# APPLICATION DEVELOPMENT FOR IMPROVING WEB SITE USABILITY BY WEB MINING METHODS

**A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of**

**MASTER OF SCIENCE**

**in Computer Engineering**

**by
Merve Ceren TÜKER**

**June 2018
İZMİR**

We approve the thesis of **Merve Ceren TÜKER**

**Examining Committee Members:**

---

**Prof. Dr. Oğuz DİKENELLİ**
Department of Computer Engineering, Ege University

---

**Assoc. Prof. Dr. Belgin ERGENÇ BOSTANOĞLU**
Department of Computer Engineering, Izmir Institute of Technology

---

**Dr. Tuğkan TUĞLULAR**
Department of Computer Engineering, Izmir Institute of Technology

**28 June 2018**

---

**Assoc. Prof. Dr. Belgin ERGENÇ BOSTANOĞLU**
Supervisor's Department
Izmir Institute of Technology

---

**Assoc. Prof. Dr. Yusuf Murat ERTEN**
Head of the Department of
Computer Engineering

---

**Prof. Dr. Aysun SOFUOĞLU**
Dean of the Graduate School of
Engineering and Sciences

# ACKNOWLEDGMENTS

I would first like to thank my advisor Assoc. Prof. Dr. Belgin Ergenç Bostanoğlu of the Department of Computer Engineering at Izmir Institute of Technology. She consistently steered me in the right direction while allowing this study to be my own work. Without her invaluable participation and input, this accomplishment would not have been possible.

Finally, I would like to express my profound gratitude to my family and my partner for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

# ABSTRACT

APPLICATION DEVELOPMENT FOR IMPROVING WEB SITE USABILITY BY WEB MINING METHODS

The explosive growth in website traffics and website usage data has resulted in the amount of valuable information contained to have a similar uptrend in web usage logs. With the increasing competition between websites, mining web usage logs to discover meaningful information is needed more than ever. Web usage mining is the procedure of using data mining methods to discover insightful patterns in web usage logs. The discovered information helps understand how users behave on the website and their needs. One of the most popular algorithmic approaches of pattern mining on web usage data is the Fp-growth algorithm. For larger volumes of data, the algorithm is generally applied to execute in parallel. Measuring and comparing performances of applications is difficult, because the algorithm performs different on usage logs with different characteristics. The characteristics of usage logs are highly related with the type of the website. In this paper, we have investigated how different characteristics of web usage logs effect the performance of the parallel Fp-growth algorithm. Five datasets with varying log characteristics were used in order to represent different business models. The results suggest that the performance is highly correlated with the number of items, number of frequent items, transaction length, similarity between frequent patterns, minimum support value and size of the log file.

# ÖZET

## WEB SİTESİ KULLANILABİLİRLİĞİNİ İYİLEŞTİRMEK İÇİN WEB MADENCİLİĞİ YÖNTEMLERİ İLE UYGULAMA GELİŞTİRİLMESİ

Web sitesi trafik ve kullanımındaki hızlı artış, web sitesi kullanımına ilişkin değerli bilgiler içeren kayıtlarda da benzer bir yükselişe sebebiyet vermiştir. Web siteleri arasındaki artan rekabetle beraber, kullanım verilerine veri madenciliği metodolojileri uygulayarak anlamlı bilgileri keşfetmek her zamankinden önemli bir hal almıştır. Web kullanımı madenciliği (web usage mining), veri madenciliği tekniklerini kullanım verileri içerisinde mevcut olan desenleri keşfetmek üzere uygular. Bu desenler web temelli uygulamaları ihtiyaçlarını daha iyi anlayarak web sitesi kullanılabilirliğini geliştirebilmek üzere kullanılabilecek değerli bilgiler içermektedir. Web kullanımı verisi üzerinde desenleri keşfetmede en sıklıkla kullanılan algoritmalardan biri Fp-growth algoritmasıdır. Büyük hacimde veriler için algoritma genellikle paralel çalışacak şekilde uygulanır. Uygulanan algoritmanın performansı kullanılan verinin karakteristik yapısıyla yakın ilişkili olduğundan, gerçekleştirilen farklı uygulamaların performansını karşılaştırmak güç olmaktadır. Verinin karakteristik özellikleri de web sitesinin türü ve amacına göre değişkenlik göstermektedir. Bu çalışmada, web kullanımına ilişkin kayıtların farklı karakteristik yapılarının paralel Fp-growth algoritmasının performansına olan etkileri incelenmiştir. Uygulama çeşitli iş modellerini temsil etmek üzere farklı karakteristiklere sahip beş farklı veri seti üzerinde çalıştırılarak performans analizi yapılmıştır. Sonuçlar algoritma performansının veri setindeki toplam eleman sayısı, özgün eleman sayısı, sıklık ölçümünde alt eşik (minimum support threshold) değeri, alt eşiğin üzerinde kalan ’sık’ eleman sayısı, sık görülen desenler arasındaki benzerlik oranı, tek bir işlemin (transaction) uzunluğu ve veri setinin boyutu gibi özelliklerle yakından ilişkili olduğunu göstermektedir.

# TABLE OF CONTENTS

LIST OF FIGURES .... viii

LIST OF TABLES .... x

LIST OF LISTINGS .... xi

CHAPTER 1. INTRODUCTION .... 1

CHAPTER 2. BACKGROUND .... 4
- 2.1. Knowledge Discovery in Databases .... 5
- 2.2. Data Mining .... 6
- 2.3. Functions and Application Fields of Data Mining .... 7
- 2.4. Frequent Pattern Mining .... 10
  - 2.4.1. Join Based Algorithms .... 12
  - 2.4.2. Vertical Mining Algorithms .... 14
  - 2.4.3. Tree Based Approaches .... 14
  - 2.4.4. Maximal and Closed Frequent Itemset Mining Algorithms .... 19
  - 2.4.5. Parallel Fp-tree Based Algorithms .... 19
- 2.5. Web Data and Web Mining .... 24
- 2.6. Taxonomy of Web Mining .... 26
- 2.7. Web Usage Mining .... 28
  - 2.7.1. Association Rules in Web Usage Mining .... 31
  - 2.7.2. Sources of Data for Web Usage Mining .... 32

CHAPTER 3. RELATED WORK .... 35
- 3.1. Serial Fp-growth based Web Usage Mining Algorithms .... 35
- 3.2. Parallel Fp-growth based Web Usage Mining Algorithms .... 36

CHAPTER 4. PARALLEL FP-GROWTH IMPLEMENTATION FOR WEB USAGE MINING .... 44
- 4.1. FP-Growth .... 44
- 4.2. Parallel FP Growth Using Apache Spark .... 48
  - 4.2.1. Preparation .... 51

4.2.2. Parallel Fp-growth ........ 51
4.2.3. Aggregation ........ 53

CHAPTER 5. EMPRICAL STUDY ........ 55
5.1. Simulation Environment ........ 55
5.2. Datasets ........ 55
5.2.1. NASA Weblogs ........ 56
5.2.2. Synthetic Datasets ........ 57
5.3. Experiments ........ 59
5.3.1. NASA Weblog Results ........ 59
5.3.2. Synthetic Dataset Results ........ 64
5.4. Discussion ........ 68

CHAPTER 6. CONCLUSION ........ 70

REFERENCES ........ 73

# LIST OF FIGURES

**Figure** **Page**

Figure 2.1. Phases of knowledge discovery…………………………………….……...… 5

Figure 2.2. Simple linear partitioning on loan dataset……………………………… 7

Figure 2.3. Simple linear regression where total debt is a linear function of income. 8

Figure 2.4. An example clustering visualization over loan dataset……….………... 9

Figure 2.5. Fp Tree structure……………………………………………….………. 17

Figure 2.6. Pattern growth……………………………………………………..… 18

Figure 2.7. Prefix paths ending with "e" and "de"………………….…………… 18

Figure 2.8. The overall PFP framework……………...………………………….. 21

Figure 2.9. An Fp-Tree…………………………………………………...…… 23

Figure 2.10. Repartition function of DFPS………………………………………… 24

Figure 2.11. Web mining taxonomy………………………………….…………..…… 25

Figure 2.12. Web content mining subtypes……………………………………… 26

Figure 2.13. Web usage mining steps …………………………….……………... 29

Figure 3.1. Sessionization phase of MIP-PFP……………………………………… 37

Figure 3.2. Deducting MIPS elements from transactions……….……..………… 38

Figure 3.3. Removal of non-MIPS and 1-item elements………………………… 39

Figure 3.4. Scalability of S-FPG compared to PFP………………………………… 44

Figure 4.1. A sample db ordering process…………………………………….…… 46

Figure 4.2. Fp-tree structure for sample db……………………………………… 47

Figure 4.3. Spark architecture……………………………………………………. 49

Figure 4.4. Parallel Fp-growth phases…………………………………………… 50

Figure 4.5. Parallel Fp-growth algorithm…………………...………………… 52

Figure 4.6. Aggregation……………………………………...………………… 53

Figure 5.1. Execution time with support change (0.01-0.09)…………………..…… 61

Figure 5.2. Log file sizes for different transaction volumes………………...….… 61

Figure 5.3. Execution time with transaction volume change…………………….… 62

Figure 5.4. Execution time with file size change…………………...…….…..… 63

Figure 5.5. Execution time with file size and executor number change…….……… 64

Figure 5.6. Relative execution time with executor number change …………..…..… 64

Figure 5.7. Execution time with support change (0.01-0.06)………..…………………… 65

Figure 5.8. Log file sizes for different transaction volumes…………….………..… 66

Figure 5.9. Execution time with transaction volume change……..…….……...…… 67

Figure 5.10. Execution time with file size change……..……….………….....……..… 68

# LIST OF TABLES

**Table** **Page**

Table 2.1. Frequent pattern mining algorithms……………………………….……… 11

Table 2.2. Transaction database……………………………….……………………… 16

Table 3.1. Statistical details of NASA dataset logs………………………..…..… 39

Table 3.2. Statistical information computed of NASA dataset logs after
preprocession…………………………………….……….………….……. 40

Table 4.1. Conditional Fp-trees for sample db……………………….………..… 48

Table 5.1. Number of frequent itemsets for each minimum support………………. 57

Table 5.2. Characteristics of the logs from different platforms……………………. 57

Table 5.3. Characteristics of synthetic web usage logs…………………………..… 58

Table 5.4. Number of frequent itemsets for each minimum support……...……..… 60

# LIST OF LISTINGS

**Listing** **Page**

Listing 2.1. A web server log…………...…………...…...………………………… 32

# CHAPTER 1

# INTRODUCTION

The available sources, volume, availability and variety of data has grown exponentially in the last decade. This growth has resulted in World Wide Web being used more than ever, causing an explosive growth in website traffics and the amount of information consisting user activities in and between pages. The popularity of www has caused a competition between websites with similar content or goals, making becoming the preferred website among alternatives and gaining user attraction a primary concern. The competition between websites has resulted in more emphasis upon the design of the web pages. Each website offers users with a form of navigation among available contents, and an interface to interact with the website. The design decisions used to be made implicitly by the person that actually implements the functionalities. When design decisions are made without concerning how users actually perceive and interact with a page, the results are rarely optimal and the risk of losing precious user attention arises. It is stated that an average human attention span has shown a significant drop from 12 seconds to 8 seconds between years 2000 and 2013 (Canada, 2015). This dramatic drop is mainly caused by the fact that connected consumers have gotten better at doing more with less time via shorter bursts of high focus. This shrink in attention spans of users, and all other tendencies mentioned above, result in an increased demand of a user-friendly, easy to navigate and understandable website structure to increase efficiency in all user operations. Data that is obtained beforehand, such as web server logs, could be inspected to detect possible weaknesses in design, helping to propose better solutions to improve user experience.

Since the volume of web server log data is enormous, the proposed approach should require as little manual interference as possible so that the process can be automated. Automation is advantageous both for reusability and for obtaining realistic results. For this purpose, we have investigated some of the previous approaches for frequent pattern mining, and propose a solution that is based on Fp-growth algorithm, which is a scalable and efficient approach (Han et al., 2000). Fp-growth algorithm represents data with the help of a prefix tree and a header table. The algorithm first performs a scan of the database to acquire support values of all frequent items. This information is then used to order items within each transaction accordingly. The ordered set of transactions are then used for creating the tree structure. For the creation of the tree, it performs a second scan of the db and incorporates an item header table during generation phase. This header table enables knowing the frequency of items without doing any further db scans and results in a faster mining phase once the tree is generated. The algorithm then performs depth first traversal for mining frequent patterns over the tree based compact data structure. The compact tree structure reduces space consumption: the Fp-tree provides a compressed version of the db at hand. Finally, using a header table structure reduces the costs for finding and counting items. This header table structure is used for accessing the frequency of items without doing any other db scans and results in better performing mining phase once the tree is generated. Overall, with the removal of the candidate generation, candidate testing and repetitive db scans, Fp-growth algorithm performs better than its ancestors. The Fp-growth algorithm offers a divide and conquer based approach, making it easy to parallelize and the parallel execution offers better scalability. To execute Fp-growth algorithm in a parallel manner, the db is sharded into sub pieces using conditional sub-trees to execute independently.

After processing sub-trees, the results are aggregated to acquire a final mining result. Since web usage data tends to be huge, we have implemented the solution to execute in parallel to be able to process higher volumes of data with less time consumption (Wang et al., 2008). For managing the parallel execution, we have used the most popular cluster computing framework, Apache Spark(TM).

By applying the Fp-growth algorithm to the domain of web usage mining, it is expected to discover meaningful insights about website usage. This information is to be used for achieving a better website structure, utilization of resources and a drop in maintenance costs. Another primary goal is to eventually enhance user engagement and user experience. There has been many studies using Fp-growth for the task of web usage mining. However, it is hard to evaluate the performance of an application since the relationship between the characteristics of data at hand and success of the mining process have not been inspected. For this purpose, we applied the algorithm on five different datasets with varying characteristics. We used one real dataset and generated four synthetic datasets, each designed to represent a different business model. The algorithm is applied and results have been observed in accordance with the dataset's characteristics. The experiments suggest that the performance of the algorithm depends on the number of items, number of frequent items, length of transactions correlation between patterns, minimum support value and file size. The same principles can be applied to evaluate different business models.

The remaining of the thesis is organized as follows. Chapter 2 provides a background; describing data mining, its functions and application fields, web mining and web data; web usage mining and it's data sources. Chapter 2 also explains various algorithms used for frequent pattern mining. Chapter 3 discusses the existing efforts in Web Usage mining and parallelizing Fp-growth algorithm. Chapter 4 presents the implementation of parallel Fp-growth algorithm. Chapter 5 provides a performance study. Chapter 6 presents a final overall statement of our study and discusses some of the future research directions.

