsets: An itemset X is said to be essential if there is no disjunctive rule of the form $A_1 \rightarrow A_2 \vee \ldots\ldots\vee A_k$ where $(A_i)_{i=1..k}$ are distinct elements in X. As for free sets, they can be efficiently tested exploiting the following property:

$$\boldsymbol{X\ is\ essential\ \Leftrightarrow Vx\ \in X, sup_{dij}(X - x)} \quad (1.12)$$

Where $sup_{dij}(X) = \{t \in r | t \cap X \neq \phi\}$

1.6 Real Time Frequent Mining[3,13]

Real time mining is basically the mining which is done on continuous data to keep them updated. Its importance is increasing because of thrust in many business applications recommender system, e-commerce and supply chain management and group decision support systems. A embarrassment of economical algorithms are planned until date

If we have the dense data sets, the performance of the algorithms based on the real dataset degrades significantly and these are the algorithms which fulfill the real world requirement and are suited to respond in real-time.

Business intelligence is playing an essential role in achieving business goal such as profitability, efficiency, customer preservation and market incursion. In most of the cases of frequent pattern mining we use historic data. Now the thing is that, if the historic data can help us in making good decision then how real time mining can make the decision process better. By using up to date information we can get rid of delays and speed up in the competitive environment. In various areas real-time decision making is important some of them are real-time supply chain management system, real-time customer relationship management, real-time recommender systems, real-time stock management and vendor inventory, real-time enterprise risk and vulnerability management, real-time operational management in which critical real-time information is required such application is in mission, airline industry, fraud detection, real time negotiation and many other areas like real-time dynamic pricing and discount offering to customers in real-time.

In the present era we need to be up to date and need to know the present demand, for that we need to analyze the activities regularly and take the decisions by analyzing the patterns generated from that set of data. Analyzing real time data (data of certain interval) can help in doing so. For having good patterns we have to maintain the past history, means on the basis of past and present we can take decision for future. This only concept we are using in our approach and through an example we will be able to know how we will find the frequent itemsets and how the approach is beneficial for the business analysis and maintain the stocks.

For knowledge discovery we need to have patterns from the process of data mining. The large databases in business in which we have transaction records is of great interest and we need to extract patterns from them. Frequent pattern mining is of great use for discovering patterns from the set transaction. Frequent pattern mining is quite complex and also the search space required for finding all frequent items is huge. Many of the algorithms have been proposed to make the search fast and accurate.

1.7 Applications of Frequent Itemset Mining[14]:

I. **Web log and click-stream mining**: in the web browser each and every click is notified and according to your clicks it will analyze your behavior. If you have noticed then in youtube it shows the recommended videos, all those videos are the result of the previous clicks that you have made. Or if you visit any ecommerce site then it also show some recommended things. This all are done by analyzing the behavior of the user or the previous result like most of the people purchase conditioner with shampoo. Like this way they can manage the site and can improve their profit and can improve the customer relationship.

II. **Network traffic analysis:** many researches are going to examine the structure and dynamics of network on the internet. This can be done by using packet analysis and network flow data. Traffic analysis is required for network management and security task. It is quite difficult task as it is difficult to manage large volume of data and to inspect the packets for the security and privacy concern. The security is provided by analyzing network and for that frequent pattern mining can help to extract those IP's which seems to be threatening for the network. By doing so we can find the unwanted IP's and can block them so that they cannot harm the organization.

III. **Trend analysis:** trend analysis is basically a practice of collecting information and attempt to spot a pattern. Future events can be predicted by using the collected information. Uncertain events which is happened in past can also be estimated by it such as based on data such as the average years which other known kings reigned and what is the frequency of a certain event during some period of time.

IV. **Fraud detection in telecommunications data:** for having fraud detection the data must be cleaned, relevant, adequate and available and there must be well defined tool and data mining process to detect frauds. In the competitive world fraud is a critical problem as there can be some procedure in which a company can face fraud. For that there should be proper mechanism to detect, control and automate fraud. We are having the set of transaction and the telecommunication data. By analyzing the telecommunication data we are able to find the fraud. We have the machine learning process which help in knowing what kind of activity can give rise to any fraud. Frequent mining can be one of the data mining algorithm which can be used in training data and can help in preventing the organization from fraud.

V. **E-business and stock market analysis[14]:** e-business and stock market is one of the most prominent application of the frequent pattern mining because it helps a lot in predicting the future and helps in taking better decision. For having good e-business they need to analyze user's behavior so that they can to better recommend the things other than he want. There should be proper synchronization between the things he is searching and the product that the site is recommending. Same in case of the stock market, by analyzing the past pattern we can predict the future.

VI. **Sensor networks:** sensor network is the network of sensor which senses the surroundings and stores the sensed information. That information needs to be analyzed to recover the unwanted events or to predict the future. Sensor networks are also used for training machines so that automatic decision can be taken. Making the system smart and expert we need to have knowledge base and the knowledge base is created through the data mining process which gives some pattern by analyzing the past data.. Suppose we have sensor which sense temperature we will be having the past record of the sensor and by that we can predict temperature of the next day.

VII. **Healthcare[15]:** frequent pattern mining can be used to find abnormity, and can help in managing medicines and can help doctors to make better decisions regarding treatment and can discover more knowledge about the genes and the past result can help them in giving better result. As in past many experiment is carried out that is if someone is facing this symptom then he suffer from this disease so by frequent pattern mining it is able to know that if a patient is having some disease than by knowing the symptoms we can predict that what can be the disease.

VIII. **Education [16]:** frequent pattern mining can help in alter the teaching method by analyzing the previous results of students. It also helps in managing projects which help in making the student focused. As by the past result we will do certain experiments that are by doing so the result is improving and so. The policy which gives the better result will be applied and can help in making the education better and guide the students in better way. Research is done instead of administration by applying these data mining processes.

IX. **Disaster prevention [17]:** by analyzing temperature, humidity and wind we can forecast any disaster and can reduce loss and casualties. The frequent pattern mining can help in doing so by having the past results it can predict the future that there can be any chance of natural calamity and people of that are can be made aware of the natural calamity which is predicted so that they can take better decision and get prepared for the thing going to happen in future.

1.8 Problem Formulation

Discovering all frequent itemset from large databases is quite challenging task as the search space required is exponential to the number of items to the database. The output is limited to reasonable space because of the support threshold. Support counting can be a tough problem in the large databases which contains millions of transactions. A very large number of frequent itemsets and rules are generated from a frequent pattern mining. It reduces not only efficiency but also effectiveness of mining.

Frequent pattern mining of transaction databases is one of the important problems in data mining as the size of databases is continuously increasing because of which the computation time as well the memory required is also increasing. Many researches are going to solve this problem. The best way to solve it, apply parallel and distributed computing techniques to discover the frequent patterns from massive databases.

1.9 Motivation

The motivation of searching frequent item set comes from the need to analyse the transactions of super market, in order to examine the customer behaviour of purchasing products. Frequent set defines that which set of products is buoyed together. For the business analyses it is often used i.e. what is the demand of customer?, what are the items that are often sold together?, which product should be launched together?, what compliments to what? All these answer are given by the frequent item analysis.

It helps in finding interesting patterns from the databases for example association rules, sequences, episodes, correlations, classifiers and clusters

The mining of association rules is one of the most popular problems of all these. The identification of sets of items, products, symptoms and characteristics, which often occur together in the given database, can be seen as one of the most basic tasks in Data Mining.

Chapter 2

Algorithms to Find Frequent Pattern from Static Data

In this section discuss some algorithms which are used for frequent data mining are discussed so that we may clearly understand that how the frequent itemsets are generated from the databases and how the result is used in decision making? We are having many algorithms, some are based on the traditional databases and some are for the massive databases. To find the frequent itemset from large databases we need to use those algorithms which are distributed and parallel in nature so that communication and computation complexity can be balanced. Now some of the algorithm is discussed as:

Frequent pattern mining studies are classified into two types: (1) the generate-and-test (Apriori-like) approach [18] and the frequent pattern growth approach [19].

2.1 Apriori Algorithm[18]

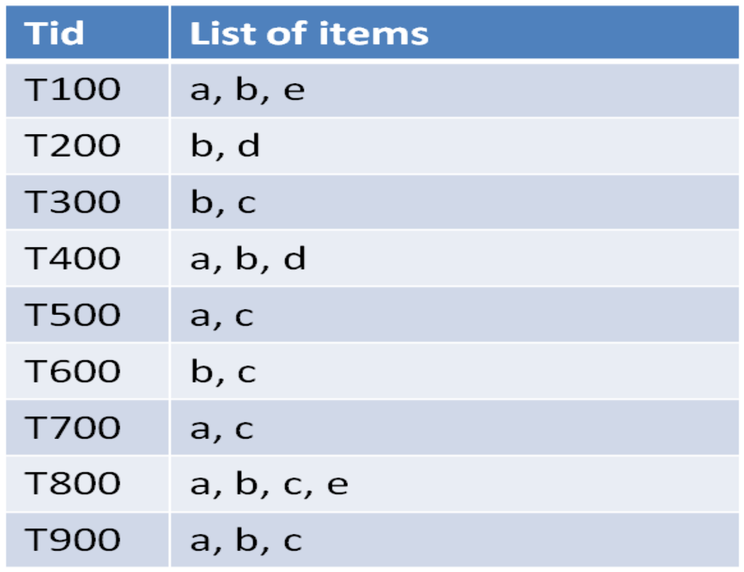
In Apriori-like method a candidate itemset of size (k+1) from frequent itemset of size k is generated iteratively. The database is scanned repetitively to test the frequency of candidate itemset. The shortcomings of the apriori are:

i. The cost of memory required is high to handle large number of generated candidate itemsets.
ii. Even though all the candidate set are loaded into the memory, the performance of accumulating frequency will be low as most of the time is spent in searching candidate set from a large pool of data.

The shortcoming of apriori has been solved by frequent pattern tree, in which transactions are compressed and stored. In this frequent patterns are generated without candidate generation, hence the scalability of the method is enhanced. Only two scans of database is required to construct FP-tree and the execution time greatly reduced. Fp-tree was also having some disadvantages, if the database is large then a complex FP-tree is generated which is difficult to store in memory. This problem is solved by further proposed algorithm for frequent itemset mining which are based on parallel and distribution mining technique.

2.1.1 Example of Apriori[2]:

The apriori algorithm can be better understood by example. Lets take an example and solve it by using the algorithm. Following are the steps to solve Apriori



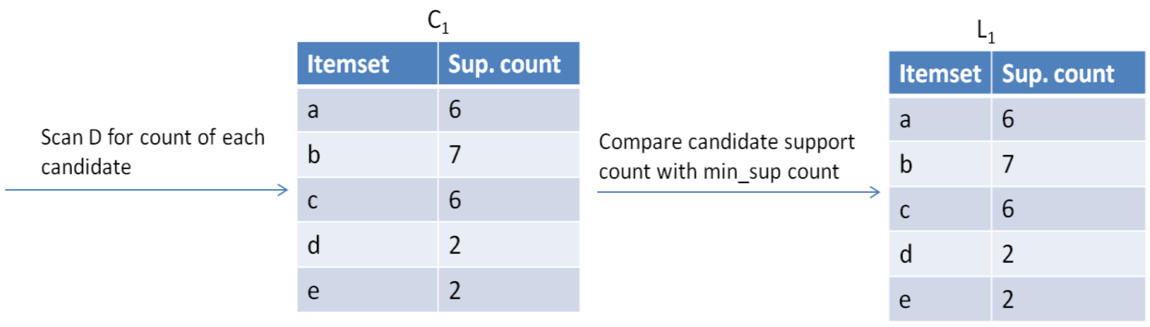
Tid	List of items
T100	a, b, e
T200	b, d
T300	b, c
T400	a, b, d
T500	a, c
T600	b, c
T700	a, c
T800	a, b, c, e
T900	a, b, c

Fig 2.1: Dataset to apply Apriori

There are 9 transaction in this database so, |D|=9. Now we will use the above dataset to find frequent itemset using apriori

1. In the iteration, we will find 1-itemsets C1, in which each item is a candidate set. In this step whole database is scanned to find items with their frequency (the number of occurrence of each item).

2. The minimum support is required so that we can prune the dataset. Suppose the min_sup=2. Find the set of frequent 1-itemset, L_1 by removing those itemset which have support less than min_sup.


Scan D for count of each candidate
Compare candidate support count with min_sup count

C_1

Itemset	Sup. count
a	6
b	7
c	6
d	2
e	2

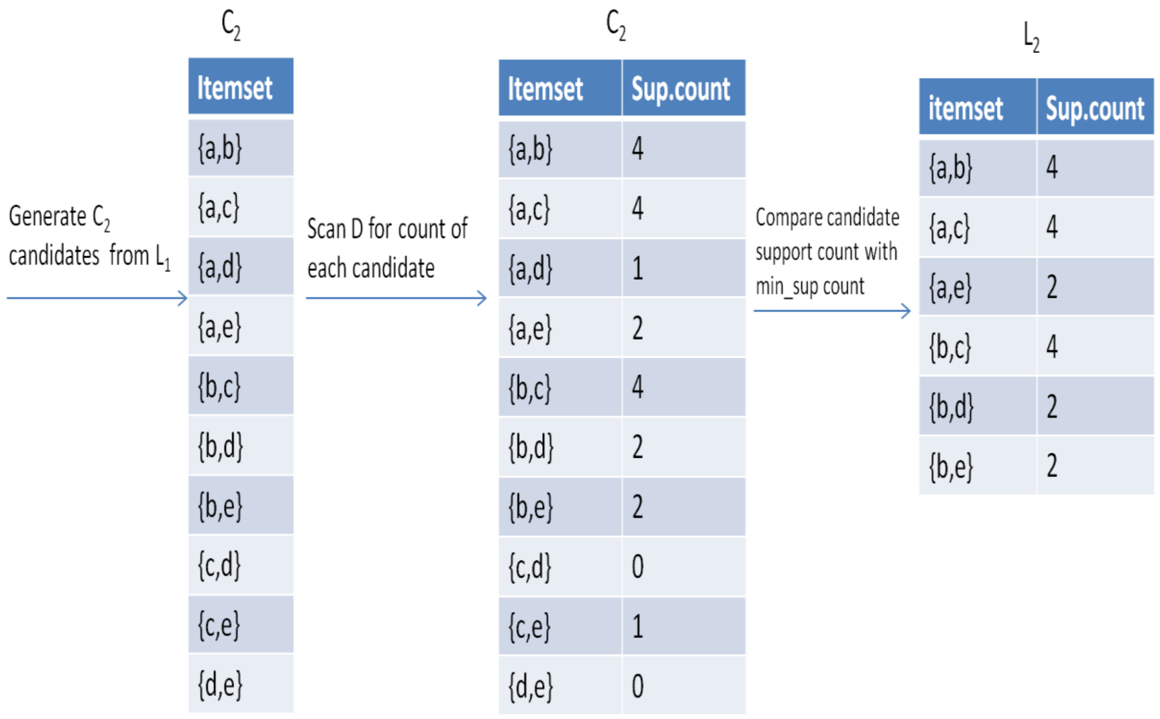
L_1

Itemset	Sup. count
a	6
b	7
c	6
d	2
e	2

Fig 2.2. Candidate itemset-1 with support count (left), frequent 1-itemset L_1 (right).

3. In this step, find the candidate set of 2-itemset and from that generate set of frequent 2-itemset L_2 and prune those itemset who have the sup.count less than min_sup.

4. In this step, find the candidate set of 2-itemset and from that generate set of frequent 2-itemset L_2 and prune those itemset who have the sup.count less than min_sup.


Generate C_2 candidates from L_1
Scan D for count of each candidate
Compare candidate support count with min_sup count

C_2

Itemset
{a,b}
{a,c}
{a,d}
{a,e}
{b,c}
{b,d}
{b,e}
{c,d}
{c,e}
{d,e}

C_2

Itemset	Sup.count
{a,b}	4
{a,c}	4
{a,d}	1
{a,e}	2
{b,c}	4
{b,d}	2
{b,e}	2
{c,d}	0
{c,e}	1
{d,e}	0

L_2

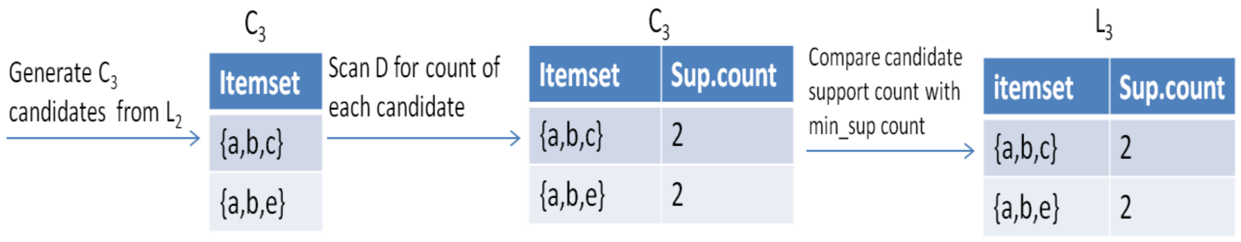
itemset	Sup.count
{a,b}	4
{a,c}	4
{a,e}	2
{b,c}	4
{b,d}	2
{b,e}	2

Fig 2.3. Candidate itemset-2 (left), Candidate itemset-2 with support count (middle), frequent 2-itemset L_1 (right).

5. Next, the support of each 2-itemset in dataset D is scanned as shown in middle table and the 2-itemset whose support is less than min_sup is pruned.