# CHAPTER 2

# BACKGROUND

Throughout the years, many approaches to the problem of discovering unknown, interesting information from larger sets of data have been proposed. This process is referred to as knowledge discovery. Knowledge discovery uses data mining methodologies to extract unknown information. Knowledge discovery in databases is examined under section 2.1. Data mining is a sub field of computer science that consists the discovery of interesting and useful patterns and relationships within large datasets that uses data analysis algorithms. Data mining combines tools from various fields such as artificial intelligence, statistics and database management. Data mining can be briefly described as having six main functions: Classification, Regression, Clustering, Summarization, Dependency Modeling and Change/Deviation Detection (Fayyad et al., 1997). Data mining is examined under section 2.2, the application fields of data mining are examined under section 2.3. Pattern mining is an important sub field of data mining which utilizes data mining methods for the purpose of discovering interesting, unexpected and useful patterns that provide new insights regarding the dataset of interest. Pattern mining methods can be focusing on various different types of patterns based on the goal of implementation such as discovering frequent patterns, rare patterns, patterns with high confidence, top patterns, meaningful unknown associations, general tendencies, outliers or exceptions, sequences of patterns and so on. One of the most popular approaches is to discover frequent patterns by specifying a minimum threshold. Frequent patterns are especially meaningful for the domain of web usage because of the high volume of website traffics. Frequent pattern mining is essential for mining associations, correlations, causality, sequential patterns and many other tasks. These approaches are briefly described under section 2.4. A comparative discussion consisting various algorithmic approaches are also given under section 2.4, briefly categorized as join based algorithms, vertical mining algorithms, tree based approaches, maximal and closed frequent itemset mining algorithms. Data mining is an interdisciplinary field that has many application domains, one of which is web data. Applying data mining techniques on web data is called web mining. Web mining is discussed under section 2.5. It has three sub fields based on which type of web data is examined: website content, website structure or web usage data. A brief taxonomy consisting these sub fields is given in section 2.6. Mining web usage patterns consists the most beneficial information for enhancing the website-user interactions, so web usage mining is discussed in detail and a general terminology of the application field is introduced under section 2.7. Additionally, the concept of association rules in the domain of web usage mining and sources of data for web usage mining are given in this section.

## 2.1. Knowledge Discovery in Databases

Knowledge Discovery in Databases (KDD) is the general process that aims to reveal previously unknown, valuable information hidden within large databases. It can also be briefly defined as a "higher level" specific application of data mining that utilizes some of its methods. Discovery of interesting knowledge is performed by making use of preprocessing data, transforming the data at hand, and by specifying thresholds and parameters. KDD combines a diverse range of domains, like data mining, such as database management, statistic, data visualization, artificial intelligence, pattern discovery and data visualisation.

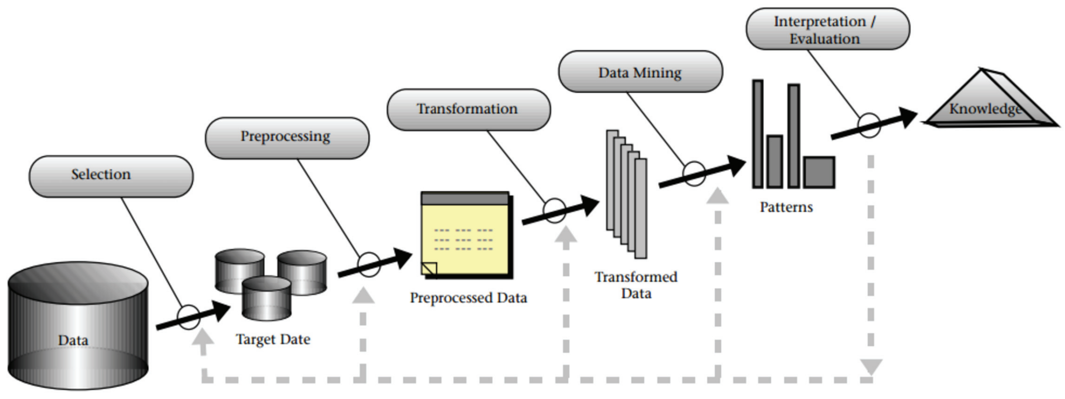


Figure 2.1 Phases of knowledge discovery
(Source: Fayyad et al., 1997)

Figure 2.1 shows the general steps of knowledge discovery (Fayyad et al., 1997) that follow the selection of a discovery goal can be briefly listed as:

- Selection: Selecting relevant data based on the scope of the discovery goal at hand.
- Preprocessing: Combining separate data sources if there are any, and removing noisy and inconsistent data.
- Transformation: Transforming data into a suitable form to perform mining and evaluation tasks.

- Data mining: Choosing a suitable data mining algorithm and applying it on the working space. The terms data mining and knowledge discovery are closely related and often used instead of one another: data mining is one of the most important processes of knowledge discovery.

- Interpretation/Evaluation: Interpreting the patterns into domain knowledge and translating the extracted patterns into human understandable, relevant terms.

## 2.2. Data Mining

Data mining is the core process of knowledge discovery. It consists the extraction of interesting and insightful patterns from large data sets using data analysis and discovery algorithms. The procedure is expected to discover a particular subset of patterns within given data. Since in most domains the space of patterns is infinite, the operation highly depends on declaring limitations. These limitations could be incorporated either by working on a sub-space of information (e.g. only studying on relationships between particular columns of a table, on a specific set of tables within a DB, etc.) or by declaring some sort of threshold to degrade the work space at hand. Data mining aims to describe data in novel ways and can also be used for predicting future trends in data. Description aims to find new structures or patterns within data that approach to it in a way that has not been yet expressed, which enables approaching the data from fresh angles. Prediction focuses on using fields that exist within a database to predict future or unknown values of variables of interest. Since these two goals complement each other, most models use a combination of them; meaning some of the descriptive models also offer prediction to a degree, and some of the predictive models also present an understandable description of the data. The distinction and balance between two approaches, however, is important when setting a discovery goal. After setting a discovery goal, it is possible to adopt from a variety of different methods in order to achieve it. The selected discovery goal highly depends on the scope and the application field.

Data mining is an interdisciplinary field that is closely coupled with statistics, natural language processing, database systems, big data, machine learning and artificial intelligence. It is considered a key phase of knowledge discovery (Fayyad et al., 1997), the overall process of deducting (enumerating) useful patterns and models from data at hand. Some of the main application areas of knowledge discovery includes marketing, telecommunications, finance, fraud detection, internet agents and manufacturing. Various application approaches using Data Mining methodologies will be inspected under section 2.3 in detail.

## 2.3. Functions and Application Fields of Data Mining

Mining data allows the exploration of the dataset from different angles, enabling new categorizations, summarizations, descriptions and predictions of data sets and relationships between them (Han and Kamber, 2006). Data mining is described to have six main functions: "Classification, Regression, Clustering, Summarization, Dependency Modeling and Change/Deviation Detection". (Fayyad et al., 1997).

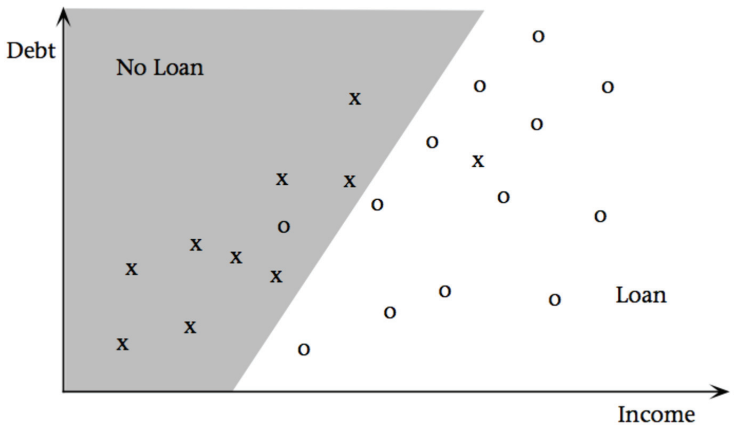


Figure 2.2 Simple linear partitioning on loan dataset
(Source: Fayyad et al., 1997)

**Classification** is the phase that classifies (maps) a certain item into one of the predefined classes (Weiss and Kulikowski, 1991). Financial market trend analysis (Apte and Hong, 1996) and automatic detection of objects of interest within a large image database (Fayyad et al., 1996) can be listed as examples that use classification techniques as part of a larger knowledge discovery application. Figure 2.2 shows a simple linear partitioning on a loan dataset.

**Regression** is the process of mapping an item from the dataset to an item with a real value or a representation of another value. Some examples of regression applications are prediction of the amount of bio-mass using remotely sensed microwave measurements, estimating the probability of survival for a patient given his/her diagnostic results or predicting customer demand for a new product. Figure 2.3 shows a simple linear regression where total debt is a linear function of income. Fit is poor because of the weak correlation between the two variables of choice.

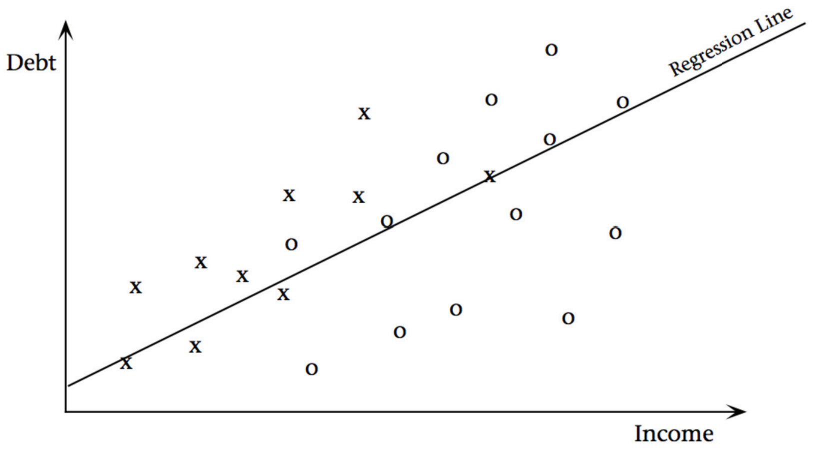


Figure 2.3 Simple linear regression where total debt is a linear function of income (Source: Fayyad et al., 1997)

**Clustering** can be briefly described as identifying sets of groups, also referred to as clusters, within a given set of data. It is a common descriptive task in data mining (Jain et al., 1988) (Titterington et al., 1985). A sample representation of data clustering over the loan dataset can be seen in Figure 2.4. Some examples of clustering usages within knowledge discovery applications are: discovering sub-populations of customers within a marketing dataset and identifying subcategories of spectra from infrared sky measurements (Cheeseman and Stutz, 1996). Another task that is closely related to clustering is the task of probability density estimation, which is used for describing techniques for estimation of the joint multivariate probability density function of all the variables or fields within a database (Silverman, 1986). Probability densitiy estimation will not be discussed in detail here, since it is not within the scope of this paper.

**Summarization** is the process of representing given data within a compact expression. An example of summarization process is tabulating the standard and mean deviations for all fields within a given data set. More sophisticated methods include deriving some summary rules, multivariate visualization techniques and the discovery of functional relationships between variables (Zembowicz and Zytkow, 1996). These techniques are often used in coordination with the fields of exploratory data analysis and automated report generation.

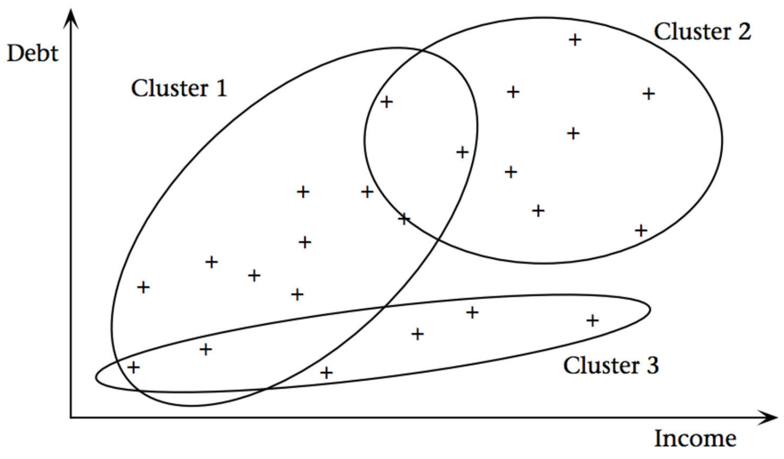


Figure 2.4 An example clustering visualization over loan dataset
(Source: Fayyad et al., 1997)

**Dependency Modeling** process aims to finding a description that shows dependencies between variables. The compositions that represent dependencies are also called dependency models. These models can be listed under two different levels. *The structure level:* variables are locally dependent on one another and are usually represented in a graphical form. *The quantitative level:* also specifies how coupled dependencies are, using numerical values. Probabilistic dependency networks use conditional independence to specify the model and the correlations in a structural way to specify how strong each dependency is (Glymour et al., 1987). Probabilistic dependency networks often find applications such as development of medical expert systems that borrow probabilistic analysis approaches, information retrieval, and modeling of the human genome.

**Change/deviation detection** is the subtask that aims to detect the most significant changes from previously measured normal values for the given dataset. Detection of anomalies has various application areas such as security, medical applications, statistic analysis systems etc.

The categories might or might not be mutually exclusive. They can also contain a richer form of representing data, such as a hierarchical structure or overlapping sub categories. All of the categories given above are intended to decompose interesting patterns from uninteresting ones. Which patterns are ’interesting’, however, can be defined differently from various perspectives. Some approaches consider rare patterns, some approaches consider items that appear frequently, some consider items with a high confidence, some consider top patterns as interesting. One of the most popular approach is to consider items that appear above a certain threshold as frequent, thus interesting. This concept is also called "frequent pattern mining", which will be examined in detail within section 2.4.

## 2.4. Frequent Pattern Mining

Frequent patterns are sets of items or structures that have a high frequency of occurrence. A threshold specified by the user is generally used in order to determine whether an itemset is frequent or not. This limit is generally referred to as a minimum support threshold. The itemsets that have a higher frequency or support value are called "frequent itemsets". Similarly, patterns that have a high frequency are referred to as frequent patterns. The term "frequent pattern mining" was initially used for analyzing and discovering rules within market basket dataset. (Agrawal et al., 1993)

The very first approach to tackle the problem of frequent pattern mining was proposed in 1994 (Agrawal and Srikant, 1994), a join-based algorithm called Apriori. The Apriori algorithm performed breadth-first search over a horizontal hash tree. It took advantage of the "downward closure property". Downward closure property states that subsets of a frequent itemset are all frequent themselves. The algorithm makes use of this property for pruning the itemset lattice. This was also known as Apriori-pruning. The algorithm generated candidate sets and performed multiple db scans to check the validity of candidates. Following Apriori, another join based approach was proposed to enhance the previous design. The new algorithm, called eclat, used a vertical layout to represent data (Zaki et al., 1997). Eclat algorithm used TID lists instead of calculating support values and intersections instead of joining. This approach required less space than Apriori and was faster if datasets were small. It also offered very fast support count, but had the risk of intermediate TID-lists getting too large for memory. The main bottleneck of this design was that the longer TID-lists got, the more costly it was to calculate frequencies. Another algorithm that offered a vertical layout based approach was Viper (Shenoy et al., 2000). Viper algorithm associates each transaction with a column and denotes values for each item based on whether it exists within a dataset or not.

To further reduce the repetitive calculation cost of support values and the cost of generation of candidate itemsets, tree projection based algorithms were offered. These approaches traded memory cost with calculation time and acquired faster results. The structure of the generated enumaration tree was used to avoid re-doing calculations. A prefix tree based approach, Fp-growth algorithm was proposed (Han et al., 2000). The algorithm performed depth first traversal for mining frequent patterns over the tree based compact data structure. First it scanned the database, to acquire all frequent items and order each transaction accordingly. Then it created the tree using this data at had. For the creation of the tree it performed a second scan of the db and incorporated an item header table during generation phase. This header table enabled knowing the frequency of items without doing any other db scans and resulted in a much faster mining phase once the tree is generated. This algorithm offered a divide and conquer approach, and with the removal of the candidate generation, candidate testing and repetitive db scans; resulted in better performance than its ancestors. Other advantages were that the structure at hand was compact, it provided node-link property caused by the structure's nature (all transactions containing a certain frequent item could easily be acquired with minimum cost). The greatest bottleneck of this approach was that it depended on memory. If the db at hand was too large for the tree to fit in memory, optimizations were a must.

Table 2.1 Frequent pattern mining algorithms

| **Algorithm** | **Join Based** | **Depth-first or Breadth first** | **Vertical Layout** | **Tree Based** | **Closed / Maximal Patterns** |
|---|---|---|---|---|---|
| Apriori | √ | breadth | | | |
| DHP | √ | breadth | | | |
| Eclat | √ | | √ | | |
| Viper | | breadth | √ | | |
| AIS | | breadth | | √ | |
| TreeProjection | | usually depth | | √ | |
| Fp-growth | | usually depth | | √ | |
| Mafia | | depth | √ | | maximal |
| MaxMiner | | | | | maximal |
| Closet | | | | √ | closed |
| Charm | | | √ | | closed |

Following Tree Projection based approaches, algorithms that focus on closed or maximal itemsets were proposed. Following tree based approaches, algorithms that used compact representations of frequent patterns such as closed and maximal frequent itemsets were proposed. A vertical layout and depth first traversal based approach called Mafia was proposed in 2001 (Burdick et al., 2001). The Mafia algorithm focused on finding maximal frequent itemsets. This algorithm used intersections over vertical bitmap structure instead of joins to detect itemsets, and used bitwise operations upon the vertical representation to calculate the supports. This approach was able to work efficiently for larger databases and longer itemsets (as long as the entire data was able to fit in memory), but did not perform as good for short itemsets. The algorithm used Apriori principle to prune data to increase efficiency. Other maximal frequent itemset mining algorithms were proposed, such as the MaxMiner algorithm. Algorithms such as Charm and Closet were proposed for finding closed itemsets.

Mining frequent patterns is still the most studied phase of data mining. It is implemented in many different applications used for mining associations (Agrawal and Srikant, 1994) (Klemettinen et al., 1994), causality (Silverstein et al., 1998), correlations (Brin et al., 1997), sequential patterns (Agrawal and Srikant, 1995).

The general characteristics of algorithms used for mining frequent patterns that are mentioned within this section, as seen in Table 2.1, such as join based, vertical layout based, tree structure based and maximal or closed pattern based algorithms will be discussed in detail in following sections.

### 2.4.1. Join Based Algorithms

Join-based algorithms use join operations in order to generate (n + 1)-candidates from n-patterns at hand that are known to be frequent. Following the generation phase, the frequency of the new candidates are checked by using the transaction database. This approach has two major costs: the generation of many candidates can cause an explosive growth at the data at hand. Second, validating each set of candidates by hitting the transaction database repeatedly is a huge overhead.

***Apriori Algorithm*** is one of the earliest solutions to the problem of mining frequent patterns. Apriori algorithm uses join operations to generate new candidates from frequent patterns at hand. The initial studies are mostly formed based on Apriori algorithm (Agrawal and Srikant, 1994). Apriori method is the simplest form of join-based algorithms. The algorithm approaches data from a level-wise perspective, where all frequent itemsets of length $(n + 1)$ are generated by using previously generated length n itemsets. Apriori algorithm takes advantage of the "downward closure property", which denotes that "subsets of frequent patterns are also frequent". Pairs of frequent n-patterns that have at least n-1 common items are needed in order to perform the join operation to generate new candidates. Consider two patterns of 3, $\{i1, i2, i3\}$ and $\{i1, i2, i4\}$ are frequent. Since both of them are frequent and they have $n - 1 = 2$ common items, they will be joined while discovering patterns with length 4. After a join operation on these two patterns, a candidate pattern of length 4, $\{i1, i2, i3, i4\}$, can be obtained. The reason that the generated pattern is called a "candidate" is that it is not yet validated to be frequent. The frequency is checked by using the transaction database to count the support value of the pattern. Since support count is repeated for every candidate, the efficiency of this phase directly effects the efficiency of the overall algorithm. Additionally, it is possible to obtain the same candidate from join operations of different $n-1$ patterns. Since this causes an increase in redundancy, a rule is imposed, stating that items in itemsets should be stored based on a lexicographic order, and that two itemsets are only to be joined when their first $(n-1)$ items are the same and have the same order. Finally, it is possible to implement the Apriori algorithm recursively to work for each level of candidates. The Apriori algorithm executes as follows: Firstly, the candidate patterns are generated by using join operations on patterns from the previous level. Secondly, the generated candidates are pruned by making use of the downward closure property. Finally, the support values of candidates at hand are checked to see if they are frequent or not. The algorithm repeats the three steps until there are no more frequent candidates to be generated. The main idea behind the design of the Apriori algorithm is to make use of an anti-monotone heuristic: "none of the super patterns of a non-frequent pattern can be frequent". Since the performance is mostly hindered by managing sets of candidates, this idea reduces the size of valid candidate sets, in order to enhance the performance. This approach suffers from edge cases such as a very large number of unique items, very long patterns, very small support thresholds. Additionally, an algorithm based on Apriori is prone to suffer from handling candidates and managing support counts. It is computationally intensive to manage candidate sets that could be huge, and repetitive db scans still need to be performed at each step. Optimizations that are proposed for the Apriori algorithm tend to address these bottlenecks.

***DHP Algorithm*** This method, "direct hashing and pruning", was proposed following the Apriori algorithm. It proposes enhancements to the bottlenecks of Apriori algorithm. One of the optimizations is to apply a pruning mechanism to the candidates at each step in order to reduce the size of the candidate sets that need to be managed. The algorithm proposes using a hash table to manage the support values of subsets. The support values are tracked and stored in the hash table during the counting phase of the previous set of candidates. Another optimization is implemented by trimming the transactions again in order to increase the efficiency of managing the support values of candidates. This optimization makes use of the fact that if at least n-frequent itemsets do not contain an item, then the item will not appear in any of the frequent itemsets of the next level (n+1). Therefore, items that do no exist in n-frequent itemsets, for instance, can be trimmed. This optimization results with reduced width to increase performance. Note that this algorithm comes with an overhead from the additional data structures, causing the efficiency to drop for longer patterns.

### 2.4.2. Vertical Mining Algorithms

These approaches represent the transactions vertically in order to increase the efficiency of counting items. Vertical representation generally means expressing the database in an inverted list. These lists are also called tidlists. With this method, transactions that contain a particular item can be seen rather than a transaction-based representation. Another key optimization that comes with this approach is that while counting supports, instead of computing the support of an n-itemset, an n-way intersection of the respective lists of the items within the set is performed. Secondly, for computing the support of an n-itemset, an intersection between the corresponding lists of two n-1 itemsets can be performed. Some of the well known algorithms that are based on vertical pattern mining approach could be listed as : Monet, partition, Eclat, VIPER algorithms which will be examined below.

***Eclat Algorithm*** Eclat algorithm partitions candidate sets into disjoint groups and approaches lattice partitions in a breadth-first manner. The candidate partitioning approach that is used is similar to the primitive versions of parallel Apriori algorithm implementations. The Eclat (Kumar and K.V.Rukmani, 2010) algorithm uses enumeration trees and uses various different strategies. The novelty that Eclat introduces is to propose many efcient variants of recursive intersection of tid lists.

***Viper Algorithm*** This algorithm (Agrawal et al., 1993) uses vertical representation. The main concept behind the VIPER algorithm is representing the transactions as bit vectors, also called "snakes". These compact structures are used for counting support values of frequent patterns efficiently. The snakes offer a compressed representation of tidlists. For the counting phase, VIPER algorithm operates similar to Eclat algorithm. The main difference between the two approaches is the form of the data structure being used.

### 2.4.3. Tree Based Approaches

Tree-based algorithms provide new approaches for enumerating sets. The candidates are traversed by using subgraphs that represent the itemset lattices. The most important rule that is made use of by tree-based algorithms is the introduction of a certain order of exploration. This is necessary in order to prevent redundancy and to set enumeration without repetition. The structures used within these algorithms are also called enumeration or lexicographic trees. These trees are generated and explorated with the lexicographic order in mind. This representational approach transforms the problem of generating frequent itemset to the problem of constructing lexicographic trees. The tree structures can be generated and traversed in a breadth or depth first manner. Some algorithms introduce pruning methods in order to increase efficiency by reducing the size of the tree. Pruning mechanisms can be very beneficial for tree based algorithmic approaches since the performance of tree based algorithms are prone to suffering from size of the structure.

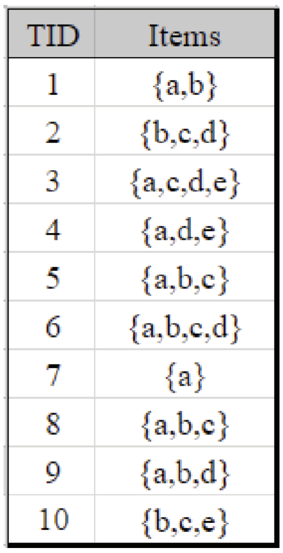
***AIS Algorithm*** This algorithm generates the tree with a breadth-first approach. The support values are counted at each level by using the transaction database. No other additional enhancements have been proposed in order to increase the efficiency of the counting, causing this approach to stay computationally limited.

There are variants of tree based algorithms that introduce recursive approaches . Being able to make use of recursive execution enables reusing the previous calculations of support values to eliminate redundant counting phases. One weakness of these algorithms is that they consume and depend on memory.

***TreeProjection*** Tree projection is a tree based approach to database projection. Tree projection approaches are based on how they construct and explore the tree structure. This could either be implemented in a breadth first or a depth first approach, or a combination of the two. Breadth first and depth first approaches have different advantages. For breadth first approaches, introducing pruning by level is possible. Depth first approaches are better for when transactions are longer. Discovering maximal patterns is more efficient when traversing the patterns depth first, since the portion (branch) of the tree under consideration is smaller. The main difference between discovering depth or breadth first lies not in the size of the actual candidate space, but how the candidates are divided and handled for managing space and memory. Tree projection algorithms can also be implemented either in a recursive or a non-recursive way. Depth-first variations generally adapt recursive methods for increased efficiency when growing a particular branch of the tree. For these adaptations, the recursion tree has the same size as the enumeration tree. The adaptations that do not introduce recursion store the projections of transactions directly as nodes of the enumeration tree.

Most tree based frequent pattern mining algorithms make us of an ordering rule. This rule could be in the form of lexicographic order, applied to sets of items as either suffix-based or prefix-based. Algorithms that are based on suffix method provide a different convention when it comes to how the sufxes of frequent patterns are extended. With these approaches, itemsets are extended with a backwards fashion, from their suffixes. The database of transactions that conform to the suffix based rule is called conditional transaction database. By using conditional transaction datasets, dataset at hand can be divided based on the current suffix under consideration. This approach traverses the database once to determine frequencies of all elements. Following a single preprocessing pass, all infrequent items are eliminated The conditional databases need only to contain frequent items. Since all items at hand are frequent, patterns can be directly generated for each. Following this step, the projected transaction database is recursively handled for each of the items at hand. This process results in the transactions being recursively projected to build up the suffix item by item. The recursive and suffix based approach enables managing the datasets efficiently. Fp- growth is one of the most well known algorithms that uses suffix based pattern exploration. Since this approach eliminates infrequent items in preprocessing phase, it has higher efficiency than the previous TreeProjection methods which count itemsets in each recursive call. Additionally, Fp- growth introduces a structure called Fp-tree to compress the dataset at hand, further enhancing manageability of the algorithm. The Fp-tree structure compresses data by making use of the ordering rules and prefix/suffix structures.

Table 2.2 Transaction database



| TID | Items |
|---|---|
| 1 | {a,b} |
| 2 | {b,c,d} |
| 3 | {a,c,d,e} |
| 4 | {a,d,e} |
| 5 | {a,b,c} |
| 6 | {a,b,c,d} |
| 7 | {a} |
| 8 | {a,b,c} |
| 9 | {a,b,d} |
| 10 | {b,c,e} |

***Fp-growth Algorithm*** has been proposed to avoid the two bottlenecks of Apriori-like algorithms, generating and managing large numbers of candidates (Han et al., 2000). It provides a novel structure, called an Fp-Tree, which is an extended prefix-tree that stores only the crucial, quantitative information of frequent patterns. The algorithm has a pre-processing phase, in which all items are counted and infrequent ones are eliminated. Next, the nodes are arranged based on their frequencies before generating the the tree structure, as seen in Table 2.2. For each itemset within the table, items are ordered by their frequencies. For example, a is more frequent than b; b is more frequent than c; and so on. The generation of the tree structure follows this phase. The initial tree only consists a null 'root'. This root is grown by inspecting the transactions one by one and inserting them in their respective branches within the tree structure. The final tree structure for the transaction database in Table 2.2 can be seen as in Figure 2.5. The ordering mechanism based on frequencies results in an increased chance of locating the most frequent nodes closer to the root of the tree; causing a higher compression in data. Within the following phase of the algorithm, a novel fp-tree based pattern growth approach is proposed, which starts from a frequent length-1 pattern, continues by examining only the respective conditional pattern base (a sub-database consisting of a set of frequent items that co-occur with the suffix pattern at hand).

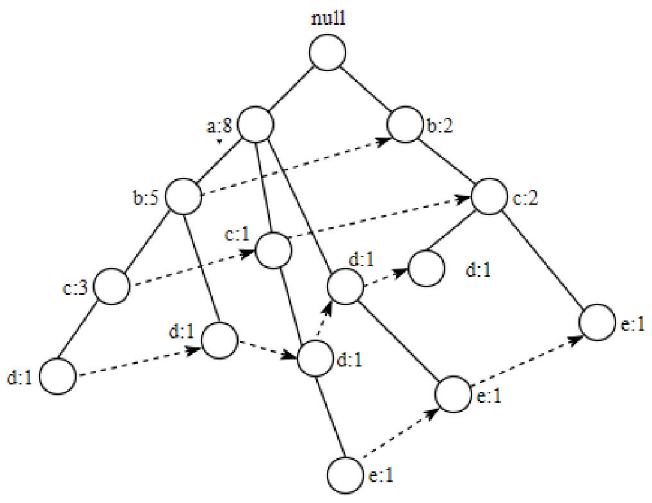


Figure 2.5 Fp Tree structure

The algorithm then constructs the respective (conditional) Fp-tree, and performs mining recursively with the use of the constructed sub-trees. The pattern growth is achieved by concatenating the suffix pattern with the new generated nodes using the conditional sub-tree. The pattern concatenation logic is executed as seen in Figure 2.6. The prefix paths after the task of dividing the fp tree into suffix pattern structure trees can be seen in Figure

2.7, applied for prefix paths ending with "e" and "de" for the complete tree shown in 2.5. the process is repeated until all paths are examined.

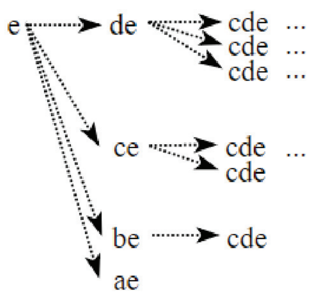


Figure 2.6 Pattern growth

This approach is highly suitable for the divide and conquer strategy and is usually implemented either recursive or parallel for optimum results. Since all frequent itemsets are encoded into the tree, this approach ensures the completeness of the result and provides a compact structure resulting from the common prefixes (shared nodes on the tree). The redundant process of generating infrequent candidates in removed from the process, causing efficiency and reduced space consumption.