6. From the C_2 candidate set generate, L_2 which contains the itemset whose support is larger than the min_sup as shown in last table.

7. Do this process until you find the largest frequent itemset that is C_k and L_k


Generate C_3 candidates from L_2
C_3
Scan D for count of each candidate
C_3
Compare candidate support count with min_sup count
L_3

Itemset
{a,b,c}
{a,b,e}

Itemset	Sup.count
{a,b,c}	2
{a,b,e}	2

itemset	Sup.count
{a,b,c}	2
{a,b,e}	2

Fig 2.4. Candidate itemset-3 (left), Candidate itemset-3with support count (middle), frequent 3-itemset L_1 (right).

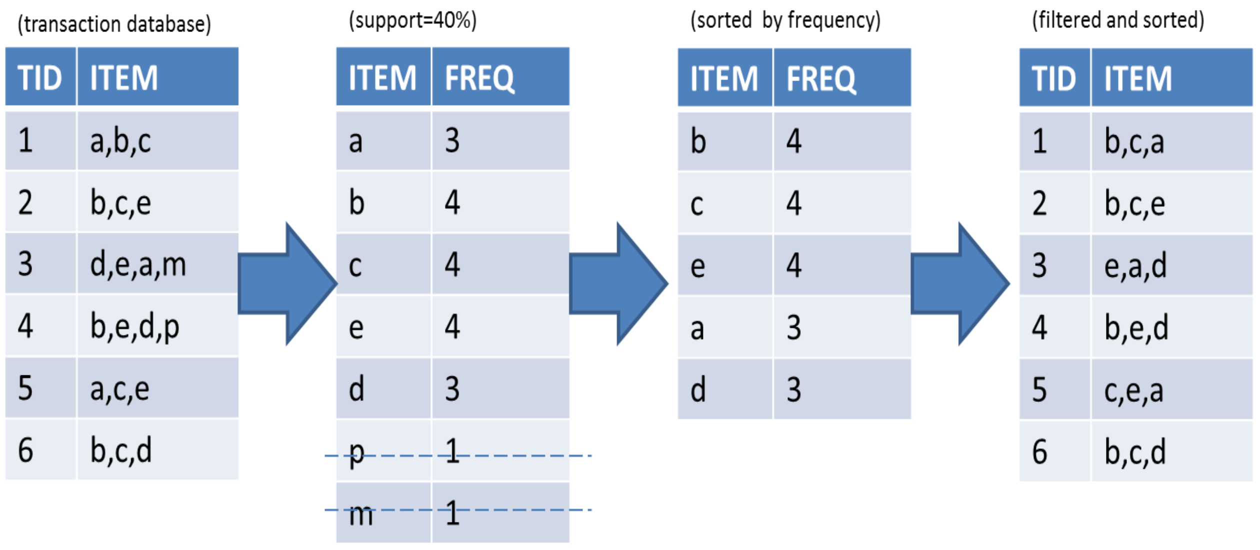
8. In this the largest frequent itemset is of size three which is shown in the above figure.

2.2 FP-growth Algorithm[19]

Han et al.[19] has proposed a data structure, FP-tree which is a tree-based data structure, and a mining algorithm, FP-growth, for discovering frequent patterns. In this algorithm the complete mining task require only two scan of database. In the first scan we calculate the frequency of each item and create a header table which records the item name and its corresponding frequency. The first node-link links to the first node with the same item name in the FP-tree. Items in the header table are sorted in descending order of their frequency. The items with support count less than min supp is filtered in second pass, and the residual items are sorted in descending arrange of the frequency values. The FP-tree is constructed using sorted items in each transaction. where item-name is the name of item for identification, count is the frequency .

For inserting transaction t in dataset D into FP-tree F, When two or more transactions share same prefix then their path can be overlapped. The number of times the path is overlapped the counter is increased. Node-links or pointers are maintained between nodes containing the same item and a singly linked list (dotted lines) is created. For each transaction in D recursively perform the insertion until each item is inserted into the FP-tree. The compression will be higher when more paths are overlapped and the FP-tree can be fitted in the memory. Once the FP-tree is constructed, FP-growth algorithm is used to discover the frequent patterns. An item in the header table is selected to construct the conditional FPtree by inserting all prefix paths of the item, which can be retrieved using the node-link structure in header table. The item name is called the conditional pattern base. Example of FP-Growth:

Step 1: In the first scan of whole database and the support of each item is calculated and stored in a table. All the elements are scanned with their support and the elements with less than min_sup is pruned. In the second scan of dataset all the transactions are filtered and sorted. Filter means that the entire item which has the support less than min_sup is removed from the dataset and sorted means all the transaction is sorted by the items in the transaction according to the descending order of the support of the items in the transaction.



(transaction database)

TID	ITEM
1	a,b,c
2	b,c,e
3	d,e,a,m
4	b,e,d,p
5	a,c,e
6	b,c,d

(support=40%)

ITEM	FREQ
a	3
b	4
c	4
e	4
d	3
~~p~~	~~1~~
~~m~~	~~1~~

(sorted by frequency)

ITEM	FREQ
b	4
c	4
e	4
a	3
d	3

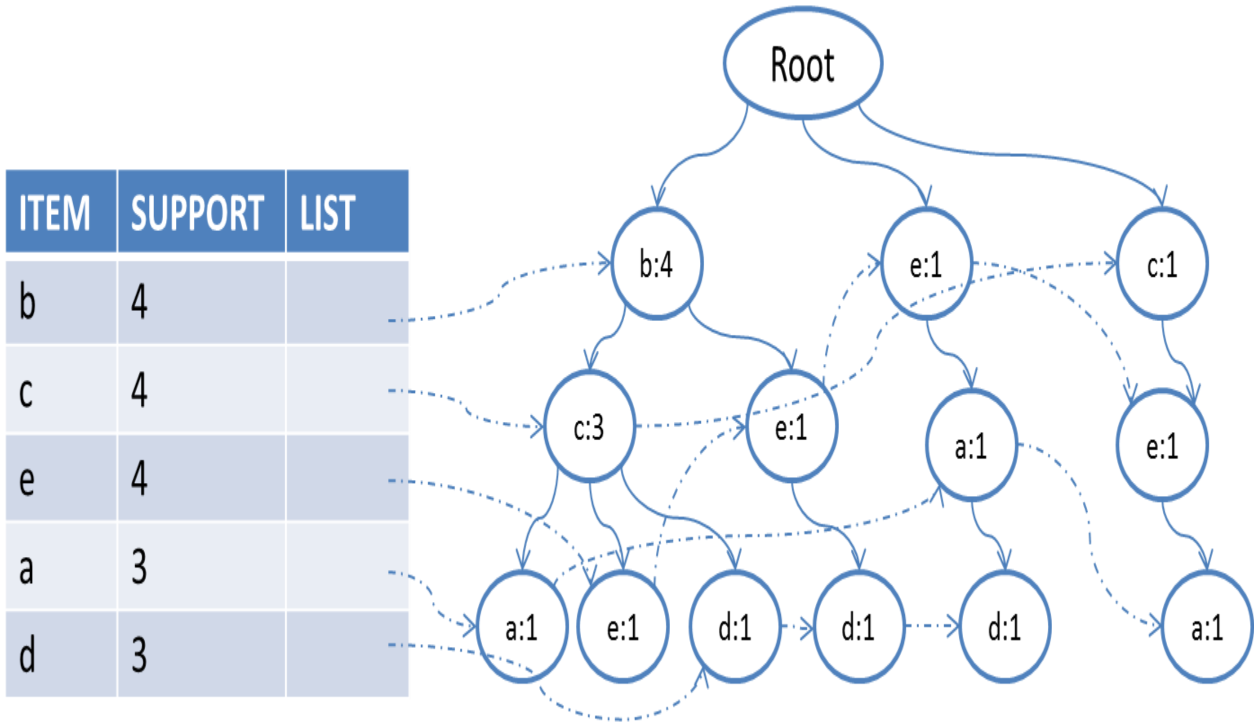
(filtered and sorted)

TID	ITEM
1	b,c,a
2	b,c,e
3	e,a,d
4	b,e,d
5	c,e,a
6	b,c,d

Fig 2.5: Transaction dataset (left), frequency of each item(2nd), items sorted by frequency(3rd), filtered and sorted traction database(right)

Step 2: once the whole dataset is scanned, filtered and sorted we will construct the FP-tree. Fp-tree is constructed by the following process:

1. FP-Growth reads 1 transaction at a time and maps it to a path
2. Fixed order is used, so paths can partly cover when transactions share items (when they have the same prefix).
    - This is case, counter are incremented
3. Pointers are maintain between nodes contain the same item, create singly linked lists (dotted lines)
    - The more paths that overlap, the higher the compression. FP-tree may fit in memory.
4. Frequent itemsets extracted from the FP-Tree.

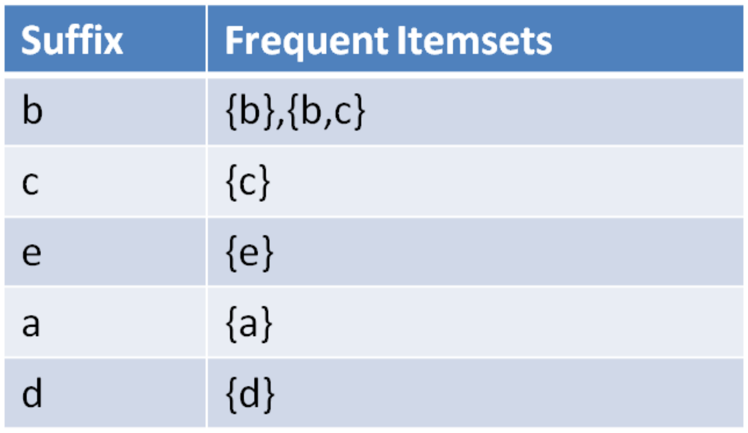


ITEM	SUPPORT	LIST
b	4	
c	4	
e	4	
a	3	
d	3	

Fig 2.6: FP-tree of the transaction database

Step 3: after constructing the FP-tee, find the frequent items from the FP-tree by the following steps:

1. FP-Growth extracts frequent itemsets from the FP-tree.
2. Bottom-up algorithm - from the leaves towards the root
3. Divide and conquer: first look for frequent itemsets ending in e, then de, etc. . . then d, then cd, etc. . .
4. First, extract prefix path sub-trees ending in an item(set). (hint: use the linked lists).



Suffix	Frequent Itemsets
b	{b},{b,c}
c	{c}
e	{e}
a	{a}
d	{d}

Fig 2.7: Frequent patterns of the transaction database

In this way we can find the frequent pattern by using FP-growh by first constructing the FP-tree and then find the frequent pattern from that FP-tree

2.3 ECLAT ALGORITHM[20,21]

Eclat uses the vertical database layout and uses the intersection based approach to compute the support of an itemset. In éclat for every item we create a list of transaction id's and store in which the item occurs, denoted by tidlist. For every itemset, its tidlist equals the intersectionof the tidlists of two of its subsets.

The Eclat algorithm is given as

Step 1. Find TidSet of each item i.e. the set of transaction the item belong (Data Scan)

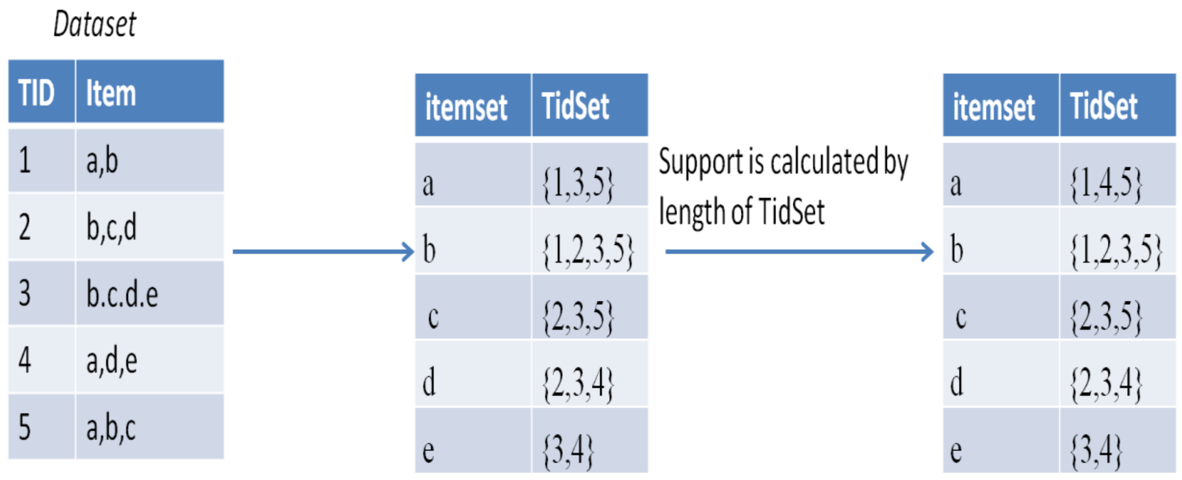
Step 2. TidSet of item is exactly the list of transactions containing item

Step 3. If $|TidSet|_{item}$ >min_sup then it is frequent else the item is infrequent.

Step 4. After finding the frequent item create i-candidate set with its TidSet

Step 5. Repeat the step to find k-candidate set which is frequent.

Eclat algorithm is a depth first search based algorithm. It uses a vertical database layout i.e. instead of explicitly listing all transactions; each item is stored together with its cover (also called tidlist) and uses the intersection based approach to compute the support of an itemset. It requires less space than apriori if itemsets are small in number. It is suitable for small datasets and requires less time for frequent pattern generation than apriori.



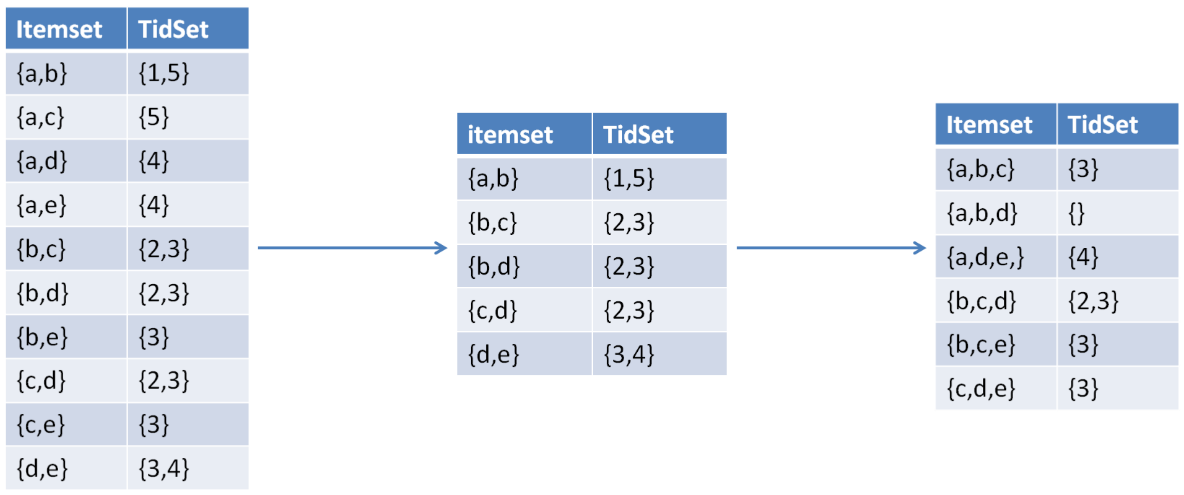
TID	Item
1	a,b
2	b,c,d
3	b.c.d.e
4	a,d,e
5	a,b,c

itemset	TidSet
a	{1,3,5}
b	{1,2,3,5}
c	{2,3,5}
d	{2,3,4}
e	{3,4}

itemset	TidSet
a	{1,4,5}
b	{1,2,3,5}
c	{2,3,5}
d	{2,3,4}
e	{3,4}

Fig 2.8: Transaction dataset(left), itemset belong to transaction ID(middle), frequent itemset-1(right)

In fig 2.8 we have a dataset on which we will apply éclat algorithm, first we will scan the database to find the transactions to which a item belong and list all the transation to which a item belong in TidSet. The support of an item is the length of TidSet, if the length of TidSet is greater than min_sup than the item or set is frequent. The rightmost table contains all the frequent items of length 1.



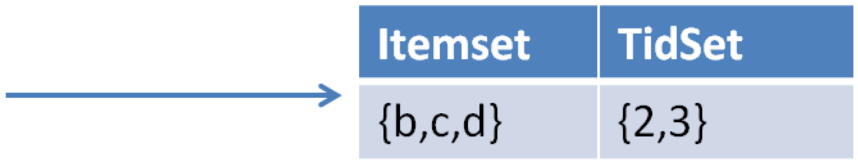
Itemset	TidSet
{a,b}	{1,5}
{a,c}	{5}
{a,d}	{4}
{a,e}	{4}
{b,c}	{2,3}
{b,d}	{2,3}
{b,e}	{3}
{c,d}	{2,3}
{c,e}	{3}
{d,e}	{3,4}

itemset	TidSet
{a,b}	{1,5}
{b,c}	{2,3}
{b,d}	{2,3}
{c,d}	{2,3}
{d,e}	{3,4}

Itemset	TidSet
{a,b,c}	{3}
{a,b,d}	{}
{a,d,e,}	{4}
{b,c,d}	{2,3}
{b,c,e}	{3}
{c,d,e}	{3}

Fig 2.9: Candidate itemset-2 with TidSet(left), frequent itemset-2middle), Candidate itemset-3 with TidSet(right),

Once we find the frequent items of length 1, we will make a set of length 2 from 1 length frequent itemset. Then we will find the transaction to which the pair belongs. The result is pruned that is all the infrequent itemsets are removed and with remaining set we create 3-candidate set.



Itemset	TidSet
{b,c,d}	{2,3}

Fig 2.10: Candidate itemset-3 with TidSet

Find all the 3-candidate set with its TidSet i.e. the set of transactions containing that set and at last find the itemset with TidSet> min_sup. In this way we can find the frequent itemset with length k.

This algorithm works as Apriori . It requires less space than apriori if itemsets are small in number .It is suitable for small datasets and requires less time for frequent pattern generation than apriori.

2.4 Relim Algorithm[22,23]

Relim is recursive elimination algorithm. Recursive elimination is associate degree rule for locating frequent itemsets that is powerfully impressed by the FP-growth rule and very kind of like the H-mine rule. It does its work while not prefix trees or the other sophisticated knowledge structures, process the transactions directly. Its main strength isn't its speed (although it's not slow, even outperforms Apriori and Eclat on some knowledge sets), however the simplicity of its structure. Essentially all the work is completed in one simple algorithmic perform, which may be written with comparatively few lines of code.

2.4.1 Preprocessing

Fig 2.11, shows an example dealings information on the left. The frequencies of the things during this information, sorted ascendingly, area unit shown within the table within the middle. If we tend to area unit given a user such that stripped support of three transactions, things f and g may be discarded. When doing therefore and sorting the things in every dealings ascendingly w.r.t. their frequencies we tend to get the reduced information.

```
a d f
c d e
b d
a b c d
b c
a b d
b d e
b c e g
c d f
a b d
```

g	1
f	2
e	3
a	4
c	5
b	7
d	8

```
a d
e c d
b d
a c b d
c b
a b d
e b d
e c b
c d
a b d
```

Fig 2.11: Transaction database (left), item frequencies (middle), and reduced transaction database with items in transactions sorted ascendingly w.r.t. their frequency (right).

2.4.2 Transaction Representation

Each dealing is pictured as straightforward arrays of item identifiers (which are whole number numbers). The initial dealings info is become a group of dealings lists, with one list for every item. These lists are kept in a very easy array, every component of that contains a support counter and a pointer to the top of the list. The list components themselves consist solely of a successor pointer and a pointer to (or rather into, see below) the dealings. The transactions are inserted one by one into this structure by merely victimization their leading item as associate index. However, the leading item is far from the dealings, that is, the pointer within the dealings list component points to the second item. Note that this doesn't lose any data because the 1st item is implicitly pictured by the list the dealings are in.

To illustrate this, Fig 2.11 shows, at the terribly prime, the illustration of the reduced information of the dataset. The primary list, such as the item e, contains the second, seventh and eight dealings, with the item e removed. The counter within the array component states the amount of transactions containing the corresponding item. It ought to be noted, as can become clear later, that this counter isn't invariably adequate the length of the associated list, though this can be the case for this first illustration of the information. Variations result from (shrunk) transactions that contain no alternative things and area unit therefore not diagrammatical within the list.

For implementations it's vital to notice that the delineated theme, with a pointer into the dealing so the leading item is skipped, will solely be applied in languages that yield pointer arithmetic. In languages during which this is often not possible (like, for example, Java) the things within the transactions is also sorted the opposite means spherical and a component counter, hold on within the list parts, could wont to specify the set of the things that's to be thought-about.

2.4.3 Recursive Processing

Recursive elimination works as follows: The array of lists that represents a (reduced) dealing information is “disassembled” by traversing it from left to right, process the transactions in a very list in a very algorithmic decision to seek out all frequent item sets that contain the item the list corresponds to. When an inventory has been processed recursively, its parts area unit either reassigned to the remaining lists or discarded (depending on the transactions they represent), and also the next list is worked on. Since all reassignments area unit created to lists that deceive the correct of the presently processed one, the list array can finally be empty (will contain solely empty lists).

Before a group action list is processed, however, its support counter is checked, and if it exceeds the user-specified minimum support, a frequent item set is rumoured, consisting of the item related to the list and a attainable prefix related to the entire list array.

One group action list is processed as follows: for every list part the leading item of its (shrunk) group action is retrieved associate degreed used as an index into the list array; then the part is value-added at the top of the corresponding list. In such a duty assignment, the leading item is additionally aloof from the group action, which may be enforced as a straightforward pointer increment (or as a counter decrement, see above). Additionally, a duplicate of the list part (with the leading item of the group action already removed by the pointer increment) is inserted within the same method into associate degree as initio empty second array of group action lists. (Note that solely the list part is derived, not the group action. each list components, the reassigned one and also the copy confer with constant group action.)

The process is illustrated for the foundation level of the formula in Fig 2.12, that shows the group action list illustration of the initial info at the terribly prime within the initiative all item sets containing the item e square measure found by process the left list the weather of this list square measure reassigned to the lists to the correct (grey list elements) and copies square measure inserted into a second list array (shown on the right). This second list array is then processed recursively, before continuing to following list, i.e., the one for item.

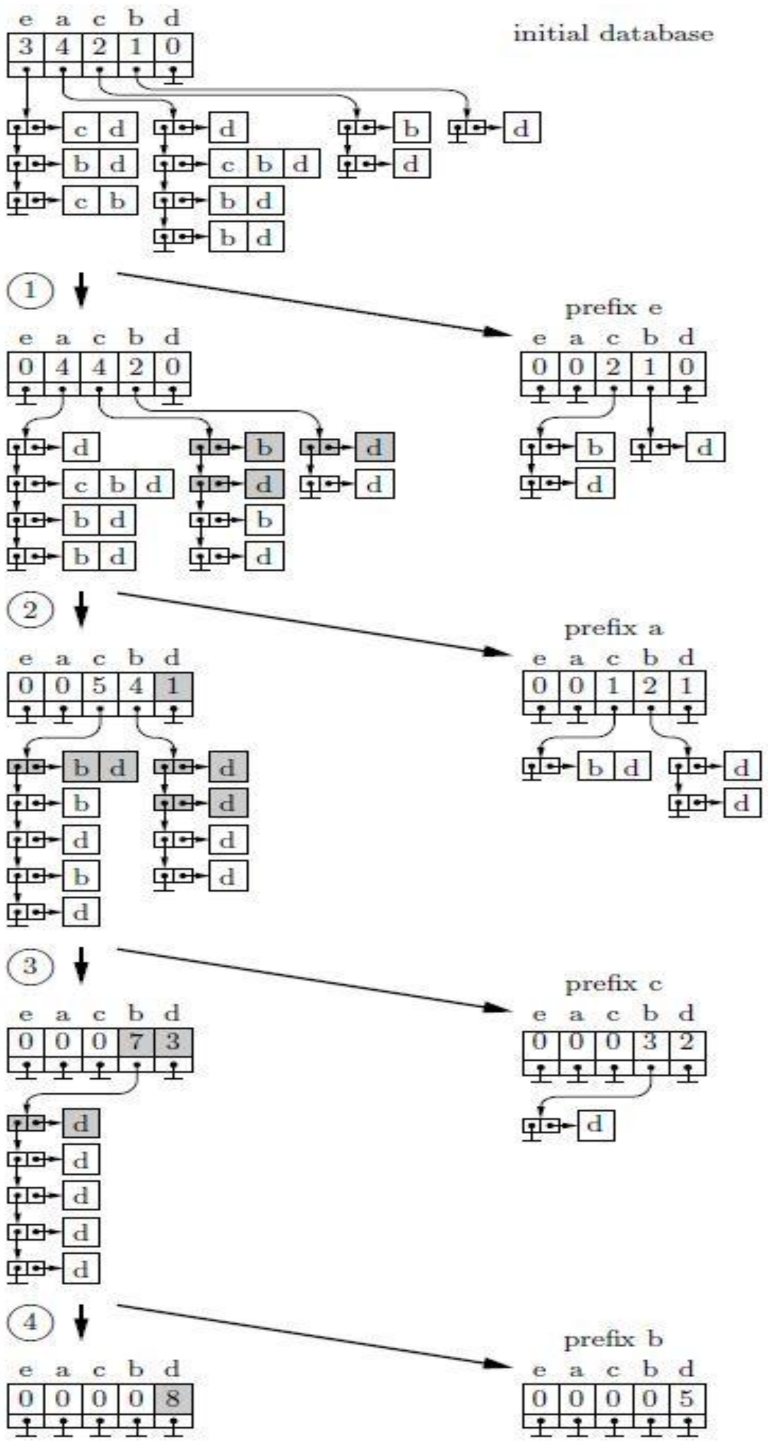


Figure 2.12: Procedure of the recursive elimination with the modification of the transaction lists (left) as well as the construction of the transaction lists for the recursion (right).

2.5 H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases [23,24]

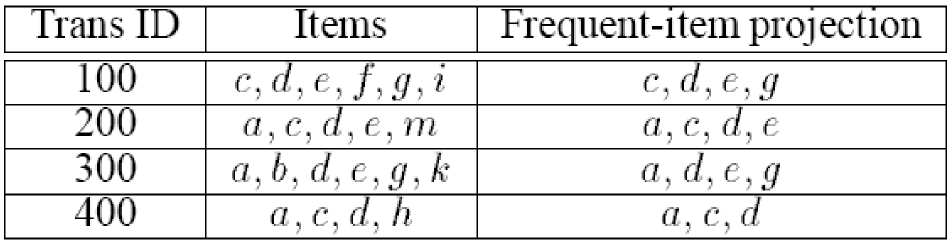
This is the algorithm for frequent itemset mining which uses a simple and hyperlinked data structure, H_struct and adjusts the links dynamically in the mining process. The most distinct feature of this algorithm is that is has limited and precisely predictable space overhead and in memory based settings, it runs really fast. The large databases can be handled through this algorithm by data portioning and for the dense dataset, FP-tree can be constructed dynamically as a part of mining process. H-mine shows high performance in various kinds of data and it is scalable for mining large databases.

2.5.1 Example of H-mine

H-mine (Mem): memory based hyper structure mining First we need to define the problem statement of frequent pattern mining.

Let *I* be the set of items where *I* = {x_1, x_2, ….x_n}. A subset of item is an itemset denoted by X i.e. X_*I* anitemset X= {x_1, x_2, ….x_m}..

Let the first two columns of Table 1 be our running transaction database TDB. Let the minimum support threshold be min_sup =2.



Trans ID	Items	Frequent-item projection
100	c, d, e, f, g, i	c, d, e, g
200	a, c, d, e, m	a, c, d, e
300	a, b, d, e, g, k	a, d, e, g
400	a, c, d, h	a, c, d

Fig 2.13: The Transaction Database TDB as Running Example

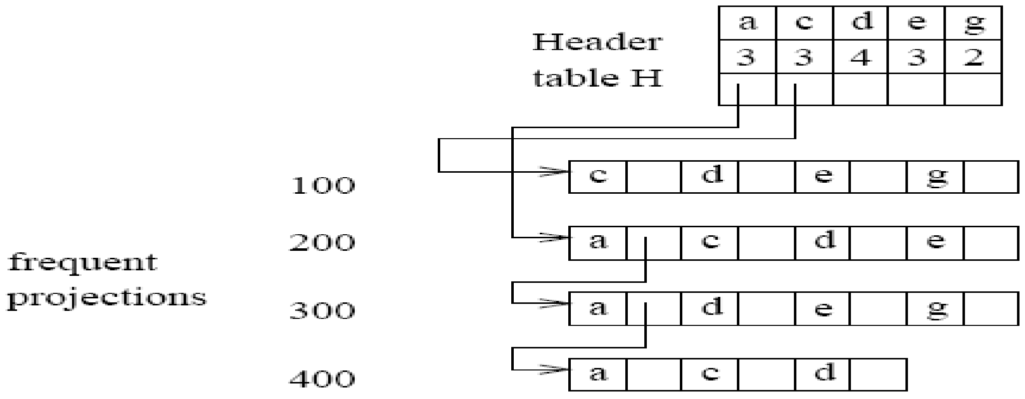


Fig 2.14: H-Struct.

A header table H is created, where each frequent item entry has three fields: an item-id, a support count, and a hyper-link. When the frequent-item projections are loaded into memory, those with the same first item (in the order of *F-list*) are linked together by the hyper-links as a queue, and the entries in header table H act as the heads of the queues. For example, the entry of item a in the header table H is the head of a-queue, which links frequent-item projections of transactions 200, 300, 400. These three projections all have item a as their first frequent item (in the order of *F-list*). Similarly, frequent-item projection of transaction 100 is linked as c-queue, headed by item c. The d-, e – and g-queues are empty since there is no frequent-item projection that begins with any of these items.

Clearly, it takes one scan (the second scan) of the transaction database TDB to build such a memory structure (called H-struct). Then the remaining of the mining can be performed on the H-struct only, without referencing any information in the original database. After that, the five subsets of frequent patterns can be mined one by one as follows:

First, let us consider how to find the set of frequent patterns in the first subset, i.e., all the frequent patterns containing item. This requires to search all the frequent-item projections containing item a i.e., the *a-projected database,* denoted as $|TDB|_a$. Interestingly, the frequent-item projections in the a-projected database are already linked in the a-queue, which can be traversed efficiently to mine the a-projected database, an a-*header table* H_a is created, as shown in Fig 2.14. In H_a, every frequent item, except for itself, has an entry with the same three fields as *H*, i.e., *item-id, support count* and *hyper-link.* The support count in H_a re (i.e., frequent-item projections in the a-queue), thus the support count in the entry c of H_a is 2.

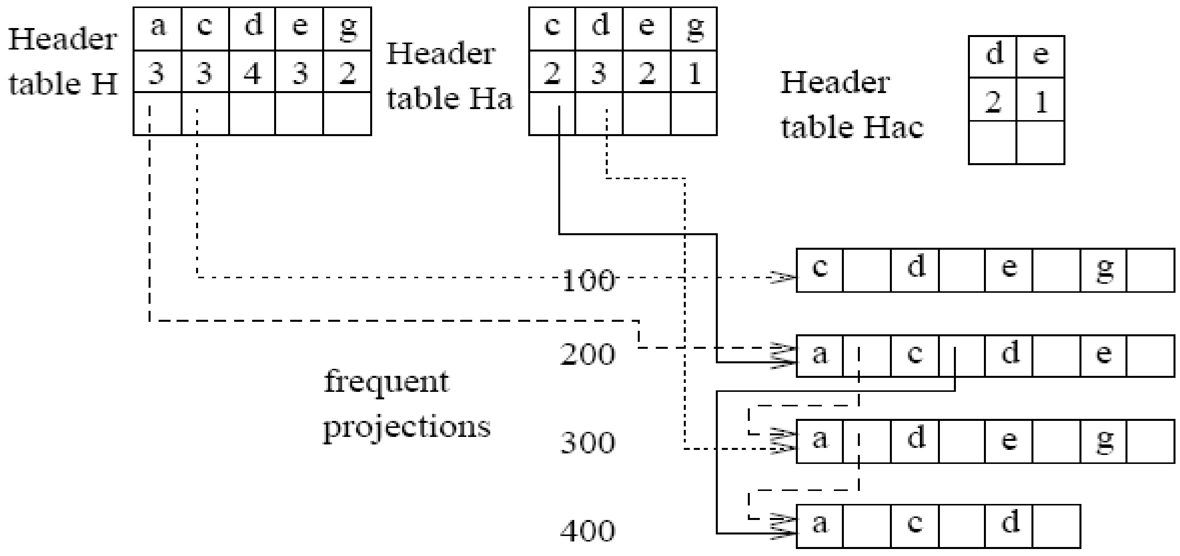


Figure 2.15: Header Table H_a and ac-queue.

By traversing the a-queue once, the set of locally frequent items, i.e., the items appearing at least times, in the a-projected database is found, which is {c:2, d:3, e:2} Note: g:1 is not locally frequent and thus will not be considered further.). This scan outputs frequent patterns {ac:2, ad:3, ae:2} and builds up links for header H_a as shown in Figure 2.15.

Similarly, the process continues for the ac-projected database by examining the c-queue in H_a, which creates an ac-*header table* H_{ac}, as shown in Figure 2.16.

Since only item d:2 is locally frequent item in the ac-projected database, only acd:2 is output, and the search along this path completes.