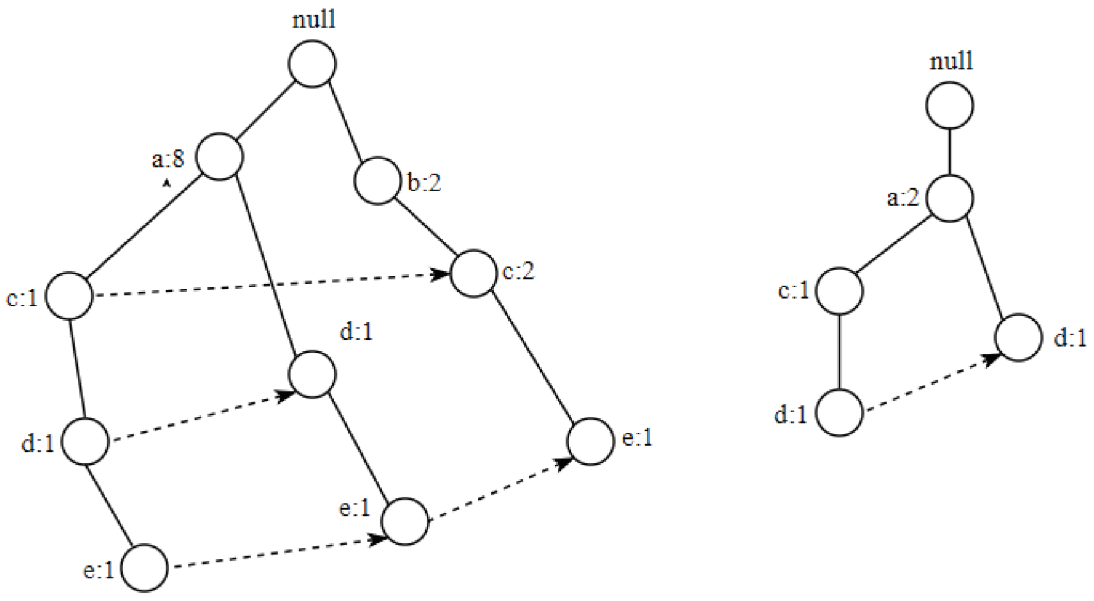


Figure 2.7 Prefix paths ending with "e" and "de"

Additionally, the employed search technique transforms into a partitioning-based divide-and-conquer method rather than the bottom-up generation of frequent itemset combinations used within Apriori-based approaches. This improvement results in a major reduction of the size of conditional pattern base, as well as the size of corresponding conditional

Fp-trees. Moreover, it transforms the problem of finding long frequent patterns into the problem of searching for shorter patterns and concatenating the suffixes. Since the least frequent items are stored as suffix within the Fp-tree, the structure offers a good overall manageability of the patterns. All these improvements result in a substantial reduction in the search costs.

### 2.4.4. Maximal and Closed Frequent Itemset Mining Algorithms

One of the greatest source consumption of frequent itemset mining is the redundant phase of counting itemsets that are actually subsets of eachother. In order to eliminate this redundancy is to discover and handle a smaller set of itemsets that are mathematically able to represent the rest of the frequent itemsets. The representations should be in such a way that they represent all frequent itemsets in a compressed way, and help eliminating the phases of discovering and storage of duplicate sets. The compact representation saves both computational time and memory consumption. There are two different approaches when it comes to how the representations reflect the itemsets at hand completely or partially. Examples of algorithms that mine closed itemsets can be listed as Closet and Charm, whereas algorithms that mine maximal itemsets can be listed as MaxMiner and Mafia.

***Mafia Algorithm*** was proposed in 2001 (Burdick et al., 2001). It was based on vertical layout structure and it proposed a depth first traversal approach. The algorithm focused on finding maximal frequent itemsets with the assumption that the entire data at hand is able to fix into the memory. The algorithm represented data in a vertical bitmap structure, allocating a set of bits to each itemset so that each bit would represent it's existence within a transaction in the db. The algorithm used intersections instead of joins to detect itemsets, and used bitwise operations upon the vertical representation to calculate the supports. This approach was able to work efficiently for larger databases and longer itemsets, but did not perform as good for short itemsets. Mafia algorithm used Apriori principle to prune the structure at hand to increase efficiency and execution time.

***Charm Algorithm*** used a vertical representation of the db at hand and mined closed frequent itemsets over this structure. Closet offered closed itemset mining over an Fp-tree structure, incorporating the pruning mechanism for efficiency. In conclusion, mining frequent closed itemsets can increase efficiency while eliminating redundant elements from mining frequent sets of items.

### 2.4.5. Parallel Fp-tree Based Algorithms

Mining frequent items, itemsets, subsequences, or other substructures is usually among the first and most popular steps for analyzing large-scale datasets, which has been an active research topic in data mining for years. Frequent pattern mining approaches can be applied on various types of data such as retail market basket data, transaction data for retail, bibliographic data, medical and health related data, biological data, geographic data, traffic accident data, sports related data, data on housing and population, social network data, synthetic data, website structure data, web traversal(clickstream) data, website visitor data and so on. Since we will be discussing the studies that focus mining web usage data in Chapter 3, studies that have applied frequent pattern mining principles in a parallel manner on other types of datasets have been observed within this section, as they are highly related to our study.

***PFP Algorithm*** The first proposal for parallelizing Fp-growth algorithm using a map reduce (Dean and Ghemawat, 2004) based approach is PFP algorithm (Wang et al., 2008). In this study, the FP-Growth algorithm is parallelized to be executed on a cluster of machines. PFP algorithm divides the workload between nodes as independent tasks so that each node in the cluster is able to execute without messaging or waiting the other nodes. This logic minimizes the costs that may arise from communications and dependencies between the nodes. The algorithm uses three MapReduce phases. Five main steps of PFP, as seen in Figure 2.8 are as follows:

- Sharding: The transaction DB is divided into P successive parts and parts (also referred to as shards) are stored on different nodes.
- Parallel Counting: Support values of all items within the DB are counted by performing the first MapReduce pass. A shard of the data is given to each mapper instance. During this step, since all items are traversed once, the set of items "I" is also discovered. The results are recorded in a list structure, called the "F-list" (frequent list).
- Grouping Items: All items "|I|" that exist on the "F-List" are divided into Q different groups. The list that contains these groups is called a "G-List". Each element in this list is assigned a unique id, "gid". Since the list structures are small because of the division, this phase is relatively fast, and the time complexity of grouping phase is O(|I|).

- Parallel Fp-Growth: The critical phase of the algorithm. Within this phase, another MapReduce pass is executed: *Mapper* – A shard of the dataset that was generated in the previous step is given to mapper instances. The mapper reads the G-List and processes each transaction in the shard individually. During this phase, items are marked according to which group they belong to. Following this "marking" step, transactions are handled for each group, and items that do not belong to the group under examination are removed.

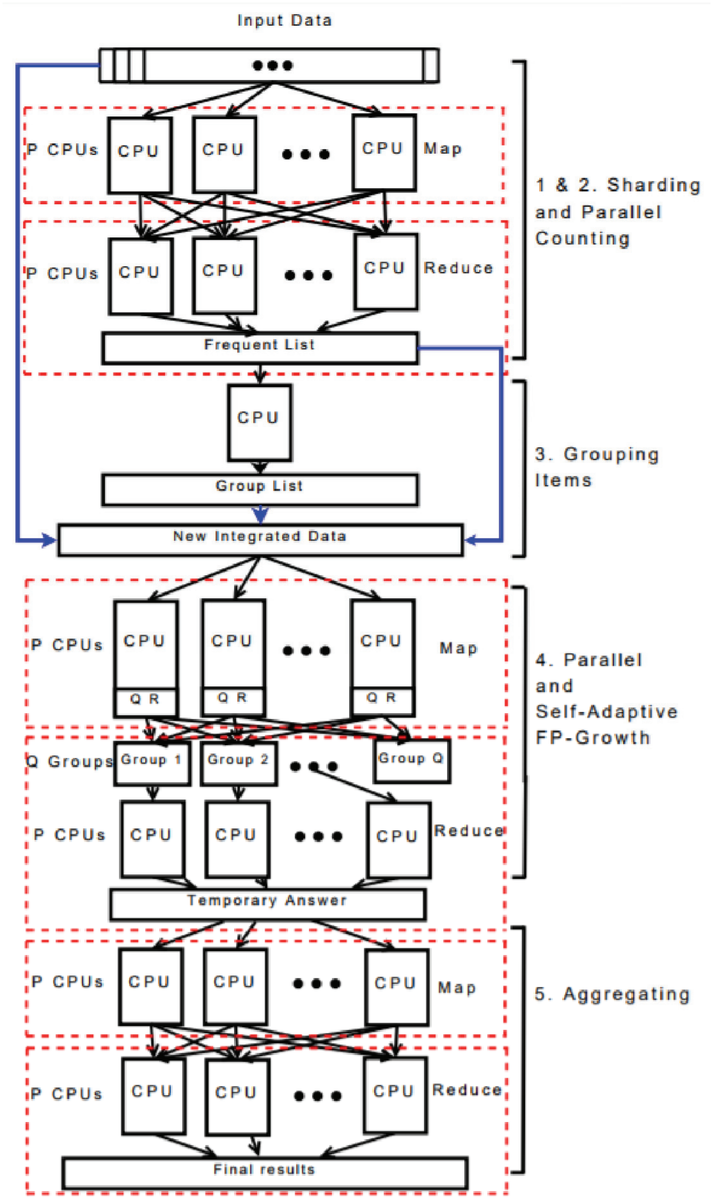


Figure 2.8 The overall PFP framework
(Source: Wang et al., 2008)

This step is repeated for each group, causing transactions to dissect into group dependent transactions. After handling every transaction, the mapper phase finishes by returning (key,value) pairs. In these pairs, the key is the group-id and the value is the group-dependent transaction that was generated.

*Reducer* – Fp-growth is performed on the shards of the dataset during this step: After all mapper instances are completed, all group dependent transactions are grouped by their gids automatically by the infrastructure. This is done for each group-id. After this operation, each shard becomes group dependent and suitable for processing simultaneously. Each reducer instance is assigned with one or more of these shards. A local tree and it's conditional sub trees are grown recursively by each Reducer. Following the generation of these structures, the patterns that are discovered are returned as output.

- Aggregating: Aggregation of the results that are generated in previous step as the final result.

The authors performed an empirical study on a dataset of 802,939 Web pages and 1,021,107 tags, and conclude that PFP algorithm is both scalable, and is able achieve virtually linear speedup.

***BPFP Algorithm*** BPFP algorithm is a MapReduce based approach for mining web usage data using parallelized Fp-growth (Zhou et al., 2010a). Within this study, experiments were performed on webdocs that are 1448580Kb in size containing 1692082 transactions. The documents were reorganized to form a set of web html documents. Two rounds of MapReduce passes are used to implement the parallel Fp-growth based algorithm. Additionally, BPFP introduces an approach to balance the parallelization. This approach is called "balanced grouping". During this step, all frequent items are divided into Q groups, with the balance between different groups in mind. The goal of balancing the load between groups is to balance the workload between parallel instances. This step can be examined under two substeps: First is the computation of "load units", which represent the cost of executing Fp-growth algorithm for each of the frequent items. Second is the division of the previously calculated "load units" into groups based on their load values. The algorithm has two major differences from PFP Algorithm: One is the novel consideration of balancing workloads, another is the elimination of aggregation step in order to list all of the frequent itemsets. A downside of this appoach is that the mean square errors and lift parameters were not considered for analysis in detail.

***IPFP Algorithm*** IPFP Algorithm, also called "improved parallel Fp-growth" algorithm, introduces a small files processing strategy based approach (Xia et al., 2013). This method is ideal for environments containing a large number of small file sets. The

IPFP algorithm makes use of a MapReduce based implementation in order to parallelize Fp-growth. The goal of the algorithm is to reduce the expensiveness of writing and reading from disk and the high execution time for processing of Hadoop. The authors discuss that the novelty reduces the memory consumption, improves the efficiency of MapReduce by managing memory and enables accessing data more efficiently, compensating for the I/O overhead of MapReduce.

Because MapReduce based parallelization causes high I/O overhead for iterative computations, later implementations such as Balanced Parallel FP-Growth with MapReduce (Zhou et al., 2010b) and the improved version of the same algorithm (YANG et al., 2016) offer balancing over the Fp-Tree in order to increase efficiency. A later implementation, S-FPG algorithm (Gassama et al., 2017) suggests using Apache Spark$^{TM}$ for the task: an in-memory, iterative computing model and framework. Spark framework makes use of a concept called RDD (resilient distributed database) (Moens et al., 2013).

***SPFP Algorithm*** Within this study, the authors propose an algorithm that is based on the IPFP algorithm. Instead of using MapReduce, which has high I/O cost, latency and difficulty of implementing iterative algorithms efficiently; the authors use the Apache Spark framework. The authors discuss that the SPFP algorithm is efficient and consumes less time because of the optimizations of the header table and the efficiency of the data management strategy.

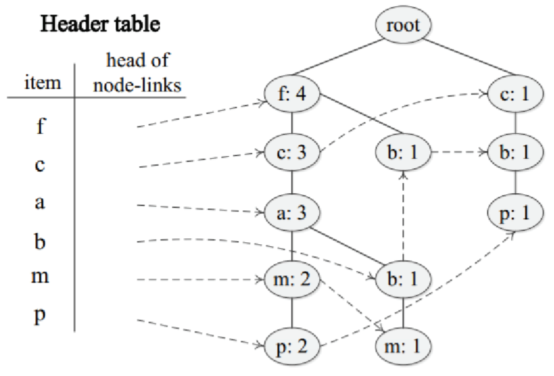


Figure 2.9 An Fp-Tree
(Source: Shi et al., 2017)

***DFPS Algorithm*** Within this study, the authors point that reading and writing on the disk at every MapReduce operation consumes a high volume of sources, causing the efficiency of the parallel Fp-growth algorithm to drop. To tackle this, the authors propose an algorithm called DFPS, "distributed Fp-growth algorithm", and use the Apache Spark framework which utilizes memory based management and increases efficiency. DFPS algorithm divides the computational tasks between nodes that are then able to execute in parallel independently (Shi et al., 2017). This division logic can be seen in Figure 2.10 for for the tree structure in 2.9. The fact that there is no need to pass messages between parallel threads or nodes makes DFPS an efficient algorithm in comparison to the PFP algorithm.

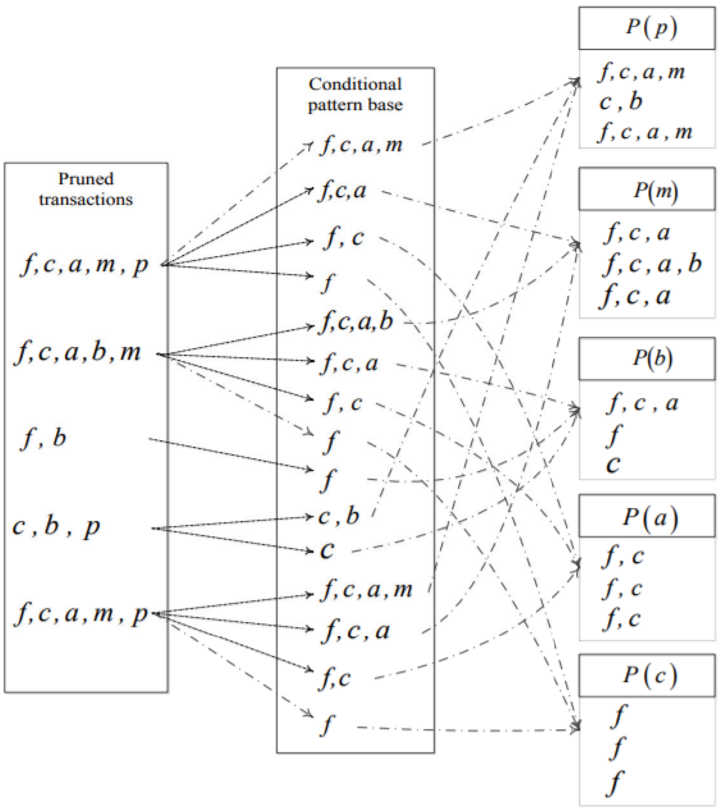


Figure 2.10 Repartition function of DFPS
(Source: Shi et al., 2017)

***DGFP Algorithm*** Focusing on the problem of imbalance between loads when the existing Fp-growth based approaches group the data set, this study proposes an optimization algorithm DGFP-growth for dividing the data set in a dynamic manner (Zhang et al., 2017).

## 2.5. Web Data and Web Mining

Web mining can be briefly described as the application of data mining techniques to the domain of Web (Agrawal et al., 1996). With the popularity of the world wide web resulting in an explosive growth in the size of the usage data, organizations are overloaded with information. Therefore it is getting more important to filter meaningful information from the huge, raw data (Pal et al., 2002). Information on what the users/consumers need, how they behave and how they interact with a website is valuable information. The web service providers are interested in predicting user behaviour to use when personalizing websites, handling the traffic load efficiently, and optimizing the design of the website to better suit the needs of different groups of users.

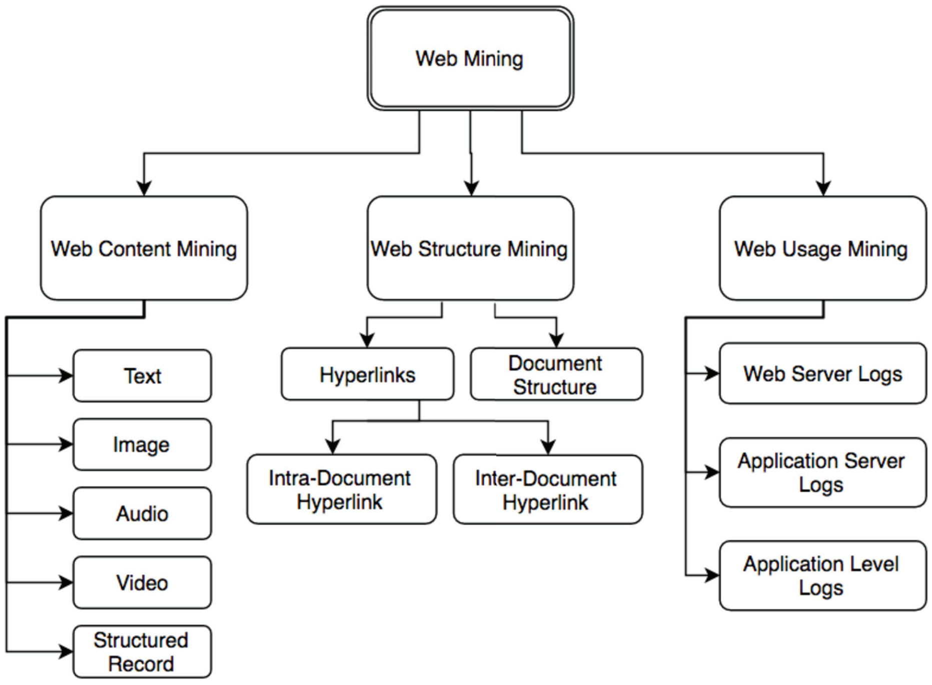


Figure 2.11 Web mining taxonomy

Web mining is a combination of multiple research areas: Data mining, world wide web, database systems, information retrieval and artificial intelligence. Although the problem could be approached by using web mining techniques as direct solutions, approaching it as part of a bigger application that addresses the problem from a generalized scope by using a combination of related fields is also common. Other application fields for web mining are data confirmation, validity verification, pipelining for web portals, fraud detection, data integrity, taxonomy capture, content management, content generation and opinion mining (Galitsky et al., 2011). Therefore, web mining is a research field that offers a broad variety of application fields, is very popular and in-demand, and contains many open problems due to the exponential growth of web usage (Kumar and S.Nandan, 2015). When web mining process is broken down into sub-steps in data mining terms, it can be said to have three main operations : Clustering, association and sequential analysis. An example for clustering steps can be given as finding a natural grouping upon a list of users. The research of which URLs are most often accessed together can be given as an example to association operation. Finally, the order in which the URLs are mostly accessed can be given as an example for sequential analysis. Note that these sub-steps, as in most research fields regarding real world problems, do not possess clearly set boundaries between each other, they overlap considerably often. One of the greatest difficulties of working on web mining is the complexity of the data that is caused by its unstructured nature. Another difficulty arises from the fact that the collected data is not completely reliable or accurate in most cases, again caused by the complex structure of the web itself. The storage and collection of world wide web data is examined under section 2.7.2.

## 2.6. Taxonomy of Web Mining

A taxonomical categorization has been proposed as in Figure 2.11 (T.Srivastava et al., 2005). This approach describes main types of data to be inspected within web mining as content, structure and usage data. Some categorizations consider user profile as a separate type (Hasan et al., 2012).

**Web Content Mining** is the application field of web mining that studies on discovering useful information from the web contents, such as text, images videos, or from data and documents that are published on the internet (Inamdar and Shinde, 2010) (Kosla and Blockeel, 2000).The traditional technique of searching the web is the search conducted on contents. Web Content mining can be defined in other terms as the extended work that is performed by search engines (Dunham, 2003). The sources might be in the form of HTML, JSON (semi-structured),in plain text (unstructured) or XML (fully-structured).

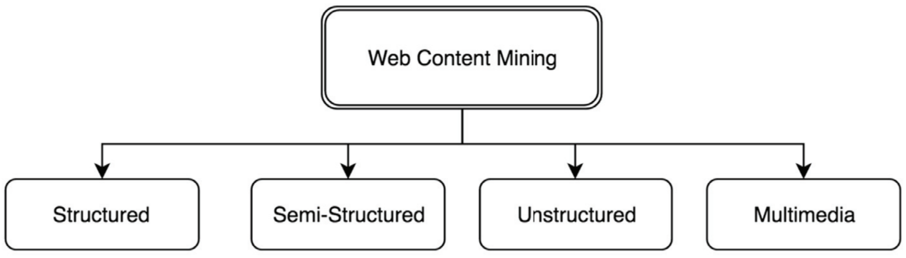


Figure 2.12 Web content mining subtypes

Web content mining mostly aims to discover patterns in large collections of documents. It is also an often used approach within ontology based learning methodologies, when merging or mapping ontologies, or within the research field of instance learning (Cooper et al., 2003).

Two approaches used in web content mining are agent based approach and database approach (Dunham, 2003) (Inamdar and Shinde, 2010). The approach to be followed is often determined by looking at how structured the data at hand is. Therefore, web content mining approaches are often categorized as seen in figure 2.12.

**Web Structure Mining** is the research field that focuses on inspection over the world wide web's hyperlink structure. It's main focus is to identify relations between web pages, either by direct links or by links of information and to discover previously unknown relationships between or within web pages by working on previously mentioned data. This research field often studies on the topological information of hyperlinks, either with including or without including their descriptions. This connection between pages allows, for instance, a search engine to pull data in accordance with a search query and to directly display links for the web sites that contain related content. The inspection of the structure is conducted by spiders or agents traversing and scanning web sites, retrieving the home page and linking the information and the reference links and thus creating a linked, specific page that contains the desired information. This collected information enables both traversal from a link to a certain web page regarding a search query and to cluster information into site maps, both increasing availability of a web page or information to users. For example, the search engine Google owes its high usability and success to the PageRank algorithm, which rates relevance and reliability of web pages based on hyperlinks directed to it from other pages, and vice versa (Page et al., 1999). This structured data collection also provides information on a certain page's ranking (Page et al., 1999) or authoritativeness (Kleinberg, 1998). Web structure mining and web content mining are often applied jointly to utilize the content and the structure of a hypertext (Stummea et al., 2006). Some studies do not discriminate between the two fields (Cooley et al., 1997).

**Web Usage Mining** is the process of automatically discovering usage patterns from Web Usage data by applying data mining techniques. The collected information is often meant to be used for better serving and understanding the needs of web-based applications (Srivastava et al., 2000a). It can be used within numerous fields of study such as web caching, web perfecting, intelligent online advertisements and group targeting and web personalization. Most approaches apply one or more of the following: clustering (either on pages or user sessions), association rule generation, sequential pattern mining and Markov models, as shown in figure 2.7 (Srivastava et al., 2000b).

## 2.7. Web Usage Mining

Web usage mining is the concept of applying data mining techniques to different types of raw data of user-web resource interactions logs, which are also commonly referred to as web usage logs (Srivastava et al., 2000a). The techniques are intended to discover insightful, new association rules. Association rules were first formally defined as a class of regularities in 1993 (Agrawal et al., 1993). These association rules are expected to be able to represent various groups of users that frequently interact with the website for different needs and expectations. The objective is to collect, model and analyze the patterns that lie within the data of interactions between users and the website. The log data can be obtained from different types of sources such as servers, proxy servers or the client machine itself. For the domain of web usage mining, the raw data generally contains web log files that have records of page requests. After discovering the frequent patterns that lie within the dataset by using frequent pattern mining algorithms, association rules are generated to further describe the dataset. Some common terms used when mining web usage data are:

- A *user* is an individual that requests a certain file or resource from one or more web servers
- An *item* in the context of web usage mining is a web resource of a particular website that can be requested by the website visitors.
- A *page view* is used for naming a one time display of the collection of resources on a user's browser.
- A *click-stream* is a sequential set of requests.
- A *user session* is a set of requests across the entire web for a single user and a given time span.
- The request set for a particular website and session is referred to as a *server session* (also commonly referred to as a 'visit').
- *Support count* refers to the frequency of occurrence of an itemset.
- *Support* refers to a fraction of transactions that contain a certain itemset.
- *Frequent itemsets* are sets of web resources that are frequently requested together by the visitors.(frequently refers to a number above a given support threshold.)

Phases that are applied within the process of web usage mining, as seen in Figure 2.13, can be listed as follows: Data Collection, Preprocessing, Pattern Discovery, Pattern Analysis.

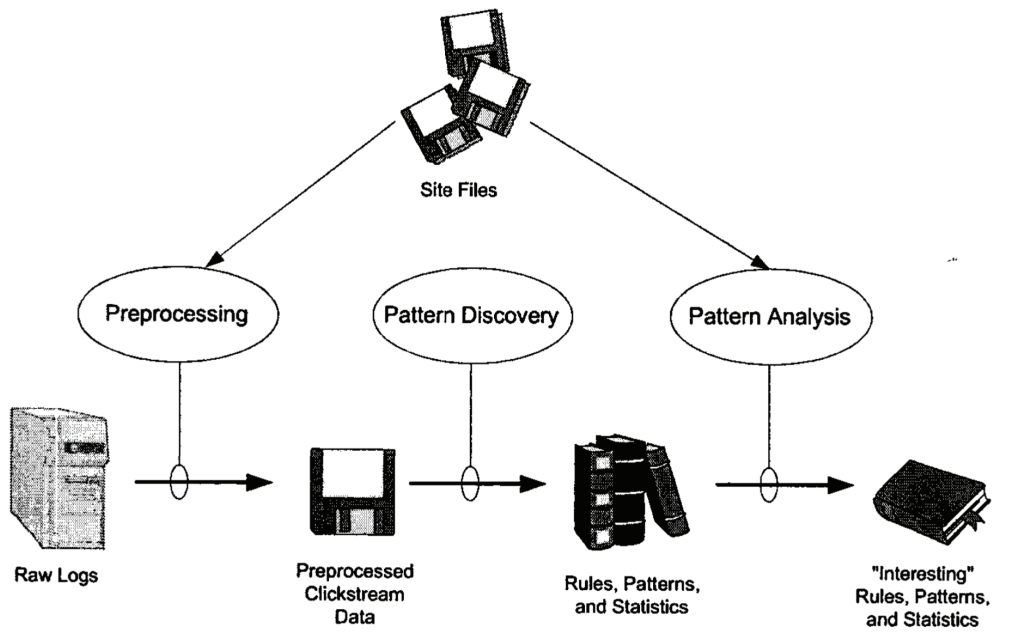

Site Files
Preprocessing
Pattern Discovery
Pattern Analysis
Raw Logs
Preprocessed Clickstream Data
Rules, Patterns, and Statistics
"Interesting" Rules, Patterns, and Statistics

Figure 2.13 Web usage mining steps
(Source: Srivastava et al., 2000)

**Data Collection** Collection is the first step to mine web usage data. The authenticity, integrity and quality of data collected affects the following steps tremendously, which makes this a critical base step in web usage mining studies. Data collection method must be scientific and organized and the type/content of data collected must be sufficient and suitable for the requirements of the scope and interest of the study. Various sources of data collection were mentioned in the previous section in detail.

**Preprocessing** Since real world data that is directly extracted often tends to be incomplete or tends to include aggregate information, noisy data or inconsistencies; a pre-treatment for increasing quality of input data by either unifying, integrating, or transforming information at hand to a more consistent state is usually necessary for healthier further discovery processes (Salleb and Vrain, 2000) (Agarwal and Psaila, 1995). Data cleaning includes routines for filling in missing values, smoothing noisy data, identification of outlier values, or reducing inconsistencies within data at hand. Integration of multiple data sources process is required to form a coherent data store by using correlation analysis, data conflict detection, and the resolution of semantic heterogeneity contribute towards smooth data integration. Data transformation processes such as normalization or aggregation, is often used to convert data into appropriate forms for following data mining processes. Data reduction and data summarization techniques such as data cube aggregation, reducing dimensions within data, data compression, numerosity reduction, and hiding sensitive information within data (discretization) can be used to reduce the size of the representation of the data, with the aim of minimizing the loss of information in mind. For summarization, techniques such as histogram analysis, binning and clustering can be used for obtaining a summarized version of the data at hand. Other examples for steps of data preprocessing phase of web usage mining can be given as :

1. user identification
2. session identification
3. user path discovery

**Pattern Discovery** Pattern discovery is the generalised name for describing the overall process of actual data mining phase that is applied on preprocessed data. These could be briefly listed as choosing an appropriate data mining task, designing, implementing and applying an algorithm in accordance with the task at hand. Different mining approaches could be applied depending on the data at hand, such as path analysis, association rule mining, sequential pattern mining, clustering and classification are used. The preference depends on the requirements of the data at hand to determine which mining technique or techniques to make use of. After the execution of the mining algorithms, data in web access logs can be transformed into knowledge to uncover the potential patterns underneath the pre-processed log data and involves analysis of these patterns. The aim of this process is to find frequent patterns, associations, correlations or structures among sets of items. An item could refer to a user or a page, depending on the goal and the method.

The knowledge discovery methods have been defined in a general scope in the previous sections. A brief description of pattern discovery techniques that are applied to the scope of web usage will be discussed here.

- *Classification* is applied to discover various profiles of users. These profiles could later be used for distinguishing users with different behaviors, helping in predicting the most interesting pages for a single user. Classification techniques could also be applied to classify sessions. This approach results in classes of sessions, rather than user types. The information gained could again be used for prediction and recommendation purposes.

- *Clustering* approach could be applied with various approaches. Pages can be clustered, with an aim to detect pages containing similar content. Another path to follow could be clustering usage data, focusing on defining groups of users that have similar browsing behaviour.
- *Regression* methods can be applied to web usage data to focus on making predictions of unknown values. This is done by mapping a data item into a real-valued prediction variable.
- *Association rule mining* is one of the most commonly applied techniques when mining web usage data. It could be used to discover unknown relations between web pages, which could be used to predict or recommend the most interesting next page (or group of pages). Association rules in web usage mining will be examined in detail in section 2.7.1.
- *Sequential pattern mining* approaches can be applied to identify relationships between occurrences of sequential events. This is beneficial when predicting subsequent visits or page views.