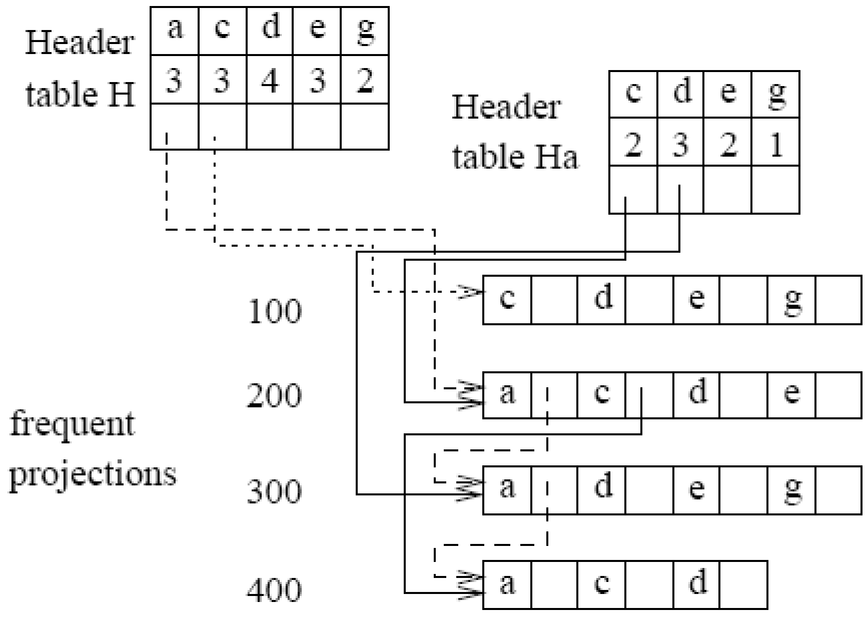


Figure 2.16: *Header Table H_{ac}.*

Then the recursion backtracks to find patterns containing a and d but no c. Since the queue started from d in the header table H_a , i.e., the ad-queue, links all frequent-item projections containing items a and d (but excluding item c in the projection), one can get the complete ad-projected database by inserting frequent-item projections having item d in ac-queue into the ad-queue. This involves one more traversal of the ac-queue. Each frequent-item projection in the ac-queue is appended to the queue of the next frequent item in the projection according to *F-list*. Since all the frequent-item projections in the ac-queue have item d, they are all inserted into the ad-queue, as shown in Figure 2.17.

It can be seen that, after the adjustment, the ad-queue collects the complete set of frequent-item projections containing items a and d. Thus, the set of frequent patterns containing items a and d can be mined recursively.

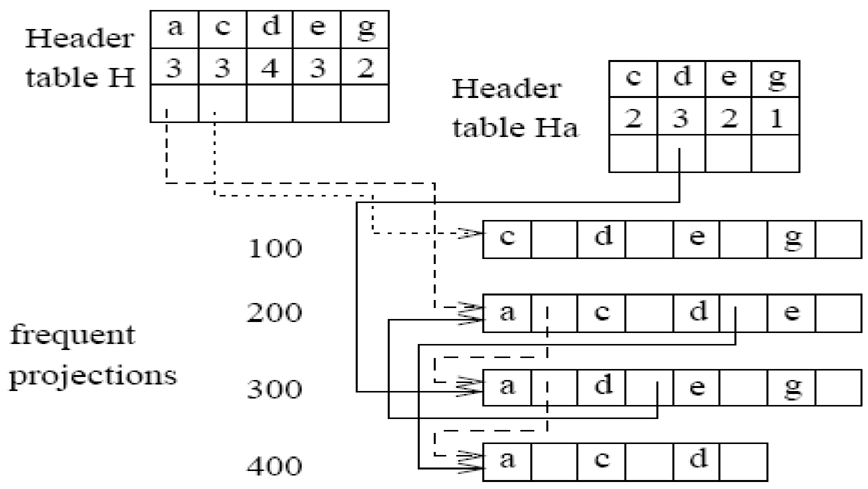


Figure 2.17: Header Table H_a and ad-queue.

Please note that, even though item c appears in frequent-item projections of ad-projected database, we do not consider it as a locally frequent item in any recursive projected database since it has been considered in the mining of the ac-queue. This mining generates only one pattern ade:2 Notice also the third level header table H_{ad} can use the table H_{ac} since the search for H_{ac} was done in the previous round. Thus we only need one header table at the third level. Later we can see that only one header table is needed for each level in the whole mining process. For the search in the ae-projected database, since contains no child links, the search terminates, with no patterns generated.

After the frequent patterns containing item a are found, the a-projected database, i.e., a-queue, is no longer needed in the remaining of mining. Since the c-queue includes all frequent-item projections containing item c except for those projections containing both items a and c, which are in the a-queue. To mine all the frequent patterns containing item c but no a, and other subsets of frequent patterns, we need to insert all the projections in the a-queue to the proper queues.

We traverse the a-queue once more. Each frequent-item projection in the queue is appended to the queue of the next item in the projection following a in the *F-list*, as shown in Figure 2.18. For example, frequent-item projection acde is inserted into c-queue and adeg is inserted into d-queue

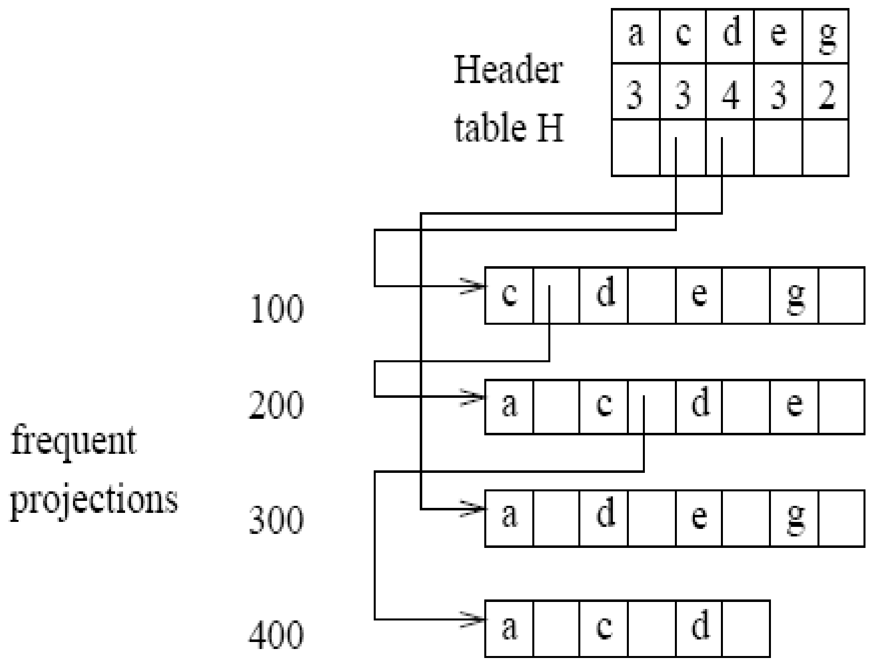


Figure 2.18: Adjusted Hyperlinks after Mining a-projected Database

By mining the c-projected database recursively (with shared header table at each level), we can find the set of frequent patterns containing item c but no a. Notice item a _ will not be included in the c-projected database since all the frequent patterns having have already been found.

Similarly, the mining goes on. It is easy to see that the above mining process finds the complete set of frequent patterns without duplication. The remaining mining process is left as an exercise to interested readers.

Notice also that the depth-first search for mining the first set of frequent patterns at any depth can be done in one database scan by constructing the header tables at all levels simultaneously.

The general idea of H-mine(Mem) is shown in the above example. Comparing with other frequent pattern mining methods, the efficiency of H-mine(Mem) comes from the following aspects.

First, H-mine(Mem) avoids candidate generation and test by adopting a frequent-pattern growth methodology, a more efficient method shown in previous studies. H-mine(Mem) absorbs the advantages of pattern growth.

Second, H-mine(Mem) confines its search in a dedicated space. Unlike other frequent pattern growth methods such as *FP-growth*, it does not need to physically construct memory structures of projected databases. It fully utilizes the information well organized in the H-struct, and collects information about projected databases using header tables, which are light-weight structures. That also saves a lot of efforts on managing space.

Third, H-mine(Mem) does not need to store any frequent patterns in memory. Once a frequent pattern is found, it is output to disk. In contrast, the candidate-generation and- test method has to save and use the frequent patterns found in the current round to generate candidates for the next round.

Chapter 3

Frequent Patterns Mining on Streaming Data

With the emergence of new application, include network traffic analysis, web click stream mining, network intrusion detection; the data we process is continuous data stream not static one [25]. Today's information society has become a restless generator of data of various kinds. Huge amount of data is being generated, from web click stream, credit card records, telephone calls records, traditional retail market store transaction [26]. These records are an extremely valuable source of information. Being continuous, unbounded flow of data which can be read only once, there is several limitation of data stream mining. Firstly, each item can be examined only once. Secondly, limited memory, though the data generated continuously. Finally, the mining results must be as fast as possible. To overcome these limitation, a data stream mining algorithm must be a single pass algorithm, must extract the useful information from the current data stream, which can be used to derive various information required once the data stream has been expired. Transaction arriving in series, forms data stream. Itemset is a set of items. A k-itemset is a set of k items. The *support* of itemset X (sup(X)) is a percentage of transactions containing the itemset[27]. *Frequent itemset* is an itemset X whose sup(x) ≥ s, where "s" is the minimum support given by user. Any subset of frequent itemset is also frequent (*priori property)*. Due to which frequent itemset mining algorithms suffer from the problem of combinatorial explosion. To alleviate this problem, maximal frequent itemset and closed frequent itemset were discovered. A frequent itemset is *maximal frequent itemset* if *none* of its proper supersets is frequent.. Various Time models, like L*andmark window model, Time-fading model, Sliding window model*, are used to mine continuously generated data streams. *Landmark window model* performs mining over all the data from landmark point (usually the time the system start) to current time of mining. *Time fading model, does* not treat all the data equally as, landmark model, and distinguish new data from old data by assigning different weights. It also considers data from start of stream up to current time. Both landmark model and time fading model does not provide time-sensitivity while performing mining and support count of entire data, from start to end is to be counted. These limitations are being overcome in *sliding window model*, which fulfil all the requirements for data stream mining; Time-sensitivity, Approximation, Adjustability[28]. Sliding window models utilizes only latest W transaction received, W is the size of window, defined by the user.

Number of algorithms has been proposed using different time models, for the extraction of frequent itemset from data stream. One challenge all these algorithm faces is to develop a data structure that can capture full stream content, in a memory-efficient manner, with a single pass. We are discussing algorithms proposed using the sliding window time model for mining data streams.

1. ***Moment Algorithm:*** Chi et al.(2006)[29] introduced one algorithm, ***Moment***, to extract *closed frequent itemset* within sliding window. They designed, a prefix based data structure, **CET** (Closed Enumeration Tree), to maintain closed frequent itemset. CET maintains the boundary between *closed frequent itemset* and rest of itemset, which makes the boundary relatively stable, whenever any itemset changes its state (frequent to non-frequent vice-versa), ultimately reducing the updating cost. An efficient algorithm to incrementally update the CET, which update the CET when newly arrived transaction change the content of window or oldest transaction being deleted from the window.

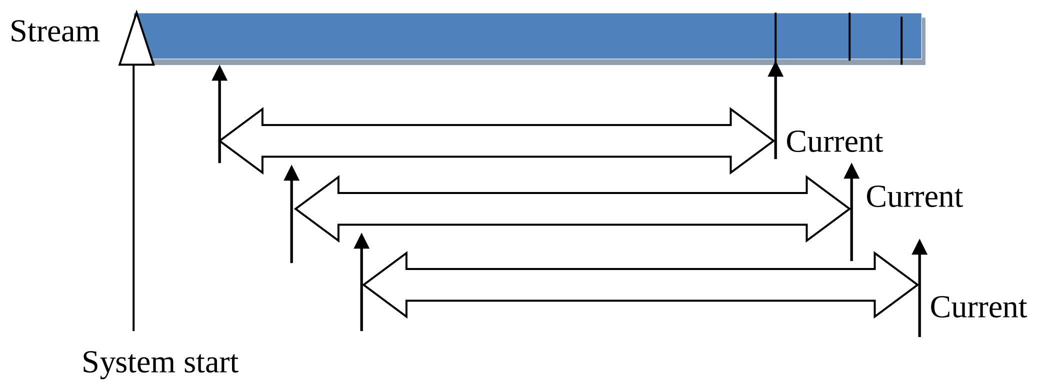


Fig 3.1: Mining frequent itemsets from the latest W transaction using sliding window model

When a transaction arrives/expires, Moment traverse the part of CET related to that transaction. For each node visited, Moment, update the CET by incrementing/decrementing its frequency. The merit of Moment is that it computes the exact set of *closed frequent itemset* over a sliding window. Although an update to a node result in a propagation of the node insertion and deletion in the CET, most of the nodes related to an incoming or expiring transaction do not change their type often. Therefore, the average update cost in the CET is small. Limitation of Moment algorithm is, it stores transaction using data structure, FP tree (Han et al.2004), which require a considerable amount of memory. Secondly, if the size of window is too large, CET can huge.

2. ***WSW algorithm:*** Pauray S.M. Tsai(2009)[30], proposed a new framework data stream mining, called weighted sliding window model, which allows user to specify the number of windows, weight of window and size of window. A single pass algorithm, called WSW, has been proposed to extract frequent itemset from data streams. The motivation for weighted sliding window model come from the fact, that the size of traditional sliding window model is fixed or defined by the number of transaction, say W. Though recent W transactions are considered, the time to cover these W transactions may be long or varies, which may effectively decrease the mining result. The weighted sliding window model defines window size by time not by the number of transaction and user can specify the number of windows, with each window assign with different weight (sum of all window weight equals to 1). For example data may be more influential at current moment and hence, should be assigned higher weight.

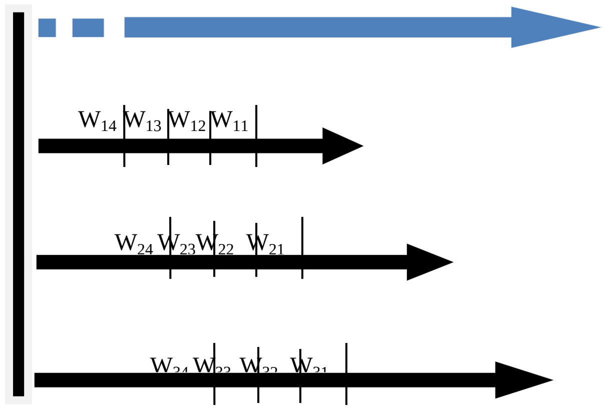

W_{14} W_{13} W_{12} W_{11}
W_{24} W_{23} W_{22} W_{21}

Figure 3.2: Weighted sliding window model

The algorithm WSW scans the data once in each window, and calculates the support count for each item present in current window, to find out frequent k-itemset (k=1). Based on this information candidate (k+1)-itemset are pointed out to find frequent (k+1)-itemset. The process terminates when no further candidates itemset can be generated. If the number of windows increases the time to determine frequent itemset increases. To reduce this runtime they proposed an improvement of WSW algorithm called, WSW-imp, which further reduce the time of deciding whether a candidate itemset is frequent or non-frequent itemset.

3. ***TMoment Algorithm:*** Fatemeh Nori et al. (2012)[31], proposed another efficient algorithm, called **TMoment**, for *closed frequent itemset* mining using sliding window model. This algorithm uses single novel prefix tree based data structure, **TCET** (Transaction Translate Closed Enumeration Tree), for storing both transaction of the window and closed frequent itemset. This reduces the considerable amount of main memory usage. The proposed algorithm, Tmoment, consist of four steps: Computing support using transaction, Building TCET, Eliminating the oldest transaction, adding the new transaction. Tmoment overcomes the limitation of Moment, high memory usage, with the help of TCET. Tmoment does not require a separate data structure (FP-Tree), as used in moment for storing transactions of window. TCET stores both transaction and corresponding frequent itemset, which considerably reduces the memory usage. On the other hand TCET affects the runtime in negative manner. List-intersection has been used to find out support of itemsets, which makes the run-time of Tmoment no better than Moment.
4. ***VSW Algorithm:*** Mahmood Depyir et al. (2012)[32] proposed a new algorithm, VSW (Variable Size sliding Window frequent itemset mining) which is suitable for observing recent changes in set of frequent itemset. In Transactional sliding window the window size is to be kept constant, which is being obtained from user. In order to determine the precise size of window, the user must have advance knowledge about time and scale of changes within data stream, which cannot be easily determined due to unpredictable changing nature of data streams. To overcome this limitation of fixed window size, VSW, algorithm has been developed. In this window size is determined dynamically based on amount concept change that occurs within the arriving data streams. Initially

window size is given by user and then adjusted based on the change of frequent pattern embedded in the incoming data stream. The window size expands as the concept becomes stable and shrinks when a concept change occurs.

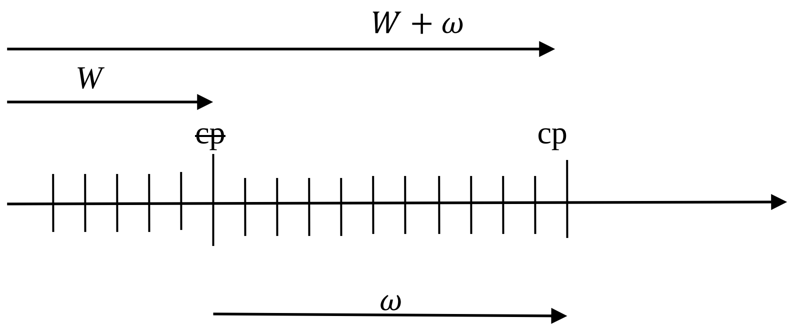


Figure 3.3: Window size reduction and checkpoint (cp) movement after change detection.

Fig. 3.3 shows the process of cutting old transaction and forming the new window. Concept change is calculated with respect to checkpoint (tid of last transaction received).If a concept change is detected, all information before the checkpoint is removed from the window and new window formed (window shrinks). If not detected, window size continues to grow with further no action.

Chapter 4

Hadoop

As the data is increasing day by day, it is becoming quite difficult to manage it. We need to have records of almost all the field and the data produced by many field is unstructured. To manage large and unstructured data there is need of special framework which is Hadoop.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures [33]. It can handle big data which is terabytes in size and mostly semi structured or unstructured. Big data cannot be handled with the traditional technologies.

4.1. Components of Hadoop [33]

Hadoop has two components:

I. HDFS (hadoop file system) and

II. Map Reduce

Both these terms are inspired with Google papers published in last decades.