### 2.7.1. Association Rules in Web Usage Mining

Generation of association rules can be examined as two separate steps. During the first phase, all the frequent itemsets are discovered. During the second phase, constraints such as minimum confidence are applied on the discovered frequent itemsets to obtain rules. The relationship between two items could be formally represented as a rule:

$$X => Y$$

Assuming X and Y are pages in a website, this association rule suggests that a strong relationship exists between pages X and Y, meaning many visitors of page X also visit page Y. How strong a rule applies to a dataset can be determined by calculating it's support and confidence values. *Support ($\sigma$)* value represents how often a rule applies within a given data set. *Confidence* determines how frequently items in Y appear in transactions that contain X.

$$\text{Support, } s(X \longrightarrow Y) \;=\; \frac{\sigma(X \cup Y)}{N}$$

$$\text{Confidence, } c(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

While the second step of association rule mining is fairly straightforward, the first step is complex and therefore requires detailed attention. Various approaches for finding frequent itemsets in a database are examined in section 2.6.

### 2.7.2. Sources of Data for Web Usage Mining

Web Usage Mining Data can be collected at the server side, client side, from the proxy server or the organization's own logs stored within a database structure. The type of information collected and stored as web data, the level of the collected data and the method of implementation to collect the data is all dependant on the choice of the source type. The decision of which source is going to be used is effected by the mining task at hand and the scope of the application.

- *Server Level Data:* Since web server logs explicitly record the browsing behaviour of a site's visitor traffic, it is considered an important source for web usage mining data.This type of web site traffic data, which is recorded by server logs is not completely reliable hence the caching mechanisms within different levels of the world wide web environment. Cached accesses and packages that are passed using the POST method are not logged in server level logs. There are various formats that the data can be stored in such as Common Log and Extended log formats. Server level web usage data can be collected using numerous methods. One of these methods, packet sniffing, is applied by extracting data directly from incoming TCP/IP packets. Another approach is to use cookies and tokens that are generated by web server for individual client browsers for the purpose of tracking site visitors. Since cookies depend on the cooperation of users and raise concerns regarding privacy issues and the HTTP protocol has a stateless connection model; it is difficult to track individual users reliably using this method. An example for a web server log is given in Listing 2.1.

```
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400]
"GET /history/apollo/HTTP/1.0" 200 6245
```

Listing 2.1 A web server log

For given example, 199.72.81.55 is the host of the request. A hostname is used when possible, otherwise this section contains an internet address. The section containing [01/Jul/1995 :00:00:01 -0400] is called the timestamp, in this example it is formatted as " DAY /MON/YYYY:HH:MM:SS". For above example the timezone setting is -0400. The phrase "GET /history/apollo/HTTP/1.0" denotes the actual request, given in quotes. 200 is the HTTP reply code for this request, and 6245 is the number of bytes contained within the reply message.

*Client Level Data:* Client side data can be collected by using remote agents, or by using a browser that has enhanced data collection capabilities. Both methods require user cooperation, either by enabling Javascripts or Java applets to work, or by using the modified browsers. With client level data collection, it is possible to identify sessions and it is possible to go-around the side effects of caching that result in difficulties when collecting web site access information. However, this approach is no better than server log collection when it comes to determination of how many times a page has been viewed. It may even introduce some extra overhead such as when the Java applet is launched or loaded for the very first time. On contrast, data collection using Javascripts consumes less time but this approach is not able to capture all user clicks. Especially reload or back buttons are missed in most cases. Methods mentioned above are most capable when it comes to capturing single-user, single-site browsing behaviours. For single user over multiple web-site behaviour capturing, using a modified browser is a much more versatile approach in most cases. The biggest cost of this approach is, as mentioned above, convincing users to accept the terms and to use this browser for their daily web browsing activities. There has been some approaches that look for ways to encourage the users to use the browser. For example, several companies have introduced reward systems to users for clicking on advertisements while browsing (Alladvantage)(Netzero).

- *Proxy Level Data:* Web proxy is an intermediate caching layer between client browser and web server layers. It is possibe to use proxy caching to decrease load time experienced by users of a web page or to decrease the traffic load at server and client sides. Efficiency of a proxy cache structure highly depends on how well the design predicts future page requests (Cohen et al., 1998). Proxy level data is able to capture real http requests for multiple clients for multiple web servers, which makes it a suitable resource when the scope of a study requires analysis of multiple, anonymous users that share a common proxy server.

The importance of analyzing web usage data to enhance websites increases as the popularity of the web grows. Many studies propose using data mining methodologies in order to discover frequent patterns. Fp-growth based approaches are particularly suitable for the task. This is caused by the divide and conquer based nature of the algorithm. Fp-growth also eliminates the cost of candidate management and performs only two db scans, and utilizes a compact data structure; making the algorithm suitable for the task of mining frequent patterns in web usage data. The following chapter discusses applications of frequent pattern mining to the domain of web usage data.

# CHAPTER 3

# RELATED WORK

With the increasing popularity in internet usage, a huge amount of website traffic data is stored on servers each second. In a network that consists millions of participants, traffic data generation can result up to gigabytes of logs per second. The increase in the amount of information and the volume of visitors results in an opportunity for analyzing these traffic data to extract informing patterns for providers to store and serve data more efficiently to users. A common approach to analyzing this log data is by applying data mining techniques which is called web usage mining. Discovering frequent patterns is the key step of web usage mining. This is performed by applying frequent pattern mining algorithms on the web log files. Frequent pattern mining can be used to discover various different interesting information regarding a website usage such as most visited page groups, most popular traversal paths among users, different user groups based on behaviour or any other variant. Discovering most frequent traversal paths or node groups within these paths allows website design to be enhanced accordingly, resulting in higher quality in website structure, better utilization of resources thus dropping maintaining costs, better user engagement and so on.

In this section, various algorithmic approaches to the problem of applying frequent pattern mining methods to the domain of mining web usage data will be examined. Since Fp-growth algorithm is suitable for mining longer patterns (web traversal paths), performs only two db-scans, eliminates the costly phases of generating and handling candidate sets, and uses a compact data structure, it is highly suitable for the task. The initial approaches for mining web usage logs have been implemented in a more straightforward, serial fashion. These implementations require a single machine/single thread to execute, thus, require more time to process as data grows. With the exponential growth in web usage data, faster implementations needed to be proposed in order to process higher volumes within an acceptable execution time. Therefore, the more recent implementations divide the problem into sub-tasks and parallelize the algorithm. The divide-and-conquer based nature of Fp-growth algorithm again makes it a suitable approach for parallelization. Section 3.1 describes a serial Fp-growth algorithm based approach to the task of mining web usage data. Section 3.2 lists and describes parallel Fp-growth based algorithms in the scope of web usage mining.

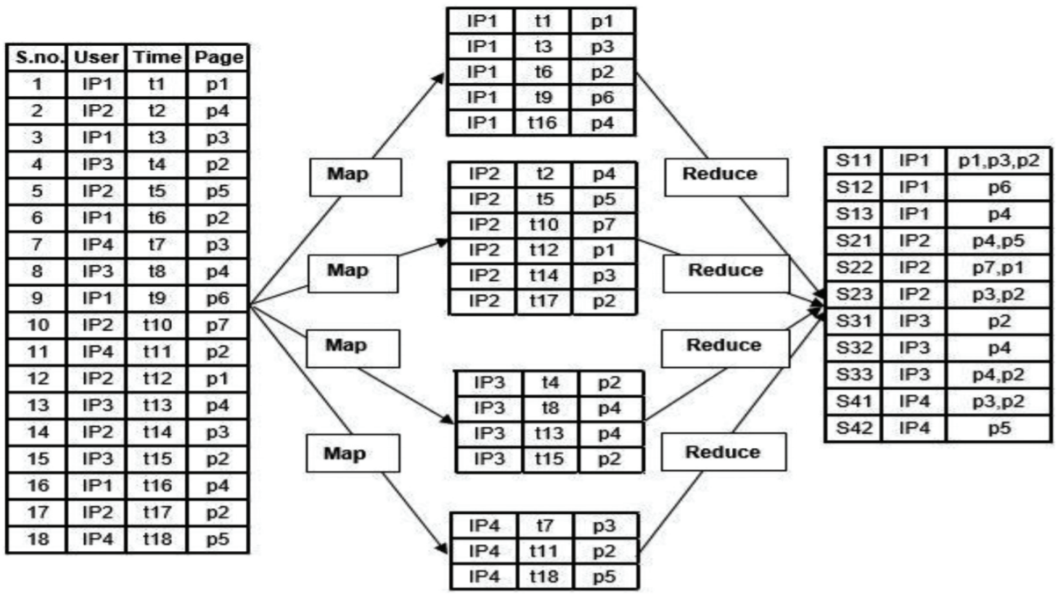
## 3.1. Serial Fp-growth based Web Usage Mining Algorithms

Although there are various implementations for mining frequent patterns using Fp-growth algorithm, the focus on mining web usage logs using this method was first proposed in 2010. The study implemented both Apriori and Fp-growth algorithm in a serial manner, executed the algorithms on world wide web usage data to discover frequent patterns (Kumar and K.V.Rukman, 2010). This study also compared Apriori algorithm and Fp-growth. Following this approach, a study proposed including the use of "utility value" to the popular approach of using support value. By using both of these concepts, the authors aimed to find not only frequent patterns, but also how important a certain page was for a user (Poovammal and Cigith, 2011). Various other studies implemented and compared Apriori and Fp-Growth algorithms in the following years (Kumar and K.V.Rukmani, 2010), (R.Kousalya et al., 2013), (Singh et al., 2014), (Shaikh, 2015). In 2015, a comparison between divisive Apriori and Fp-growth approaches in the domain of web usage mining was studied (Sharma et al., 2015). Within following year, Fp-growth algorithm was used specifically for improving user navigation patterns (Kaliyaperumal and Dorairangaswamy, 2016).

***SPFP-Tree Algorithm*** A novel method called the single pass frequent pattern tree (SPFP-tree) was proposed in 2016. This method proposed generating the Fp-tree with performing only one db scan in an incremental approach. Following the construction, the algorithm changes the tree structure dynamically for creating and rearranging a compact tree structure that is ordered by frequencies. This is achieved by ordering items by their frequencies in a descending way from root to leaves. The ordering is performed repeatedly immeadiately after including a new transaction (Shahbazi et al., 2016).

## 3.2. Parallel Fp-growth based Web Usage Mining Algorithms

The most primitive studies that parallelize FPM algorithms performed by distributing the work among threads that use a shared memory. Because of the huge volume of web usage data, the limitation of single-machine methods motivated researchers to introduce parallelization to the process of mining web usage data. Shortly, as implementations evolved, parallelization logic moved from a more thread-based, single machine logic to parallelizing the task to be executed on a cluster of computers. A thread based parallelization approach using a tree structure mine the frequent patterns was proposed in 2006 (Chen et al., 2006). In this study, the authors partition a single Fp-Tree into sections. Each section is then handled independently by parallel threads. Additionally, a heuristic method to be used for adjusting the workload among threads was introduced. The solution is executed on various datasets including kosarak, a dataset that contains 990 000 instances of click-streams that occurred in a news portal. Although the algorithm performed well on other non web-based datasets, it did not perform well in the tree generation phase. The authors discuss that this is caused by the nature of the dataset, that the logs contain too many short transactions (most of the transactions consist a small number of itemsets).



| S.no. | User | Time | Page |
|---|---|---|---|
| 1 | IP1 | t1 | p1 |
| 2 | IP2 | t2 | p4 |
| 3 | IP1 | t3 | p3 |
| 4 | IP3 | t4 | p2 |
| 5 | IP2 | t5 | p5 |
| 6 | IP1 | t6 | p2 |
| 7 | IP4 | t7 | p3 |
| 8 | IP3 | t8 | p4 |
| 9 | IP1 | t9 | p6 |
| 10 | IP2 | t10 | p7 |
| 11 | IP4 | t11 | p2 |
| 12 | IP2 | t12 | p1 |
| 13 | IP3 | t13 | p4 |
| 14 | IP2 | t14 | p3 |
| 15 | IP3 | t15 | p2 |
| 16 | IP1 | t16 | p4 |
| 17 | IP2 | t17 | p2 |
| 18 | IP4 | t18 | p5 |

| | | |
|---|---|---|
| IP1 | t1 | p1 |
| IP1 | t3 | p3 |
| IP1 | t6 | p2 |
| IP1 | t9 | p6 |
| IP1 | t16 | p4 |

| | | |
|---|---|---|
| IP2 | t2 | p4 |
| IP2 | t5 | p5 |
| IP2 | t10 | p7 |
| IP2 | t12 | p1 |
| IP2 | t14 | p3 |
| IP2 | t17 | p2 |

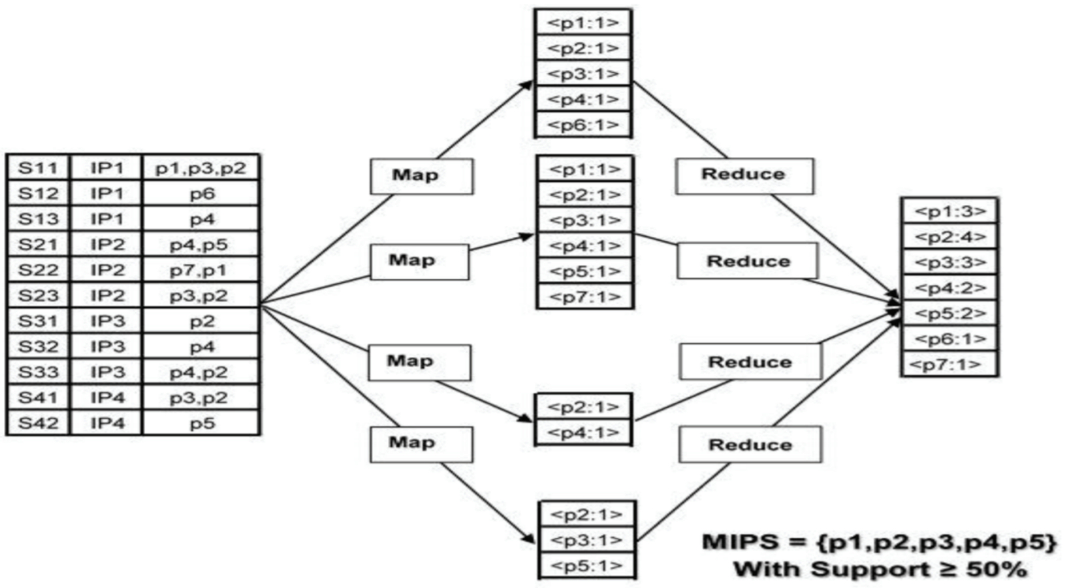
| | | |
|---|---|---|
| IP3 | t4 | p2 |
| IP3 | t8 | p4 |
| IP3 | t13 | p4 |
| IP3 | t15 | p2 |

| | | |
|---|---|---|
| IP4 | t7 | p3 |
| IP4 | t11 | p2 |
| IP4 | t18 | p5 |

| | | |
|---|---|---|
| S11 | IP1 | p1,p3,p2 |
| S12 | IP1 | p6 |
| S13 | IP1 | p4 |
| S21 | IP2 | p4,p5 |
| S22 | IP2 | p7,p1 |
| S23 | IP2 | p3,p2 |
| S31 | IP3 | p2 |
| S32 | IP3 | p4 |
| S33 | IP3 | p4,p2 |
| S41 | IP4 | p3,p2 |
| S42 | IP4 | p5 |

Figure 3.1 Sessionization phase of MIP-PFP
(Source: Sisodia et al., 2016)

***MIP-PFP Algorithm*** This study introduced the concept of "interestingness" in order to improve the PFP algorithm. The algorithm was named "most interesting pattern-based parallel FP-growth" or "MIP-PFP" algorithm (Sisodia et al., 2016). The algorithm processes web server log data and introduces a novel concept called most interesting pattern sets. The most interesting pattern sets, also called "MIPS", are determined by investigating the percentage of users that have accessed each page. This is performed by counting the page hits for every individual page from every session. The authors proposed implementing the counting operation with a MapReduce based approach. For the task of detecting frequent patterns, the implementation used the Apache Spark framework. The discovered patterns are then used for predicting web user navigation behaviours.



| | | |
|---|---|---|
| S11 | IP1 | p1,p3,p2 |
| S12 | IP1 | p6 |
| S13 | IP1 | p4 |
| S21 | IP2 | p4,p5 |
| S22 | IP2 | p7,p1 |
| S23 | IP2 | p3,p2 |
| S31 | IP3 | p2 |
| S32 | IP3 | p4 |
| S33 | IP3 | p4,p2 |
| S41 | IP4 | p3,p2 |
| S42 | IP4 | p5 |

Figure 3.2 Deducting MIPS elements from transactions
(Source: Sisodia et al., 2016)

MIP-PFP is divided into three phases: preprocessing phase, the mining phase of frequent patterns on sequential data, and the cleansing phase that removes unnecessary sequences. Within the pre-processing phase, raw weblogs are cleansed from faulty transactions and irrelevant pages or attributes. This phase also contains sessionizing the requests in a 1-hour timeout based manner. The authors state four rules as principles for sessionizing requests. All of the following properties must be satisfied in order to locate two separate records in the same session: (1) both records have the same IP, (2) records have a link between them, (3) the visits all occur within a given time threshold 1 (threshold 1), (4) a predefined session duration is not surpassed 2 (threshold 2). Figure 3.1 describes the application of sessionization phase on an example transaction database by using map-reduce phases.

The key novelty that this study introduces is the concept of most interesting page sets (MIPS). These page sets are used for including an interestingness measure to the pages under consideration. Within the study, authors specified this threshold as 50%. By making use of a map-reduce based model, URLs from all sessions are counted in order to discover MIPS. After preprocessing the data at hand, MIPS are generated, as explained above. The MIPS elements are then used for forming the sequence data in such a way that transactions only contain pages that are included in MIPS. All other pages (non-MIPS) are removed from all transactions. An example for the generation phase can be seen in Figure 3.2. Within the example, pages p6 and p7 remain below the support threshold, therefore are not included as MIPS items.

After discovering the MIPS items, this information is applied on transactions at hand in order to remove all non-MIPS items from sessions. This phase in MIP-PFP algorithm reduces both storage and execution time requirement. The MIP process is shown as the initial step within Figure 3.3.

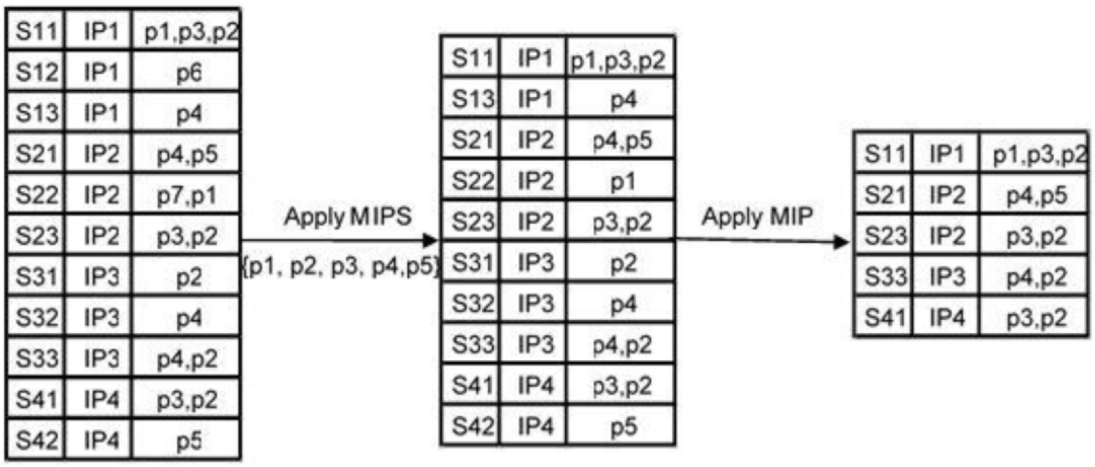


Figure 3.3 Removal of non-MIPS and 1-item elements
(Source: Sisodia et al., 2016)

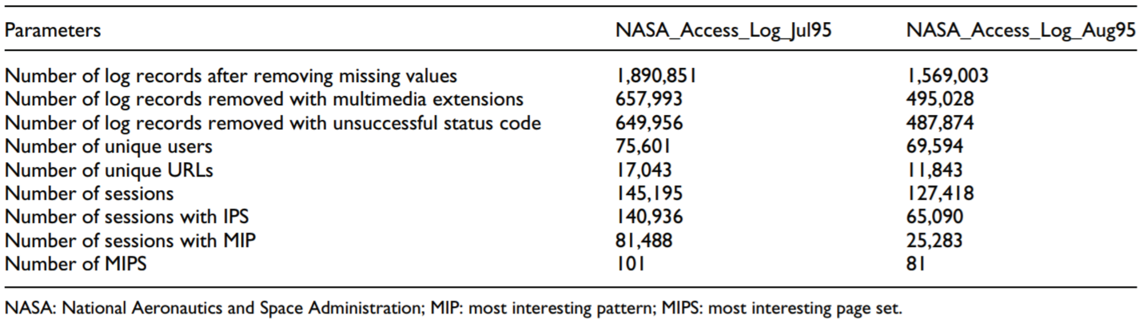
During the following phase, the sequences that have 1 element are cleansed from the set as they do not contain any patterns to serve the authors' needs for finding frequent, interesting patterns. The remaining patterns are called the most interesting pattern (MIP). This process can be observed within Figure 3.3 as the second step. Experiments were performed on "National Aeronautics and Space Administration" (NASA) weblog datasets. Two weblog files "NASA_Access_Log_Jul95" and "NASA_access_log_Aug95", containing months of clickstream data were used in this study. General statistics of the dataset can be seen in Table 3.1.

Table 3.1 Statistical details of NASA dataset logs
(Source: Sisodia et al., 2016)

| | NASA_Access_Log_Jul95 | | | | NASA_Access_Log_Aug95 | | | |
|---|---|---|---|---|---|---|---|---|
| | 1000 | 10,000 | 100,000 | 1,000,000 | 1000 | 10,000 | 100,000 | 1,000,000 |
| Number of unique URLs | 165 | 752 | 3150 | 13,025 | 145 | 700 | 2626 | 9564 |
| Number of sessions | 127 | 972 | 9150 | 83,306 | 98 | 894 | 8729 | 86,293 |
| Number of sessions with MIPS | 106 | 852 | 8137 | 74,817 | 77 | 625 | 7306 | 76,375 |
| Number of sessions with MIP | 57 | 505 | 4886 | 43,733 | 39 | 247 | 3602 | 38,346 |

Further statistical information regarding the dataset after cleaning and preprocessing phases can be seen as in Table 3.2. While evaluating the approach, authors conclude that as the minimmum support value for interestingness increases, the number MIPS decreases; therefore it is given as 0.01 to obtain a large number of MIPS items within this study. Similarly, as the support threshold for MIP increases, the number of frequent sequence patterns decreases. The authors kept this minimum support to 0.001 in order to obtain a significant number of frequent patterns.

Table 3.2 Statistical information computed of NASA dataset logs after preprocession (Source: Sisodia et al., 2016)



| Parameters | NASA_Access_Log_Jul95 | NASA_Access_Log_Aug95 |
|---|---|---|
| Number of log records after removing missing values | 1,890,851 | 1,569,003 |
| Number of log records removed with multimedia extensions | 657,993 | 495,028 |
| Number of log records removed with unsuccessful status code | 649,956 | 487,874 |
| Number of unique users | 75,601 | 69,594 |
| Number of unique URLs | 17,043 | 11,843 |
| Number of sessions | 145,195 | 127,418 |
| Number of sessions with IPS | 140,936 | 65,090 |
| Number of sessions with MIP | 81,488 | 25,283 |
| Number of MIPS | 101 | 81 |

NASA: National Aeronautics and Space Administration; MIP: most interesting pattern; MIPS: most interesting page set.

The authors also conducted experiments by modifying with the partition number for Spark$^{TM}$ (between 1 to 8 partitions). The authors observed that for the standalone application, Spark$^{TM}$ performed best when given two partitions. The results from the experiments suggested that MIP-PFP algorithm that used Apache Spark$^{TM}$ framework was able to run 10 times faster than PFP algorithm using Hadoop.

***S-FPG Algorithm*** In 2017, a spark-based parallel implementation of Fp-growth algorithm was proposed in 2017(Gassama et al., 2017). The authors performed an analysis on datasets "bms-webview1" and "bms-webview2" that contained months of usage data of an e-commerce website.

**Algorithm 1:** Counting items

```
Input  : the input file : f and min_sup : d
Output: F1, frequencies of items
for each transaction line in T do
    for each item I in transaction do
        mapItem(I => (I,1))
    end
end
T = flatmap(line => f.getTransaction())
F1 = reduceByKey(_+_).filter(item with frequency above d)
```

The authors argue that MapReduce based applications have high I/O costs, since MapReduce is a disk-based mode. They conclude that a MapReduce based model is not suited for Fp-growth based approaches for mining frequent itemsets which rely on intensive iterated computation. Instead, they propose using Spark$^{TM}$ framework for the task, which is much more suitable for iterative computation since Spark$^{TM}$ is able to execute iterative tasks on the memory much faster. The authors state that Spark$^{TM}$'s in-memory primitives improve performance to 10 times faster for frequent itemset mining (Sisodia et al., 2016). The S-FPG process is shown in detail within Algorithm 1. The output of this part of the algorithm, F1, holds the information of frequencies of items and is to be used for generating the header table. Following this step, the Fp-Tree is built as described in Algorithm 2.

**Algorithm 2:** S-FPG Algorithm

```
Input  : transaction list T, F1 from first step
Output: Fp-Tree
1  create root with label "null"
2  for $each\ transaction\ line\ in\ T$ do
3      sortItems(items in transaction, F1)
4      addItems(sorted items, tree)
5  end
6  fpGrowth(tree)
```

**Algorithm 3:** addItems(sorted list,tree)

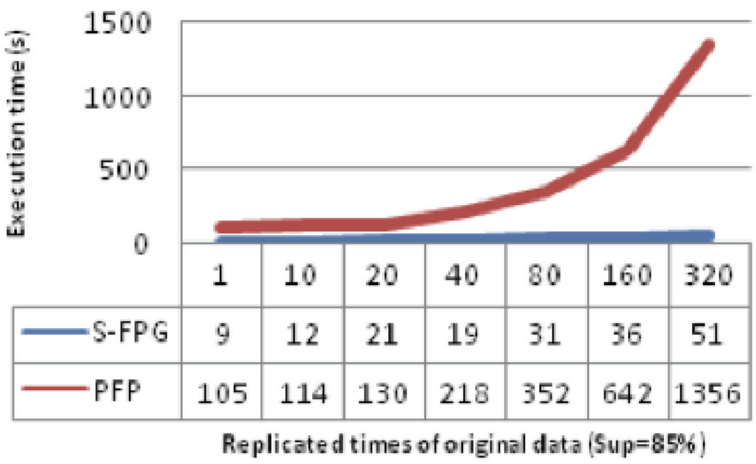
```
Input  : sorted list,tree
1  if $root\ has\ the\ head\ of\ the\ item\ as\ a\ child\ N$ then
2      increment count of the child N
3  else
4      create new node N with count = 1
5      link N to root
6      link to other nodes with same item
7  end
8  if $head\ is\ not\ null$ then
9      addItems(currentList,node N)
10 end
```

During the second step, tree structure is initialized by creating the root as null. Following initialization, frequent items in each transaction line are ordered according to the frequency list, F1 which was found in the initial step. Following the sorting step, the sorted list is added to the tree, with the order preserved. A detailed pseudocode of this step can be seen in Algorithm 3. Note that the node-link structure from header table to between instances of the same item is preserved while creating and inserting nodes.

**Algorithm 4:** fpGrowth()

```
Input  : tree
1  if $there\ exists\ only\ one\ path\ P\ in\ tree$ then
2      generate all possible subpaths of P
3  else
4      for $each\ item\ i\ in\ header\ table$ do
5          generate the conditional pattern base pbi for i generate the conditional tree
             ti for pbi
6          if $ti\ is\ not\ empty$ then
7              fpGrowth(Tree-i)
8          end
9      end
10 end
```

Finally, fpGrowth() method is called for the tree structure at hand, which is explained in Algorithm 4. The combinations of subpaths are considered within this step. Each subpath represents a frequent itemset, with support value equal to the support of the least frequent node in the subpath. This logic sets basis for the parallelization of the algorithm, since conditional pattern bases can be handled independently. Experiment results have shown that the S-FPG performs about 15 times faster than PFP for bms-webview1 and 10 times faster than PFP for bms-webview2. The experiments also showed that the implementation scales well and efficiently processes large datasets, compared to PFP algorithm, as seen in Figure 3.4. Frequent pattern discovery is the most significant and time consuming phase of web usage mining. The explosive growth in website traffics and usage data has forced the studies to work on scalable methods for mining this usage data. Fp-growth approach is suitable for the task because of the structure's compact nature and its convenience for parallelization. Spark$^{TM}$ Framework offers in-memory calculations of parallelized tasks on clusters, which as a results makes it a suitable Framework for the task. Although there already are many implementations using Fp-growth based parallelization with Spark framework on various collections of web usage logs, and other types of data, the nature of the test data itself has not been inspected in accordance to the success of the implementations.



| | 1 | 10 | 20 | 40 | 80 | 160 | 320 |
|---|---|---|---|---|---|---|---|
| S-FPG | 9 | 12 | 21 | 19 | 31 | 36 | 51 |
| PFP | 105 | 114 | 130 | 218 | 352 | 642 | 1356 |

Figure 3.4 Scalability of S-FPG compared to PFP
(Source: Gassama et al., 2017)

# CHAPTER 4

# PARALLEL FP-GROWTH IMPLEMENTATION FOR WEB USAGE MINING

Even though there already are implementations of parallel Fp-growth using Spark framework on web usage logs and other types of data, the nature of the test data itself has not been inspected in accordance to the success of the implementations. This analysis is needed for comparing algorithms and applications in a reliable way. Since the nature of a website highly effects both the design goals and the statistical structure of the generated log files. For example, the logs collected from a website with a more content based nature that focuses on capturing user attraction for a long duration would be different in nature than the logs of a more transaction oriented website that focuses on selling a single item, or an online university platform that enables users to complete operations as quick as possible, etc. For this purpose, we have implemented parallel Fp-growth using Apache Spark$^{TM}$ for web usage mining on web usage logs with different characteristics. To be able to compare performance, we have used five different web usage logs as datasets. One of the datasets is taken from a real website server while the other four are generated in order to represent usage logs of different types of websites.

Section 4.1 describes the Fp-growth algorithm in detail. Section 4.2 describes the application of Fp-growth algorithm in parallel manner using Apache Spark framework and discusses the phases in detail.