Before discussing components of hadoop we must know about some terms which is mainly used in hadoop

4.2 Blocks

Physical disk and a file system is divided in blocks, so block is the fundamental unit of space. In Hadoop also we divide the large file in blocks and the default size of block is 64 MB and it can be customized. Unlike file system for a single disk, HDFS block do not take up the whole block if its size is smaller. It provides benefit over it

- First, a file on the HDFS node can be larger than the single disk in the network.
- Second, blocks simplify the storage subsystem in terms of metadata management of individual files.
- Finally, blocks make replication easier, providing fault tolerance and availability.

The blocks are replicated on the other machines (at least three). The replication is done on multiple machines, if the file become unavailable due to corruption or machine failure then it can be recovered from the other machine. The failure is transparent to the user that is , if file is unavailable from one machine then it will read from other machine and all this process is transparent to user.

4.3 HDFS (hadoop file system) [34]:

HDFS is the component which handles the storage of data. It is the file system which handles the efficient storing and processing of large files on one or more clusters. It is designed on the philosophy of write once and read many times so that the computing can be made more efficient. The biggest advantage of HDFS is fault tolerance without losing data. In HDFS the large file is broken into small blocks and distributed over clusters of low cost hardware. Each block is copied at least on three clusters so that they can be made fault tolerance.

4.4 Namenodes and Datanodes

Hadoop works in multiple cluster environments. It works on master-slave architecture one cluster works as master and all other works as slave. The master node contain namenode and the slaves have datanodes. The filesystem namespace ,where the data will be stored, the filesystem tree, and the metadata for all the files and directories is managed by Namenode. Namenode keeps track of all the datanodes that which datanode is residing on which cluster. Datanode contains the HDFS blocks and notify the namenode that which data or block it is containing.

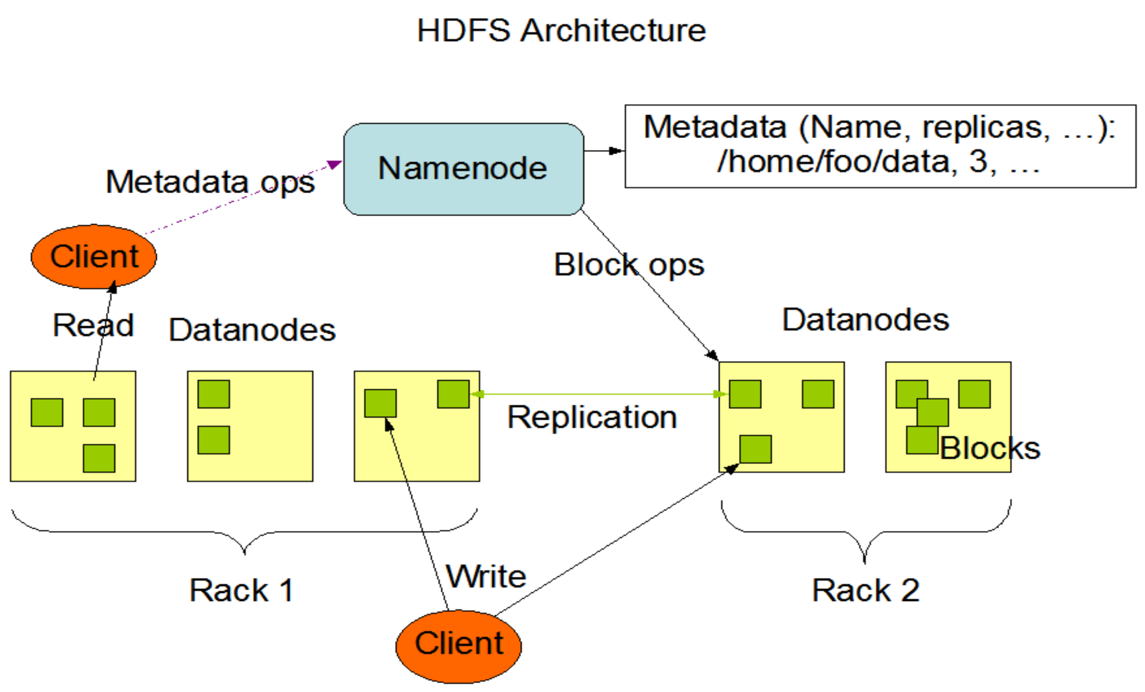


Fig 4.1: HDFS Architecture

4.5 Map Reduce [36]

It is the processing component of Hadoop. It is a programming model for parallel processing. Map reduce has two phases: map phase and reduce phase. In map phase the problem is divided into sub problems and processes independently and in reduce phase the output of the map phase is combined to give the output of the large problem. The developers define the corresponding map and reduce function.

As input and output, each function has key-value pairs. In the map reduce A MapReduce job there is a mapper program, a input data and the configuration details. Hadoop runs a job by dividing it into two types of tasks: map tasks and reduce tasks.

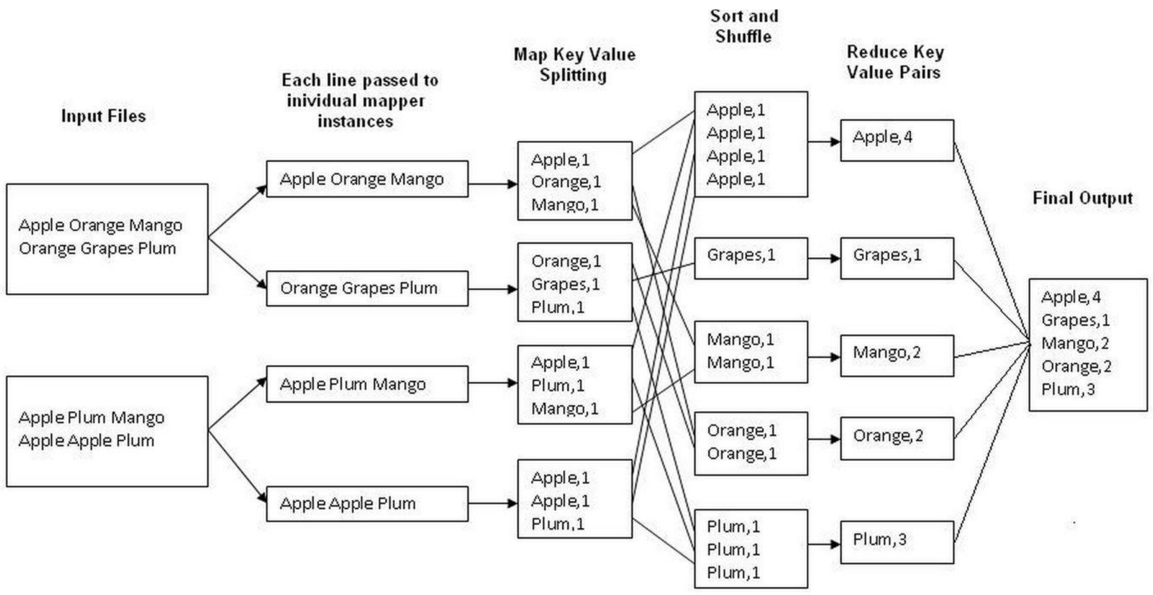


Fig 4.2: Example of Map Reduce

The map task invokes a user-defined map function that processes the input key-value pair into a different key-value pair as output. When the demand of high processing is there, then it is better to add more mapreduce job rather than adding more map and reduce functions. In creating a MapReduce program, the first step for the developer is to set up and configure the Hadoop development environment. Then the developer can create two separate map and reduce functions. Ideally, unit tests should be included along the way in the process to maximize development efficiency. Next, a driver program is formed to run the job on a subset from the developer's development environment, where the debugger can identify a potential problem. Once the MapReduce job runs as expected, it can be run against a cluster that may surface other issues for further debugging. Two node types control the MapReduce job execution process: a job tracker node and multiple task tracker nodes. The job tracker coordinates and tracks all the jobs by scheduling task trackers to run tasks.

They in turn submit progress reports back to the job tracker. If a task fails, the job tracker reschedules it for a different task tracker. Hadoop divides the input data set into segments called splits. One map task is created for each split, and Hadoop runs the map function for each record within the split. Typically, a good split size matches that of an HDFS block (64MB by default but customizable) because it is the largest size guaranteed to be stored on a single node. Otherwise, if the split span two blocks, it may be possible that the processes gets slow as some of the split is to be transferred across the network to the node operation the map task. Because map task output is an intermediate step, it is written to local disk. A reduce task ingests a map task output to produce the final MapReduce output and store it in HDFS. If a map task were to fail before handing its output off to the reduce task, Hadoop would simply rerun the map task on another node. When many map and reduce tasks are involved, the flow can be quite complex and is known as “the shuffle.” Due to the complexity, tuning the shuffle can dramatically improve processing time.

4.6 Problems Solved by Hadoop

Hadoop solves three major challenges of scale-out computing using low-cost hardware:

1. Slow disk access to data;
2. Abstraction of data analysis (separating the logical programming analysis from the strategy to physically execute it on hardware);
3. And server failure.

The data abstraction that MapReduce solves is described by the Namenode which allow the access of the required data only; the two other challenges are solved by using hadoop by applying elegant strategy of using low cost hardware. If all the data is on single disk then the access of data will take time but if distribute the data on clusters then the disk access speed can be increased as the data will be accessed parallel. The storage capacity can also be maintained as the number of clusters can be included when needed.

Chapter 5
Proposed Approach

Mining large amount of data is quite difficult. To scan and manage large databases increase the computation time very much. The time to find the frequent itemset is much less than scanning the whole databases. Most of the approaches works on the parallel and distributed mining, but it faces a problem of the communication cost. In parallel and distributed mining we use the divide and conquer strategy. But in this there is a problem of transferring divided to each machine which is time consuming. If the data is so large and complex in terms of items and user behavior then this strategy suffers a lot.

5.1 Real Time Mining and its significance

Real time mining is basically the mining which is done on continuous data to keep them updated. Frequent pattern mining in real-time is of increasing thrust in many business applications such as e-commerce, recommender systems, and supply chain management and group decision support systems, to name a few. A plethora of efficient algorithms have been proposed till date. However, with dense datasets, the performances of these algorithms significantly degrade.

Business intelligence is playing an essential role in achieving business goal such as profitability, efficiency, customer preservation and market incursion. In most of the cases of frequent pattern mining we use historic data. Now the thing is that, if the historic data can help us in making good decision then how real time mining can make the decision process better. By using up to date information we can get rid of delays and speed up in the competitive environment. In various areas real-time decision making is important some of them are real-time supply chain management system, real-time customer relationship management, real-time recommender systems, real-time stock management and vendor inventory, real-time enterprise risk and vulnerability management, real-time operational management in which critical real-time information is required such application is in mission, airline industry, fraud detection, real time negotiation and many other areas like real-time dynamic pricing and discount offering to customers in real-time.

5.2 Proposed Approach

In the present era we need to be up to date and need to know the present demand, for that we need to analyze the activities regularly and take the decisions by analyzing the patterns generated from that set of data. Analyzing real time data (data of certain interval) can help in doing so. For having good patterns we have to maintain the past history, means on the basis of past and present we can take decision for future. This only concept we are using in our approach and through an example we will be able to know how we will find the frequent itemsets and how the approach is beneficial for the business analysis and maintain the stocks.

I have proposed a real time mining strategy, in which mining is done on the regular basis. In my approach the data is directly save on the cloud and we will set the time or period at which the mining algorithm mine the data collected.

In this algorithm we will find the frequent patterns which are frequent on the present date. In this we will maintain a list of frequent itemset which will be stored in the list which is globally frequent.

All the transactional databases of organisation will be stored in the common storage. Storage can be cloud or can be their own storage system. This figure can be better understood by an example; take the example of vishal mega mart. There are many stores of Vishal mega mart around India. The transactions of all the stores are stored on common storage system for mining. Now you must be thinking it is centralized that is damage in the storage system will crash the whole organisation. But this is not true for this we are having fault tolerance system that is the storage system has its replicated copy of data at another storage system which is the supportive system and used for load balancing and security and fault tolerance.

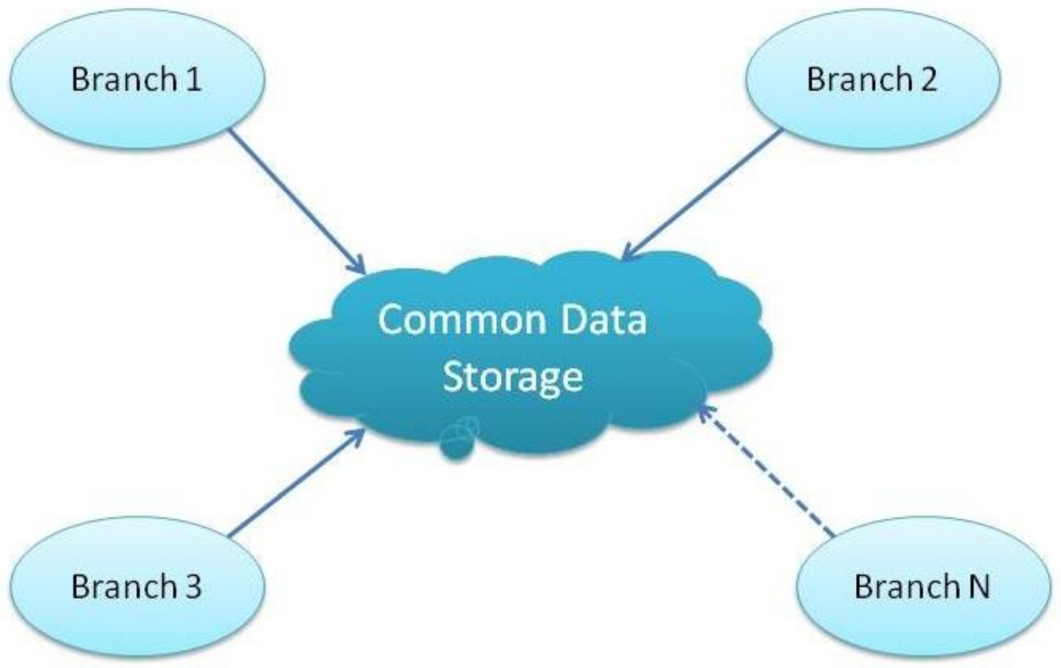


Fig 5.1: All the Organization Store its Data at a Common Place

The mining will be done on the regular bases, set a time interval at which the data mining algorithm starts mining data mine.

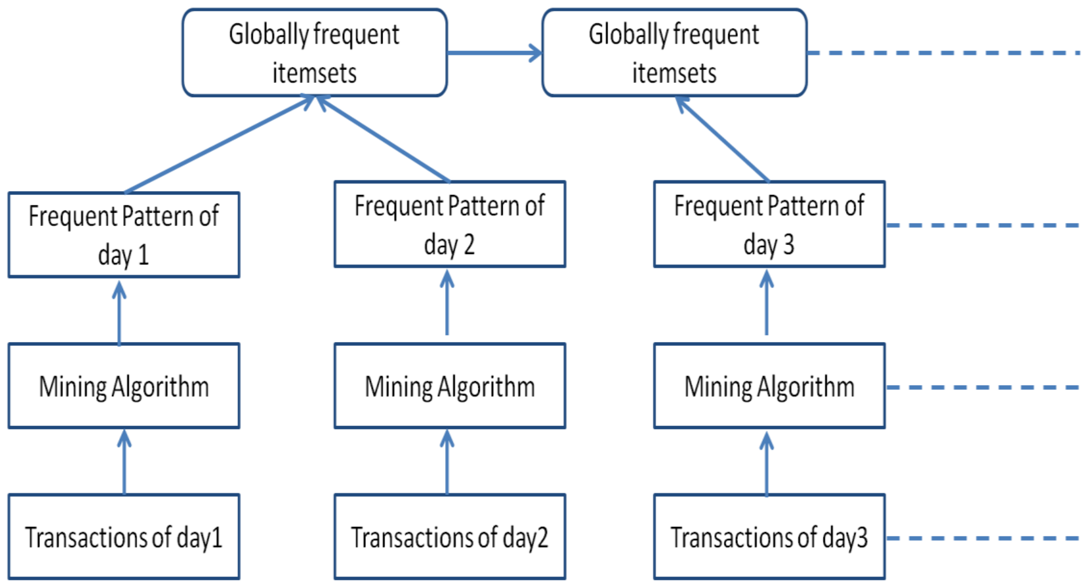

Globally frequent itemsets
Globally frequent itemsets
Frequent Pattern of day 1
Frequent Pattern of day 2
Frequent Pattern of day 3
Mining Algorithm
Mining Algorithm
Mining Algorithm
Transactions of day1
Transactions of day2
Transactions of day3

Fig 5.2 Mining Process

How the mining process is going on is shown by figure 5.2.

5.3 Algorithm of Approach

In the very first step the data collected in the first interval under goes mining by using FP-growth algorithm. The advantage of FP-growth algorithm is that it generates FP-tree without generating candidate sets. It requires only two scan of database to generate FP-tree. Hence the computation cost of this algorithm is low than any other algorithm. After generating the FP-tree we will convert it into bFPT (briefest FP-tree) so that less storage is required to store the result. From the FP-tree we generate frequent patterns which are further used in getting the result of mining next interval. Fig3 shows the flow chart of how the process of mining is done on the regular basis. If we mine on daily basis the scenario would be as

- Day 1: the data is stored from various sources. Suppose at 12:00 am mining process is triggered automatically, all the data stored in Day 1 goes under processing and the FP-tree is generated by the FP- growth algorithm. This is further compressed and store in the form of bFPT for future use. Now the patterns are generated from the FP-tree which will act as input for the global list of frequent itemset.

- Day 2: the data is stored from various sources. Suppose at 12:00 am mining process is triggered automatically, all the data stored in Day 2 goes under processing and the FP-tree is generated by the FP- growth algorithm. This is further compressed and store in the form of bFPT for future use. Now the patterns are generated from the FP-tree which will act as input for the global list of frequent itemset.

- Now we have the data of day1 and day2. We will merge the frequent pattern to find the globally frequent patterns. In merging process we will find the average of a pattern from both the frequent dataset if the average is greater than t he min_sup then include it in global set of frequent mining otherwise the pattern is infrequent.

- Once the global list of frequent itemset is generated we will merge the further frequent patterns into the global list by the process we had followed in merging the dataset of day1 and day2

- Now we will merge global frequent pattern (GFP) and dayN dataset

- Today: data frequent pattern of yesterday and data stored today are mined though the algorithm and we have the updated result.

Note: *It is not necessary that the mining should be on daily basis. It can be weakly or at alternate days. It depends on the requirement of the user or on the size of the data stored. The only thing that this approach wants to say is that the mining should be automatically triggered periodically as set by the user.*

This is the procedure that how the approach is working and how it is reducing computation as well as communication cost. In what sense it is easy to maintain. As all the process is automatic so this is easy to implement and also convenient to use. It keeps you upto date and helps you to take business decision on the present requirement. It helps you to know what the present requirement is. On this basis business analyst can take better decision as its knowledge extracted from frequent patterns is up to date.

So this approach not only reduces time and space complexity but also helps in taking better decisions.

Pseudo code of algorithm (GFP)

Step 1: Collect data from various sources of first interval.

Step 2: Apply frequent pattern mining algorithm to get frequent patterns on dataset of first interval.

Step 3: Collect data for interval 2 from various sources.

Step 4: Apply frequent pattern mining algorithm to get frequent patterns on dataset of second interval.

Step 5: Merge the frequent patterns of the datasets to find the globally frequent itemset

(i) Find the average of frequent pattern from both the dataset.

(ii) Repeat step (i) for all the frequent item of both the intervals.

Step 6: Create list of globally frequent list of frequent itemset by merging the frequent patterns of the dataset of interval 1 and interval 2.

Step 7: Collect the data set of interval 3 and find the frequent patterns of interval k

Step 8: Merge the frequent pattern generated with the global frequent itemsets.

Step 9: Repeat the step 6 –step 8 for the dataset of each interval.

Above is the pseudo code for the proposed approach which mine on real time basis and keep you updated regarding the frequent patterns of the transactions.

5.4 Advantages of Approach

- Mining is done at regular bases: the data is directly stored in the database where the mining is done. So the mining is always done on the updated data. This is one of the advantages of this approach.

- Frequent patterns are upto date: the results are always upto date as the mining is done on the recent data. By doing so we have various advantage, if we talk about the already proposed algorithm then they mining do on large databases which is petabytes in size which is generated in years in a organisation, it may be possible the pattern which is frequent is not frequent now i.e if first 100 transaction shows the pattern frequent but after first 100 transaction it becomes un frequent, but still it will be shown frequent in result, but in our approach this won’t happen the frequent patterns will change with time.

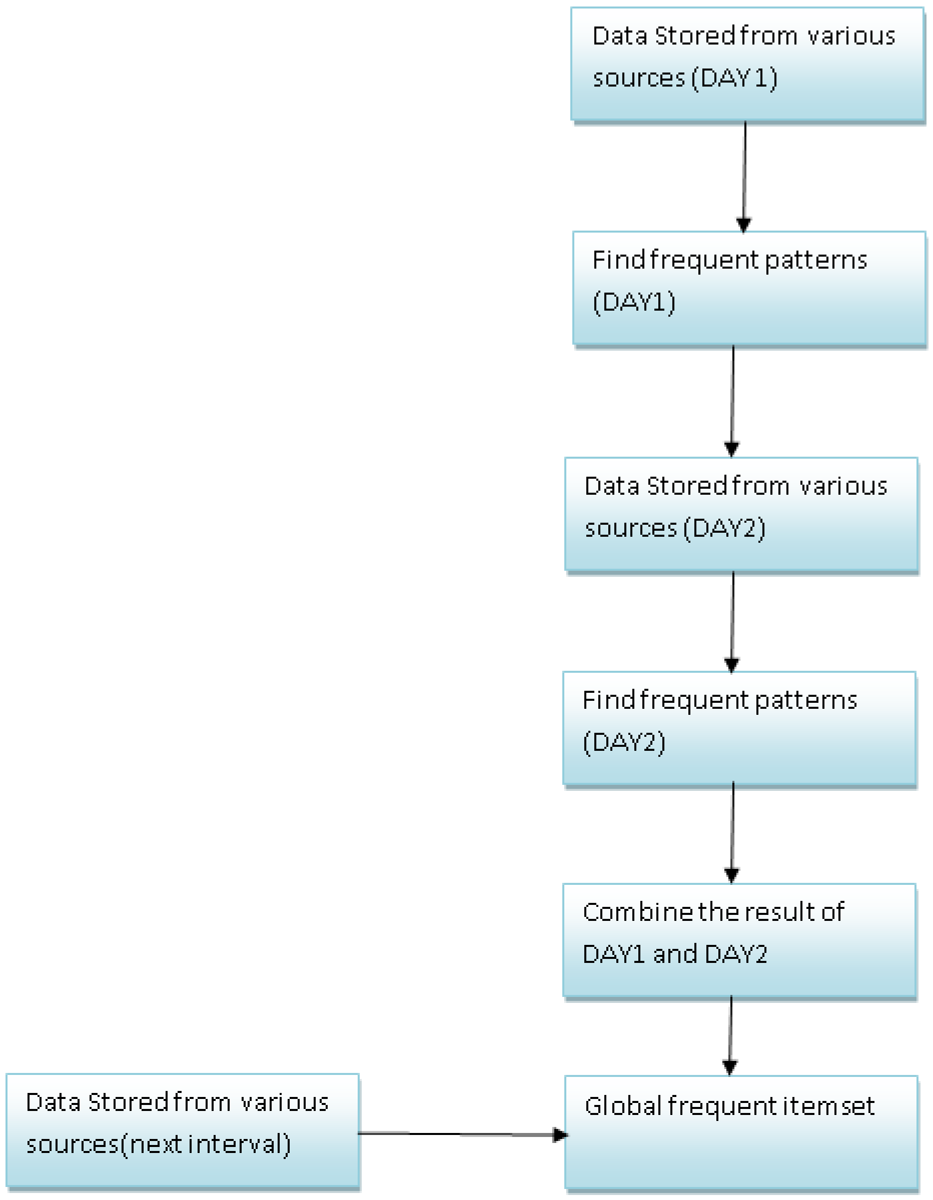

Data Stored from various sources (DAY 1)
Find frequent patterns (DAY1)
Data Stored from various sources (DAY2)
Find frequent patterns (DAY2)
Combine the result of DAY1 and DAY2
Global frequent itemset
Data Stored from various sources(next interval)

Fig 5.3 Block diagram of GFP algorithm

- Computation time is low: as the computation is done on regular basis, the time for computing is low. The data collected in a day can maximum reach to GB's which not much large data to mine is. So the time to mine the data is small as compared to mine large amount of data.

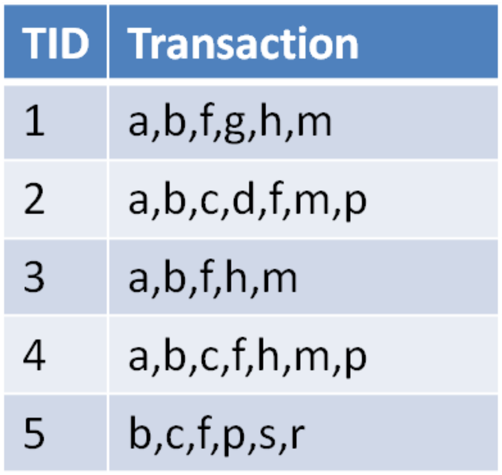
- No need to have high computation devices (like super computers) for computation: computation power required for mining can be moderate that is we don't need the system with very high computation power. All other approaches require high computation power because of large amount of databases.

- Cheap to apply: All other approaches require high amount of storage, high computation power and high RAM which is very costly, but our approach is very cheap in comparison to all other approaches. Storage cost is very much less than computing cost and our approach although requires much storage but computation cost is less in comparison to other approaches. So this algorithm is much cheaper to apply. By implementing this approach we can make mining cheap and convenient process.

- Complexity reduces: generally the complexity increase with the increase in size of the database and also increases with the increase in itemsets. Our approach solves this problem by reducing the complexity by reducing the size of data to be mined.

5.5 Example of the Proposed Approach

Consider a dataset for interval 1



TID	Transaction
1	a,b,f,g,h,m
2	a,b,c,d,f,m,p
3	a,b,f,h,m
4	a,b,c,f,h,m,p
5	b,c,f,p,s,r

Fig:5.4 Dataset for day 1

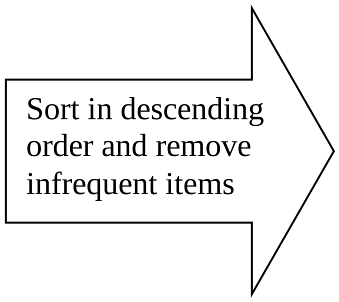
Suppose the dataset in fig 5.4 is the data of the Day 1. First we create the FP_tree of this dataset. Consider the support of 30%. The fp tree of the dataset of day1 is generated by applying FP-growth algorithm. It follows the following steps:

1. In the first scan of dataset the support of each item is calculated and stored in the frequency table. The items having the frequency less than the min_sup is pruned.

2. In the second scan of the items in all the transaction is arranged in descending order of their support.

3. After having the sorted and filtered transaction we will apply the algorithm to construct the FP-Tree, so that we can find the frequent items from that tree.

Item	Support
a	4
b	5
c	3
d	1
f	5
g	1
h	3
m	4
p	3
r	1
s	1



Item	Support
b	5
f	5
a	4
m	4
c	3
h	3
p	3

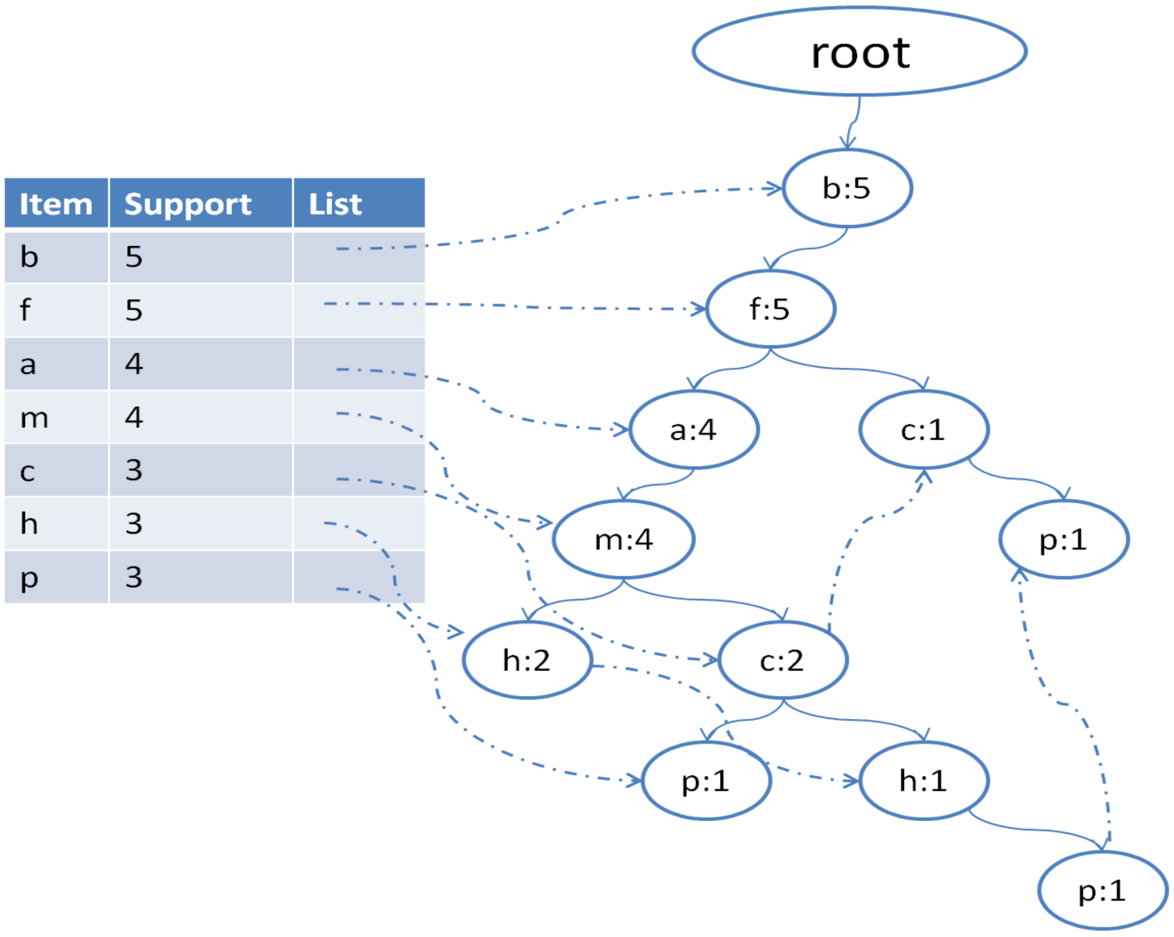
Fig 5.5: Items with their support (left), frequent items in descending order of their support(right)

Left hand side table is showing the support of each item and stored in the table. It is the result after the first scan of dataset. The table in the RHS is showing the table in which all the items in the LHS list are sorted in the descending order and the items having the value less than 30% is removed from the list.

TID	Transaction
1	b,f,a,m,h
2	b,f,a,m,c,p
3	b,f,a,m,h
4	b,f,a,m,c,h,p
5	b,f,c,p

Fig 5.6: Sorted and Filtered Day1 dataset's transactions

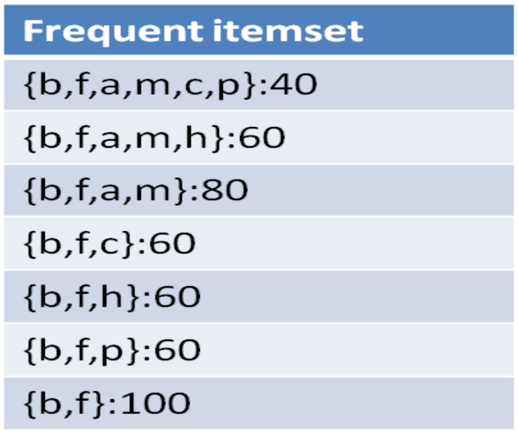
The above is the result of second scan of the dataset. In this all the items in transaction is arranged in descending order of their support. From this table we will construct the fp-tree, which stores all the transaction in the form of tree. And the this tree also stores how and when an item is appearing in which transaction.



Item	Support	List
b	5	
f	5	
a	4	
m	4	
c	3	
h	3	
p	3	

Fig 5.7: FP-tree of the dataset of DAY 1

After constructing Fp-tree from the dataset find the frequent patterns from the FP-tree.

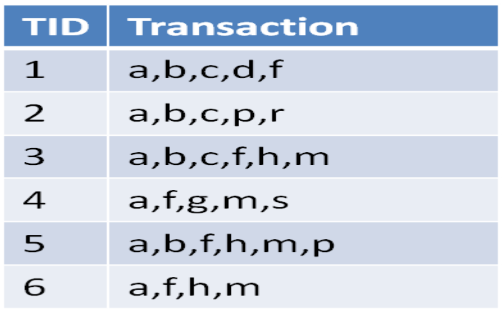


Frequent itemset
{b,f,a,m,c,p}:40
{b,f,a,m,h}:60
{b,f,a,m}:80
{b,f,c}:60
{b,f,h}:60
{b,f,p}:60
{b,f}:100

Fig 5.8: Frequent patterns of dataset DAY1 with their support

From the tree generated by applying the FP-growth algorithm find all frequent itemset. Frequent pattern mining has the property if a set is frequent then all its subset will be frequent. So we need not to mention all the frequent sets, just mention the superset with it support in percentage.

Day 2: collect the data of the second day. Table below shows the data collected on 2nd day. And apply the same process to extract the frequent itemset as applied on the dataset of day 1.



TID	Transaction
1	a,b,c,d,f
2	a,b,c,p,r
3	a,b,c,f,h,m
4	a,f,g,m,s
5	a,b,f,h,m,p
6	a,f,h,m

Fig 5.9: Dataset of DAY2

Item	Support
a	6
b	4
c	3
d	1
f	5
g	1
h	3
m	4
p	2
r	1
s	1

Sort in descending order and remove infrequent items

Item	Support
a	6
f	5
b	4
m	4
c	3
h	3
p	2

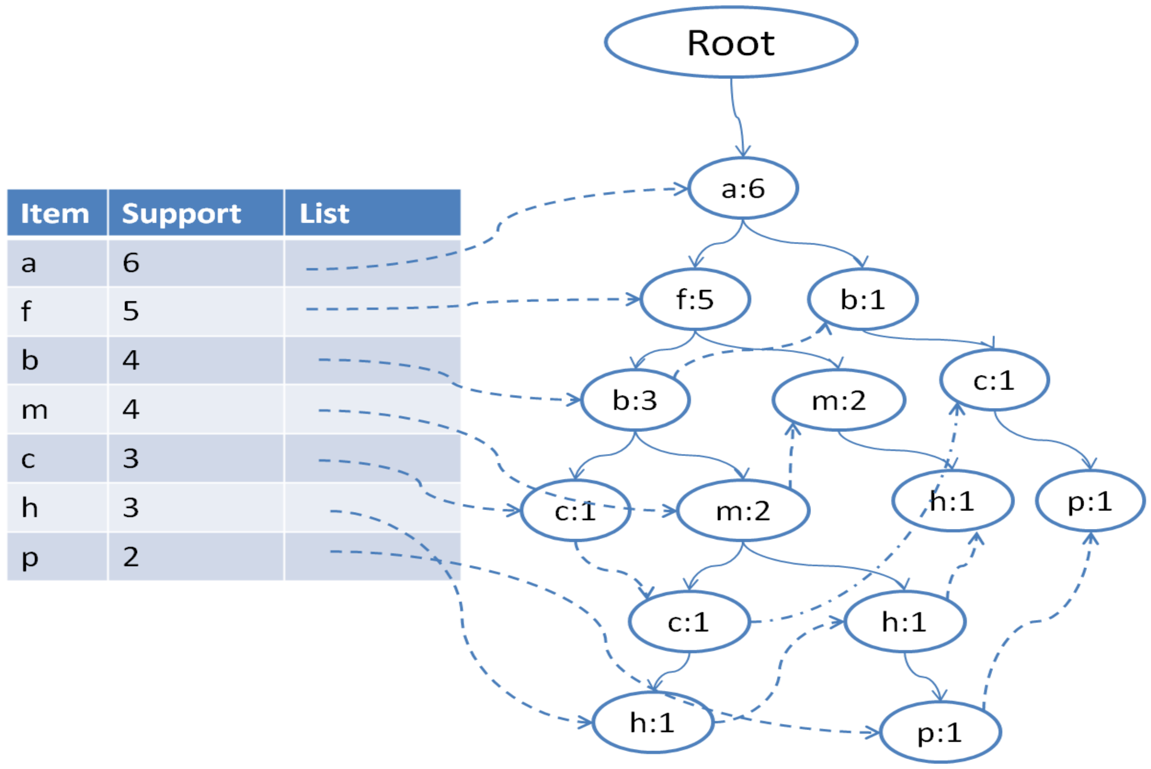
Fig 5.10: Items are arranged in descending order and less frequent items are removed

As we have done in first dataset we will sort all the transaction in descending order.

TID	Transaction
1	a,f,b,c
2	a,b,c,p
3	a,f,b,m,c,h
4	a,f,m
5	a,f,b,m,h,p
6	a,f,m,h

Fig 5.11: Sorted and Filtered Day2 dataset's transactions

From the above table we will generate FP-tree and find the frequent itemsets from the tree by using FP-growth algorithm.



Item	Support	List
a	6	
f	5	
b	4	
m	4	
c	3	
h	3	
p	2	

Fig 5.12: FP-tree of the dataset of DAY2

From the tree of Day2 we will extract the frequent patterns which is shown in the below figure.

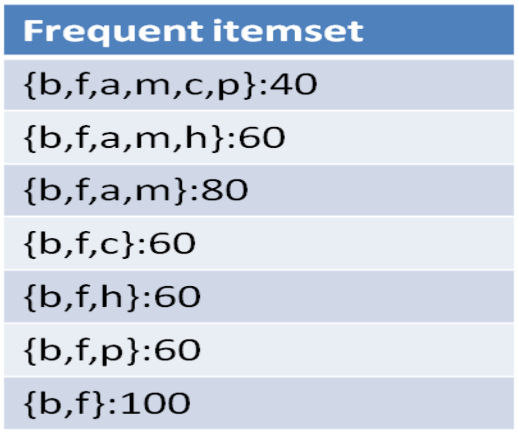
Frequent Itemsets
{a,f,b,m,h}:33.33
{a{f,b,c}:33.33
{a,f,b}:50
{a,b,c}:50
{a,f,c}:33.33
{a,b,p}:33.33
{a,f,m}:66.66
{a,b}:66.66
{a,f}:83.33
{a,h}:50
{a}:100

Fig 5.13: Frequent patterns of dataset DAY2 with their support

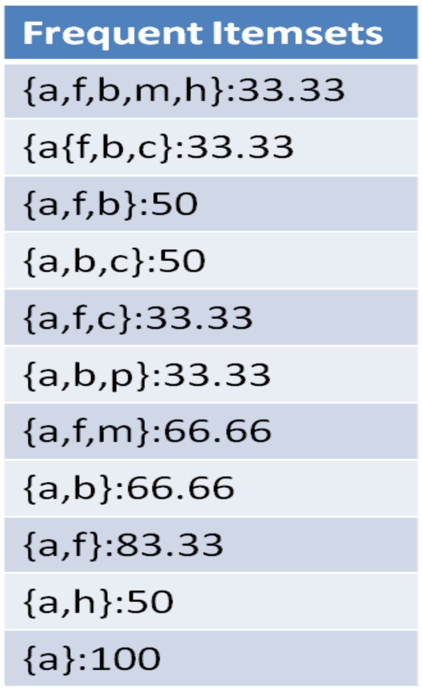
Now we have the table of frequent itemsets of day1 and day2.

By combining the frequent itemset of both the day we will generate set globally frequent itemset. Once we have set of globally frequent itemset, we will combine the result of succeeding day with the global frequent set. In this way we can maintain the past history and can also focus on present.

Now we will discuss how global frequent set is generated and how the further results are merged in it.



Frequent itemset
{b,f,a,m,c,p}:40
{b,f,a,m,h}:60
{b,f,a,m}:80
{b,f,c}:60
{b,f,h}:60
{b,f,p}:60
{b,f}:100



Frequent Itemsets
{a,f,b,m,h}:33.33
{a{f,b,c}:33.33
{a,f,b}:50
{a,b,c}:50
{a,f,c}:33.33
{a,b,p}:33.33
{a,f,m}:66.66
{a,b}:66.66
{a,f}:83.33
{a,h}:50
{a}:100

Fig 5.14: Frequent pattern of DAY1(left) and frequent pattern of DAY2(right)

We have the frequent item sets of day1 and day2. Now we have to merge the itemsets to have the global set of frequent items.

The itemsets which will be in the global set are those which have the support greater than min_ sup. The support of itemset is the average of the support in both the frequent item dataset. This will be clearer by the example.

Calculation of itemsets to be in global set:

Find the support of the frequent itemset that will be in global list of frequent item set. The support of itemsets to be in global frequent list is calculated by having the average of the frequent pattern from both the list.

The dataset of day 1 gives the frequent itemset {b,f,a,m,c,p} with support of 40% while this set is not frequent in the dataset of day2, so the average of support is 20%. The frequent pattern {b,f,a,m,c,p} is not frequent in the global list as its support is less than the min_sup.

- The support of pattern {b,f,a,m,c,p}

$$\{b,f,a,m,c,p\} = \frac{40+0}{2} = 20\%$$

- The support of pattern {b,f,a,m,h}

$$\{b,f,a,m,h\} = \frac{60+33.33}{2} = 46.665\%$$

- The support of pattern {b,f,a,m}

$$\{b,f,a,m\} = \frac{80+33.33}{2} = 56.665\%$$

- The support of pattern {b,f,c}

$$\{b,f,c\} = \frac{60+33.33}{2} = 46.665\%$$

- The support of pattern {b,f,p}

$$\{b,f,p\} = \frac{60+0}{2} = 30\%$$

- The support of pattern {b,f}

$$\{b,f\} = \frac{100+50}{2} = 75\%$$

- The support of pattern {b,f,a,c}

$$\{b,f,a,c\} = \frac{40 + 33.33}{2} = 36.665\%$$

- The support of pattern {b,f,a}

$$\{b,f,a\} = \frac{50 + 60}{2} = 55\%$$

- The support of pattern {b,a,c}

$$\{b,a,c\} = \frac{40 + 50}{2} = 45\%$$

- The support of pattern {a,f,m}

$$\{a,f,m\} = \frac{66.66 + 60}{2} = 63.33\%$$

- The support of pattern {b,a}

$$\{b,a\} = \frac{66.66 + 80}{2} = 73.33\%$$

- The support of pattern {a,f}

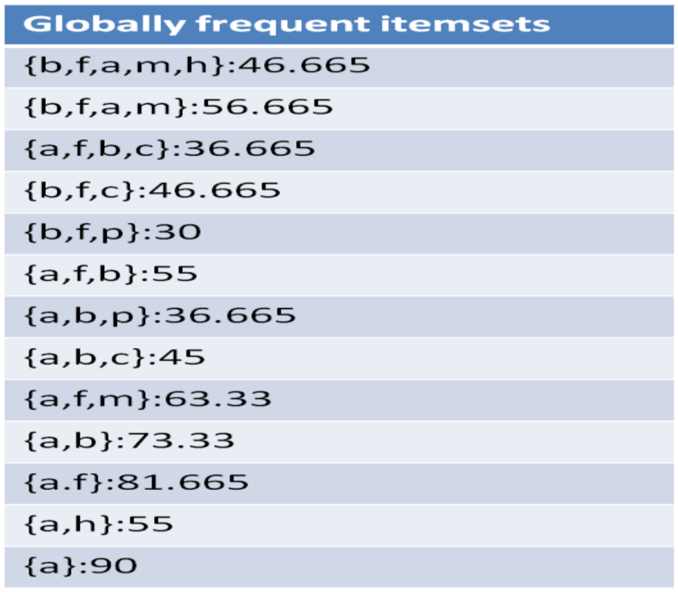
$$\{a,f\} = \frac{80 + 83.33}{2} = 81.665\%$$

- The support of pattern {a,h}

$$\{a,h\} = \frac{50 + 60}{2} = 55\%$$

- The support of pattern {a}

$$\{a\} = \frac{100 + 80}{2} = 90\%$$



Globally frequent itemsets
{b,f,a,m,h}:46.665
{b,f,a,m}:56.665
{a,f,b,c}:36.665
{b,f,c}:46.665
{b,f,p}:30
{a,f,b}:55
{a,b,p}:36.665
{a,b,c}:45
{a,f,m}:63.33
{a,b}:73.33
{a.f}:81.665
{a,h}:55
{a}:90

Fig 5.15 Globally frequent pattern list

In this way we can find the frequent patterns which are globally frequent and maintain a global list. Once the global list is generated the frequent patterns of further datasets will be merged with the global list and the process will be same as we have merged two dataset above to construct the global list.

In this way we can keep our self-updated and can keep track of all the branches and manage the need of each branch remotely. This process has various advantages which we have discussed above.

Parallel frequent mining on large data:

The data is stored from the various sources it may be possible that the data collected becomes large in size. In that case we will apply distributed frequent mining.

Now how we will apply distributed algorithm on the large data.

1. Divide the large data into the smaller data.

2. Now the problem is how to partition the data such that the FP-tree of smaller dataset is independent of the other. So divide the data into small data

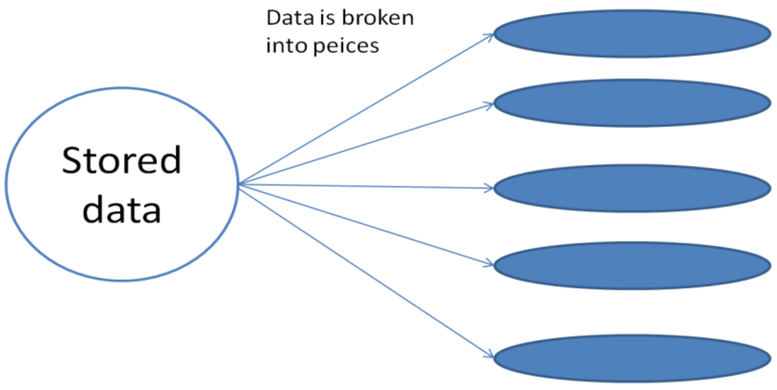
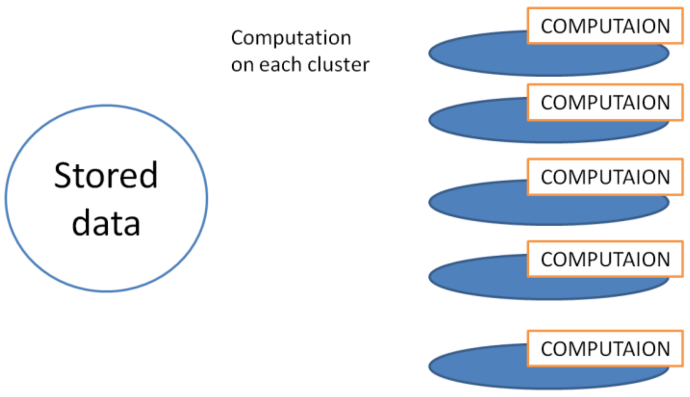


Fig 5.16: Stored data is split

3. On each cluster there will be the processing, which will compute the frequency of each item in the subpart or subset of data. In this way the first scan of dataset in FP-growth which gives the frequency of each item will be computed in parallel by dividing data into smaller part.



5.17: Each split is computed at spate cluster

4. The result computed by each cluster is given back to stored data and the result is combined.

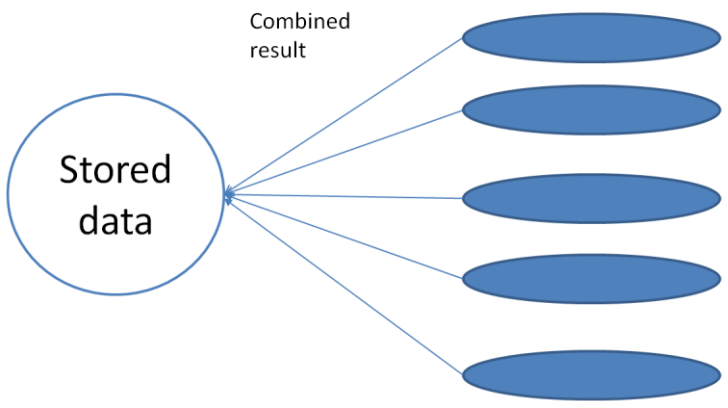


Fig 5.18: Computed result of each cluster is combined at common storage

5. Once we get the frequency list of each item we can eliminate all those items which are not frequent from all the transaction in the dataset. This can also be done in parallel by again dividing the dataset and with each subset pass the frequency list.

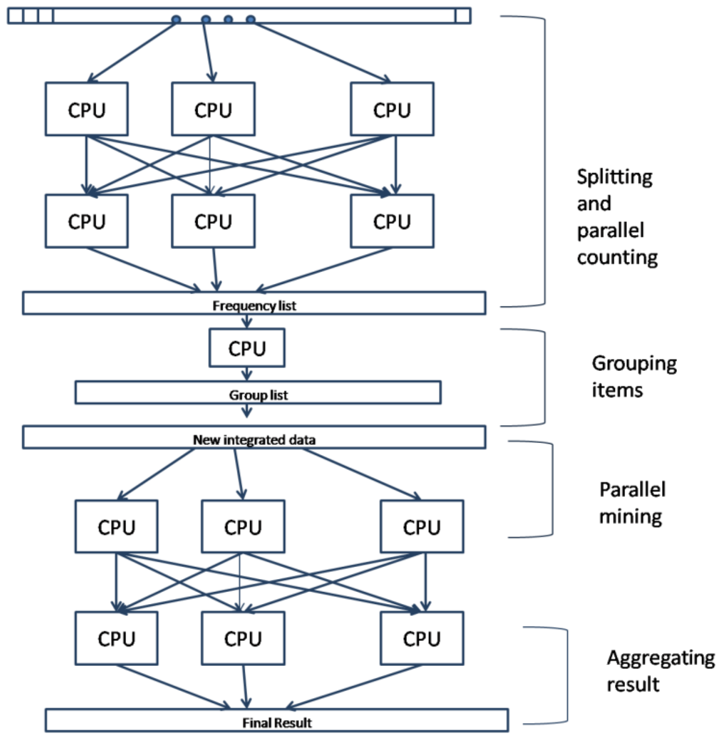


Fig 5.19: Distributed frequent pattern mining using map reduce

Map Reduce Steps:

Step 1: Splitting: Dividing DB into consecutive parts and storing the parts on P different computers. Such partition and allocation of data is called sharding, and each part is called a shard1.

Step 2: Parallel Counting : in this there is a MapReduce pass which is used to count the support values of all items that show in DB. Each shard of DB is the input of a mapper. This step implicitly discovers the items' vocabulary I, which is usually unknown for a huge DB. The result is stored in F-list.

Step 3: Grouping Items: Dividing all the |I| items on F- List into Q groups. The list of groups is called group list (G-list), where each group is given a unique group-id (gid). As F-list and G-list are both small and the time complexity is O(|I|), this step can complete on a single computer in few seconds.

Step 4: Parallel Mining : The key step of PFP. This step takes one MapReduce pass, where the map stage and reduce stage perform different important functions:

- Mapper (Generating group-dependent transactions): Each mapper instance is fed with a shard of DB generated in Step 1. Before it processes transactions in the shard one by one, it reads the G-list. With the mapper algorithm, it outputs one or more key-value pairs, where each key is a group-id and its corresponding value is a generated group-dependent transaction.

- Reducer FP-Growth on group-dependent shards: once all mapper finishes their work and give a group id to the related transactions, the mapper reducer will automatically work on it and categorize each transaction automatically.

- Each reducer instance is assigned to process one or more group-dependent shard one by one. For each shard, the reducer instance builds a local FP-tree and growth its conditional FP-trees recursively. During the recursive process, it may output discovered patterns.

Step 5: Aggregating: the last step is used to aggregate the results generate in Step 4 as our final result.

Above are the steps how we can compute frequent itemset in parallel if we have large dataset by using map reduce method. The mapper will divide the data into cluster and reducer will compute the output on each cluster and gives the result back.

The approach proposed is a real time mining of frequent data itemsets which keeps us updated regarding the present frequent patterns of the market. This approach is suitable for the business in which the decisions have to be done on daily bases data collection. I have shown how this approach will work and how it can help in finding frequent patterns.

Chapter 6

Results

6.1 Comparison between Apriori, FP-growth and Proposed Algorithm (GFP)

We have taken various parameter and matrices to compare the previously proposed approaches with the proposed algorithm (GFP).

Table 1: Comparison between apriori, FP-growth and GFP

Approaches / **Parameters**	**Apriori**	**FP-growth**	**Proposed Algorithm (GFP)**
Technique	Use Apriori property and join and property	It constructs conditional FP_tree and contional pattern base which satisfy min_sup	It divides the data and then construct the FP-tree of each divided part.
Memory utilization	Due to large number of candidate generated it requires large memory	Due to compact structure and no candidate generation it requires les memory	Less as all the patterns are generated in parallel
No. Of scan	Multiple scans for generating candidate set.	Scan the db only twice	Scan the db only twice
Time	Execution time is more as time is wasted in producing candidates every time	Execution time is less than apriori	Execution time is less than FP-growth

6.2 Comparison between Relim, H-mine and Proposed Algorithm (GFP)

We have taken various parameter and matrices to compare the previously proposed approaches with the proposed algorithm (GFP).

Table 2: Comparison between Relim, H-mine and GFP

Approaches / **Parameters**	**H-Mine**	**Relim**	**Proposed Algorithm (GFP)**
Data Structure	H- Struct	Relim Data structure	Fp-tree
Memory utilization	High due to tree like structure	High due to tree like structure	Very high due to parallel tree structure
No. Of scan	Scan the db only twice	Scan the db only twice	Scan the db only twice
Time	Less than relim	moderate	less

6.3 Performance of GFP

We have done the study on various parameter effects on the proposed algorithm GFP. We have analyzed what is the effect on the algorithm by varying the parameter and have noticed the effect of that parameter variation on the proposed algorithm (GFP).

Table 3: Performance metrics for GFP Algorithm

S. No.	**Parameter**	**Effect**
1	Increasing number of cluster	Speed up the performance
2	Increasing min_sup	Time taken to find frequent itemset decreases
3	Increasing size of data set	Time taken to find frequent itemset increases
4	Number of frequent item increases	Time taken to find frequent itemset increases
5	Number of distinct item increases	Time taken to find frequent itemset increases

6.4 Some Experimental Result

We have done many experiments on the various datasets downloaded from various repositories. We have compared it from other algorithms of finding frequent pattern

In this chapter we have analyzed and simulated the performance of approach. We have compared it with different approaches using JAVA(jdk1.7.0) and hadoop environment.

- Configure hadoop by using the manual
    - ➢ http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

- I have taken various dataset from various repositories, the repository is **Frequent Itemset Mining Dataset Repository**:
    - ➢ Chess: http://fimi.ua.ac.be/data/
    - ➢ Mushroom: https://archive.ics.uci.edu/ml/datasets/Mushroom
    - ➢ Connect: http://archive.ics.uci.edu/ml/datasets/Connect-4
    - ➢ Pumsb: http://fimi.ua.ac.be/data/
    - ➢ T10I4D110K: http://fimi.ua.ac.be/data/

The dataset I have taken is from UCI machine learning

- http://archive.ics.uci.edu/ml/

From the UCI repository we have taken the Wholesale customers and other dataset

- http://www.rdatamining.com/resources/data
- http://fimi.ua.ac.be/data/(Frequent Itemset Mining Dataset Repository)
- The following dataset was donated by Tom Brijs and contains the (anonymized) retail market basket data from an anonymous Belgian retail store.
- retail (.gz)

Table 4: Values of the dataset taken

Dataset	**Number of items**	**Average trans size**	**No. of trans**
Chess	75	37	3196
Mushroom	119	23	8124
Connect	129	43	67557
Pumsb	2113	74	49046
T10I4D100K	1000	10	100000

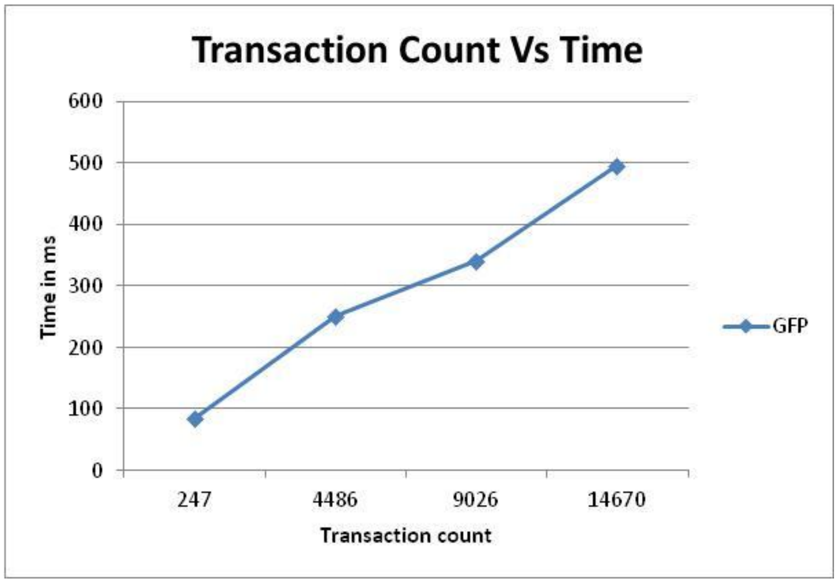


Fig 6.1: Transaction Vs. Time Graph of GFP

The first experiment (figure 6.1) is time vs size of dataset(no. of transactions) . the time did not change much with the increase in data size. The time required to find frequent itemsets is increased gradually with the increase in datasize. The proposed approach is not much effected by the size of dataset. So this approach works well for little large datasets

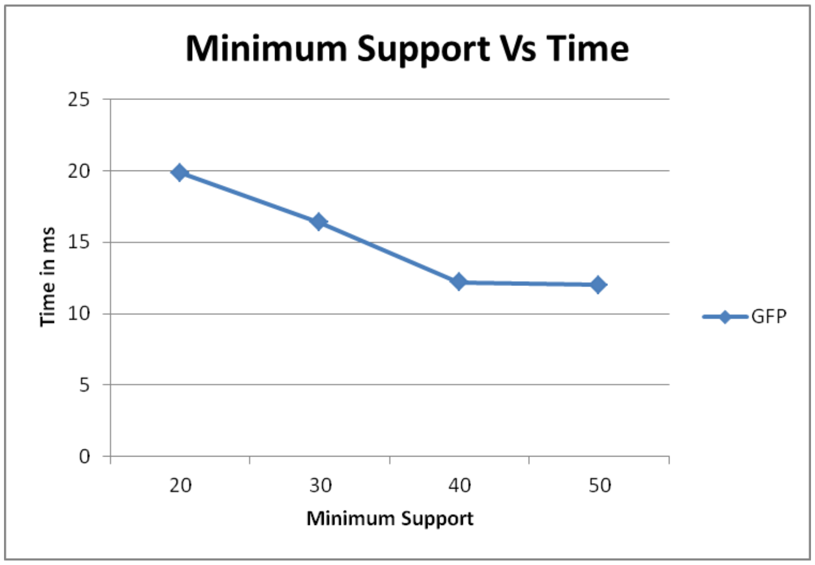


Fig 6.2: Minimum Support Vs. Time Graph of GFP

The second experiment (figure 6.2) which we have done is time vs support i.e. what is the effect on time when we increase or decrease min_support and we find by increasing the min_sup the time required for finding frequent itemset is decrease. The time required to find the frequent itemset is decrease with the min_sup, high the min_sup low the time require to mine.

Chapter 7

Conclusion

As the data is increasing with a great extent, leaving the work for tomorrow will make the work complex and overhead to solve the problem. In the literature review many algorithms of frequent pattern mining is explained.

Apriori is the first algorithm which was used for finding the frequent patterns but it was having many disadvantages like, if we have n-frequent length-1 itemset, then there will be n(n-1)/2 length-2 candidates, so large amount of memory cost is required to have k+1 itemsets or to handle large number of candidates. The second disadvantage is that discovering of length l-frequent pattern requires generating 2^l-2 candidates. So the scalability of Apriori-like algorithms is restricted by the large memory cost.

The problems of apriori-like methods are solved by FP-growth methods to much extent. The multiple scanning of the dataset in apriori is resolved by FP-growth algorithm, to find the frequent patterns it requires only 2-scan of the dataset. The scalability of appriori was restricted by the candidate generation as to find l length pattern we require to have 2^l-2 which is exponential. So on increasing the length of pattern the complexity will increase exponentially. This problem is solved by the FP-growth algorithm to much extent as it does not require candidate generation for generating frequent patterns. In FP-growth algorithm first transactions are compressed and then stored. But still this approach has some disadvantage; it is not much scalable because all of the reconstructed FP-trees with the header table are stored in main memory and we have limited amount of main memory. The FP-tree requires more memory space as it is tree like structure. Handling complex data through FP-growth algorithm will lead to the construction of complex tree and the patterns extracted from them can be complex. If we have complex dataset then the reconstructed FP-trees will be more and they all have to be stored in main memory which is quite costly. So handling complex datasets though this method, is not a feasible solution.

The next algorithm explained is relim known as recursive elimination. This algorithm is inspired by FP-growth method and solved the problem of FP-growth method. In FP-growth method we need to construct the prefix trees and other complicated data structures. Relim does not need to construct the complex structures, it process the transactions directly. The strength of this algorithm is not its speed but the simplicity of its structure. This algorithm is based on step by step elimination of items from the dataset with recursive processing of transaction subsets. This algorithm works in various steps Pre-processing, Transaction Representation, Recursive Processing, Optimization.

After relim H-mine is discussed, in this algorithm there is simple and hyperlinked data structure called H-struct and H-mine takes the advantage of this structure and dynamically adjusts links in the mining process. The advantage of this algorithm is that it has very limited and precisely predictable overhead of space and in memory-based settings it runs really fast.

This algorithm can be scaled up to massive databases by portioning databases, and when dataset becomes dense, we can construct the FP-tree dynamically as mining process part. This is a mining method for frequent itemstes which has given high performance in various kind of data and performs better than the previous algorithm. It works well on massive databases and is space preserving mining method.

Then we have discussed some of the algorithm based on the streaming data and discussed about the map reduce.

The proposed algorithm GPF is a real time mining of frequent data itemsets which keeps us updated regarding the present frequent patterns of the market. This approach is suitable for the business in which the decisions have to be done on daily bases data collection. With the help of an example we analysed that how the algorithm is working. We have done various experiments regarding the approach and it gives better result for real time data.

References

1. Usama Fayyad ,Gregory Piatetsky - Shapiro ,and Padhraic Smyth, “The KDD Process for Extracting Useful Knowledge from Volumes of Data” in Magazine of Communications of the ACM Volume 39 Issue 11, Nov. 1996, Pages 27-34
2. Han, Jiawei, Micheline Kamber, and Jian Pei. Data mining: concepts and techniques. Morgan kaufmann, 2006.
3. Thabet Slimani and Amor Lazzez “Efficient Analysis of Pattern and Association Rule Mining Approaches” in International Journal of Information Technology and Computer Science(IJITCS) Vol. 6, No. 3, February 2014
4. Sotiris Kotsiantis, Dimitris Kanellopoulos, “Association Rules Mining: A Recent Overview”, in GESTS International Transactions on Computer Science and Engineering, Vol.32 (1), 2006, pp. 71-82
5. R. Agrawal and R. Srikant. “Fast algorithms for mining association rules.” in J.B. Bocca, M. Jarke, and C. Zaniolo, editors, Proceedings 20th International Conference on Very Large Data Bases, pages 487–499. Morgan Kaufmann, 1994.
6. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD’98*, pages 94–105.
7. B. Liu, W. Hsu, and Y. Ma, “Integrating classification and association rule mining” in *KDD’98*, pages 80–86.
8. Bj¨orn Bringmann, Siegfried Nijssen, and Albrecht Zimmermann, “Pattern-Based Classification:A Unifying Perspective”
9. Yuhong Li et al. “Discovering Longest lasting Correlation in Sequence Databases” in The 39th International Conference on Very Large Data Bases, August 26th 30th 2013, Riva del Garda, Trento, Italy. Proceedings of the VLDB Endowment, Vol. 6, No. 14 Data Mining and Knowledge 289 (1997Discovery 1, 259–)
10. Srivatsan Laxman, P.S. Sastry and K.P. Unnikrishnan, “Discovering Frequent Episodes and Learning Hidden Markov Models: A Formal Connection” in IEEE transactions on knowledge and data engineering, vol. 17, no. 11, november 2005

11. Frédéric Flouvat, Fabien De Marchi, Jean-Marc Petit, “A new classification of datasets for frequent itemsets” in Journal of Intelligent Information Systems-2010 Volume 34, Issue 1 , pp 1-19
12. Thabet Slimani and Amor Lazzez, “Efficient Analysis of Pattern and Association Rule Mining Approaches” in International Journal of Information Technology and Computer Science (IJITCS), vol.6, no.3, pp.70-81, 2014
13. Rajanish Dass and Ambuj Mahanti ,” An Efficient Real-Time Frequent Pattern Mining Technique Using Diff-Sets “ in Computational Science – ICCS 2005, pp 818-821
14. Paul B. Chou, Edna Grossman, Dimitrios Gunopulos, Pasumarti Kamesam “Identifying Prospective Customers” in Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining Pages 447-456
15. Takuya Oyama, Kagehiko Kitano, “Mining Association Rules Related to Protein-Protein Interactions” in Genome Informatics 11: (2000), pages 358–359
16. Yiming Ma, Bing Liu, Ching Kian Wong, Philip S. Yu, Shuik Ming Lee, “Targeting the Right Students Using Data Mining” in Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining -2000, Pages 457-464
17. Zhongnan Zhang and Weili Wu, “Mining dynamic interdimension association rules for local-scale weather prediction” Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International .
18. R Aggrawal., T Imielinski.,A Swami.. “Mining Association Rules between Sets of Items in Large Databases”. InProc. Int’l Conf. of the 1993
19. ACM SIGMOD Conference Washington DC, USA. J. Han, J. Pei, Y. Yin, and R. Mao. “Mining frequent patterns without candidate generation: A frequent-pattern tree approach”. Data Mining and Knowledge Discovery, 2003
20. U.Chandrasekhar et al., “A Survey of latest Algorithms for Frequent Itemset Mining in Data Stream” in International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970) Volume-3 Number-1 Issue-9 March-2013
21. Aakansha Saxena, Sohil Gadhiya, “A Survey on Frequent Pattern Mining Methods Apriori, Eclat, FP growth”, IJDER, vol 2, issue 1, 2014

22. Christian Borgelt, “Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination” in Proceeding OSDM '05 Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations Pages 66 – 70.

23. B. Goethals "Survey on Frequent Pattern Mining", *manuscript*, 2003

24. Jian Pei et al “H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases” in Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference .

25. Jiawei Han, Hong Cheng ·& Dong Xin ·Xifeng Yan. “Frequent pattern mining: current status and future directions”.Springer (2007).

26. Hua-Fu Li, Chin-Chuan Ho. “incremental updates of closed frequent itemsets over continuous data streams .” Expert Systems with Applications pp.2451-2458, Elsevier 2009.

27. Manku,g., & motwani, R.” Approximate frequency count over data streams” In proceeding of the VLDB conference, pp.346-357 , 2002.

28. Gannella,c.,Han,J.,pei,j.,Yan,X.,& Yu,P.s.” Mining frequent pattern in data stream at multiple time granularities.” Next generation data mining, pp.191-210, (2003)

29. Yun Chi, Haixm Wang. “Catch the moment: maintaining closed frequent itemset over a data stream sliding window.” Springer-Verlag 2006

30. Pauray S.M Tsai.” Mining frequent itemsets in data streams using the weighted sliding window model”, Expert Systems with Applications, Vol. 36,pp. 11617–11625, Elsevier 2009.

31. F.Nori, Mahmood Deypir. “A sliding window based algorithm for frequent closed itemset mining over data streams”. J. Syst. Software (2012), Elsevier

32. Mahmood Deypir, M Sadreddini, & S Hashemi. “Towards a variable size sliding window model for frequent itemset mining over data streams”.Elsevier(2012).

33. http://hadoop.apache.org/

34. Andrew Lampitt, “Hadoop: A platform for the big data era” HPC and HadoopDeep Dive, INFOWORLD.COM

35. http://www.cloudera.com/content/cloudera/en/about/hadoop-and-big-data.html

36. Jeffrey Dean and Sanjay Ghemawat,” MapReduce: Simplied Data Processing on Large Clusters” in Magazine Communications of the ACM - 50th anniversary issue: 1958 – 2008 Volume 51 Issue 1, January 2008, Pages 107-113.
37. http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/