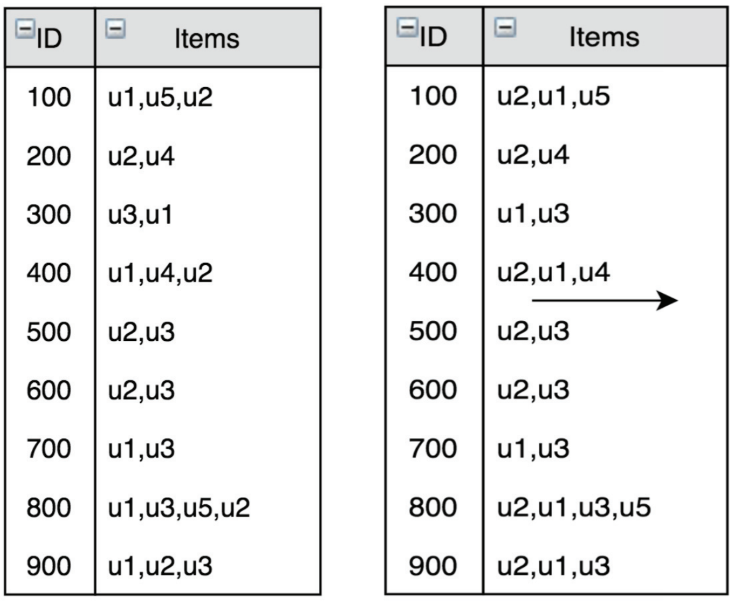
## 4.1. FP-Growth

A naive solution to the problem of finding frequent itemsets is generating all of the possible itemsets (candidates) and then decide on their usability by counting their occurrence. This approach is called the Apriori algorithm (Agrawal and Srikant, 1994). Solutions based on this approach are not easily scalable since the task at hand quickly becomes a combinatorial explosion problem because of the huge volume of input data. Another classic approach for finding frequent itemsets is the Fp-growth algorithm. Fp-growth executes without generating and then testing all candidates (Han et al., 2000).

The algorithm models the input data as a set of transactions, each of them containing a set of items. It then performs two DB-scans, one for calculating support values, filtering the infrequent items, collecting and sorting the sets of frequent items, and second for generating the respective suffix trees within the Fp-tree. The tree structure is a compact representation of frequent items and their respective support values. Nodes of the same item (request) that exists in multiple branches within the tree are linked to eachother using a map structure. This representation prevents any additional db scans to calculate support values and eliminates the need to generate candidate sets as in (Agrawal et al., 1996). Common prefixes between itemsets are stored within the same path in the Fp-tree by increasing the support values of common nodes. The compact design for managing nodes and supports, and the fact that only the frequent patterns that appear in a designated (minimum support) proportion of all transactions are inserted into the tree structure, results in a compressed and efficient representation of the data at hand, reducing memory consumption. Within the following step of the algorithm, this generated structure is used to extract the frequent itemsets from suffix trees.

***Preprocessing*** The datasets we have used each contain server level request logs of a website that represents a certain business model. Within our application domain, each transaction represents a single user session. Each item within a transaction itemset is a request for a source from the user, which represents an operation that the user has performed within that session. During the data cleaning, we have pruned requests that received an error as response, as identifying the underlying reasons of faulty requests was out of the scope of our study. While identifying sessions, we made an assumption that a user session would exist within a thirty minute time span, is not interrupted between log files, and categorized requests that were sent from the same IP address within that duration as the same session. Since we have collected server level data, we considered requests from the same IP address within a session time span are from a single user.

***Counting and ordering*** The data is modeled as sets of items for each transaction. A representation of this model is shown in Figure 4.1. Here, the ID column corresponds to a single session and the Items column corresponds to a series of web requests sent within that transaction. Within the initial state, the items in each transaction are not ordered in a meaningful way for our context. To be able to generate an Fp-tree efficiently, the items are first ordered by their frequencies. This is achieved by performing a db scan to count frequencies of items. Counting phase also enables the detection of infrequent items. Following the counting phase, each transaction is handled so that the items are ordered in a descending fashion. The phase finishes after all transactions ordered in itself. An example of this phase can be seen in Figure 4.1.



| ID | Items |
|---|---|
| 100 | u1,u5,u2 |
| 200 | u2,u4 |
| 300 | u3,u1 |
| 400 | u1,u4,u2 |
| 500 | u2,u3 |
| 600 | u2,u3 |
| 700 | u1,u3 |
| 800 | u1,u3,u5,u2 |
| 900 | u1,u2,u3 |

| ID | Items |
|---|---|
| 100 | u2,u1,u5 |
| 200 | u2,u4 |
| 300 | u1,u3 |
| 400 | u2,u1,u4 |
| 500 | u2,u3 |
| 600 | u2,u3 |
| 700 | u1,u3 |
| 800 | u2,u1,u3,u5 |
| 900 | u2,u1,u3 |

Figure 4.1 A sample db ordering process

***Tree generation*** Following the counting and ordering, the tree generation phase occurs. The generated tree structure for the sample db can be seen in Figure 4.2. Within this phase, the tree is initialized by creating a root and labeling it as null. The root is generally represented by {} Then each transaction is again handled one by one and added into the tree. For the first transaction, first item is added as a child of the root node. Second item is added as a child of the first item, third item is added as a child of the second item, and so on, until every item within the transaction is added into the branch. In order to track the support counts and nodes in the tree efficiently, a header table is generated. This table contains links to the items within the tree structure, making finding and tracking counts of items much more efficient. In case of an intersection between transactions, a new node is not added, instead the count value for corresponding node is updated. Note that this intersection has to start from the first element of both transactions. As opposed to this, the intersection may end anytime, causing the branch to split into different sub-branches. An example of this is the node labeled "u2:7" and "u1:4" in Figure 4.2. Since item "u2" can be found as the first item in 7 different transactions, every one of them uses the same node to be represented. However, since second items differ, the tree is split into three sub-branches after "u2". The same structure repeats after node "u1:4". This approach provides a compression of the data at hand, and hugely benefits the fact that the data at hand is ordered. After every transaction is included within the tree structure, ideally it is expected to compress data so that tree is smaller in size than the initial db structure.

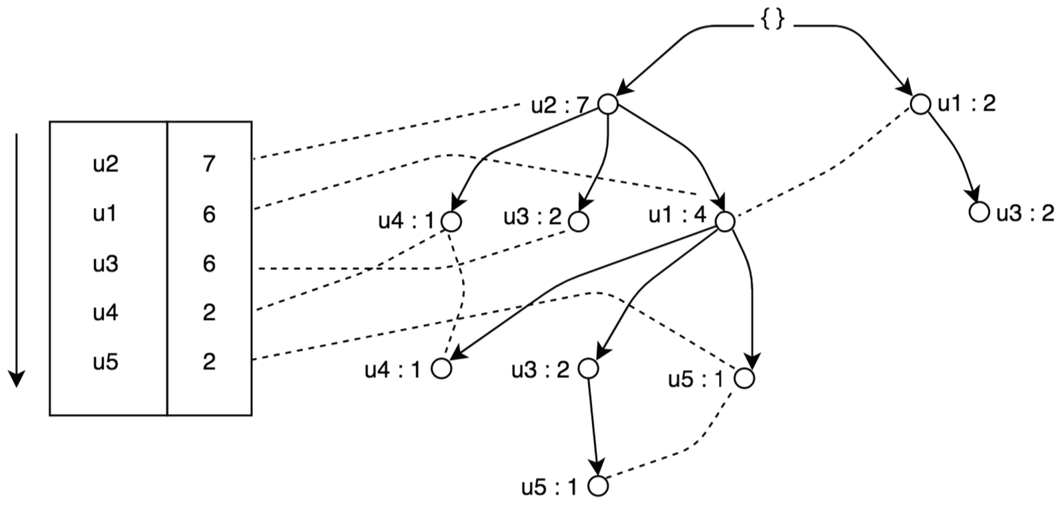


Figure 4.2 Fp-tree structure for sample db

***Mining*** After preprocessing the data and generating the tree structure, the mining phase begins. This time, the tree is handled starting from the leaves, the most infrequent items. After an item is selected, conditional pattern base and conditional sub-trees for that particular item is generated. This is done by tracking every path that ends with that particular item. Tracking every instance within the tree makes use of the header table. The count of the leaf or leaves containing current item is equal to the support of that particular conditional Fp-tree. Using this structure, the conditional pattern base is extracted. The conditional Fp-trees are then used for generating frequent patterns. Further mining phase examples are listed under Figure 4.3 for the sample db.

The conditional pattern base for "u5" from the sample db is :

$$u5 : \{\{u2, u1, u3 : 1\}, \{u2, u1 : 1\}\}$$

The conditional Fp-Tree is :

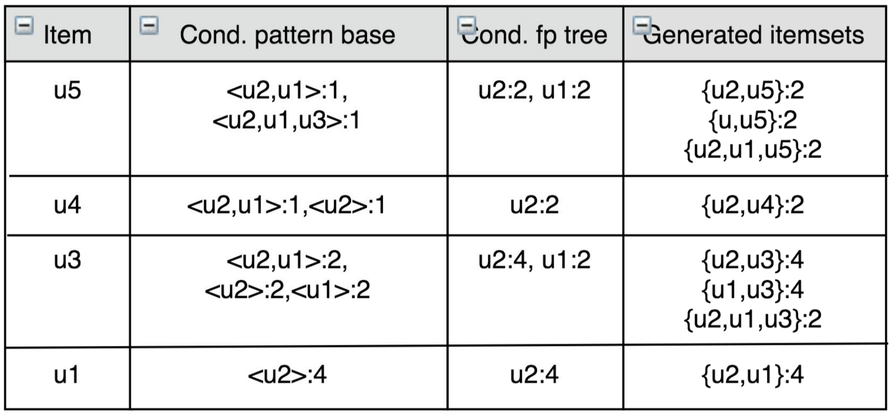
$$u5 : \{u2 : 2\}, \{u1 : 2\}$$

Frequent patterns for "u5" from sample db are:

$$u5 : \{u2, u1, u5 : 2\}, \{u1, u5 : 2\}, \{u2, u5 : 2\}$$

There are several bottlenecks of implementing the Fp-growth algorithm in a serial manner: 1. *Storage* For the cases where the size of the Fp-tree is too big to fit in memory or the disk, the need to divide the db at hand into partitions to represent the complete one arises. 2. *Computation distribution* Since Fp-growth is an algorithm suitable for parallelization, implementing the algorithm serially does not utilize it's advantages completely. 3. *Costly communication* Older implementations based on the idea of dividing the database by sequentially dividing the transactions has a high risk of seperate Fp-trees to depend on eachother. This results in the need of passing messages between threads and causes a high source consumption. High dependency between parallel threads of execution might cause an increase in execution time. 4. *Low support value* Setting the support threshold with a very small value to obtain longer sets of items may consume an unacaptable computational time, or the Fp-tree could overflow the storage. 5. *Complex error recovery* is hard to implement and consumes valuable sources.

Table 4.1 Conditional Fp-trees for sample db



| Item | Cond. pattern base | Cond. fp tree | Generated itemsets |
|---|---|---|---|
| u5 | <u2,u1>:1, <u2,u1,u3>:1 | u2:2, u1:2 | {u2,u5}:2 {u,u5}:2 {u2,u1,u5}:2 |
| u4 | <u2,u1>:1,<u2>:1 | u2:2 | {u2,u4}:2 |
| u3 | <u2,u1>:2, <u2>:2,<u1>:2 | u2:4, u1:2 | {u2,u3}:4 {u1,u3}:4 {u2,u1,u3}:2 |
| u1 | <u2>:4 | u2:4 | {u2,u1}:4 |

Although Apriori algorithm is also parallelizable (Li et al., 2012); considering the performance reasons and our need for applying the solution to large-scale web usage data, we have decided that a parallel implementation of the Fp-growth algorithm that uses map reduce jobs would better suit the needs of the current mining task at hand (Wang et al., 2008).

## 4.2. Parallel FP Growth Using Apache Spark

Since mining large web usage files requires a scalable and efficient approach, we have implemented the Fp-growth algorithm in a parallel manner, using the Apache Spark$^{\text{TM}}$ framework for the task. Spark$^{\text{TM}}$ (Zaharia et al., 2010) is an in-memory, iterative computing model and framework that was developed by the AMPLab of UC Berkeley. For managing clusters, Spark$^{\text{TM}}$ uses a cluster manager and is able to support Spark standalone, Hadoop and Apache Mesos. For managing the storage, Spark$^{\text{TM}}$ is able to support Hadoop's Distributed File System, Cassandra and Amazon S3 (Shi et al., 2017). Since

Spark$^{TM}$ allows loading data into memory by using "resilient distributed datasets", also referred to as RDDs, it is also suitable for executing iterative implementations. RDDs also provide a rebuild mechanism in case a partition is lost, increasing reliability of the system. Spark$^{TM}$ is suited for implementing data analysis, data mining and machine learning algorithms and aims to improve scalability by making use of RDDS, clusters and distributed machines. RDD architecture represents the dataset by splitting it into immutable partitions and storing them in memory on worker nodes of the cluster. Additionally, instead of writing intermediate results on HDFS, these results are kept in-memory.

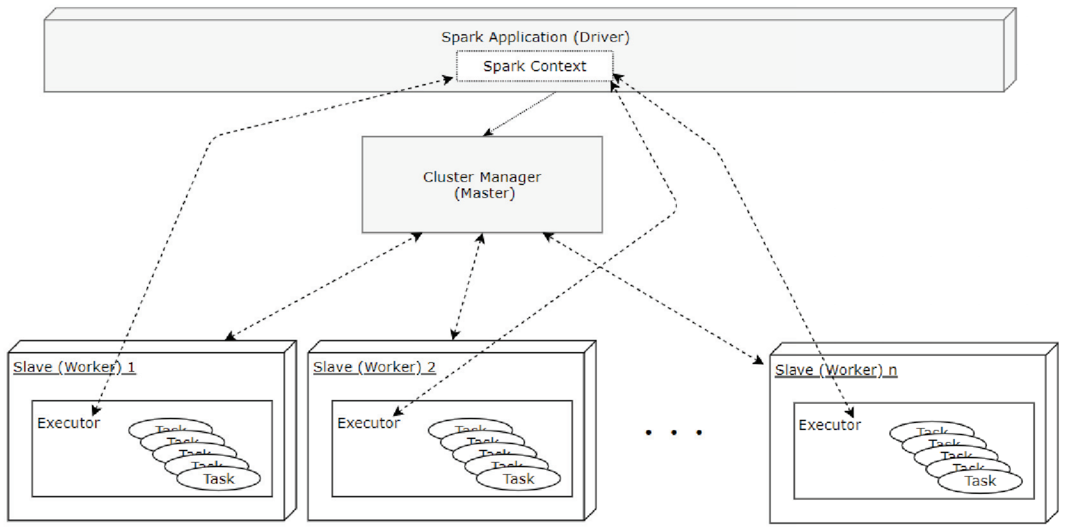


Figure 4.3 Spark architecture

This approach eliminates repeated access to the HDFS, reducing the I/O costs greatly and provides performance enhancements especially for iterative or interactive operations by allowing in-memory data sharing between multiple jobs. The partitions can be operated in a parallel manner. For certain applications, Spark's in-memory primitives provide performance up to 10 times faster in comparison to Hadoop's MapReduce which operates on disk space (Shi et al., 2017).

The general characteristics of Spark Architecture is as follows: Each Spark cluster has a Master and any number of Slave nodes. The master node of the Spark Application is called the Spark Driver. The driver contains a JVM which has Spark Context, which is used to contact with cluster manager or worker nodes directly. After the user code submits a job, the driver initiates the main function, schedules tasks and tracks the operations by using the directed acyclic graph (DAG) structure. An execution plan is generated at this stage. After planning the execution, driver than connects to the cluster manager to obtain resources. The cluster manager arranges the worker nodes (also called Spark Executors) and receives tasks from the driver to distibute among workers.

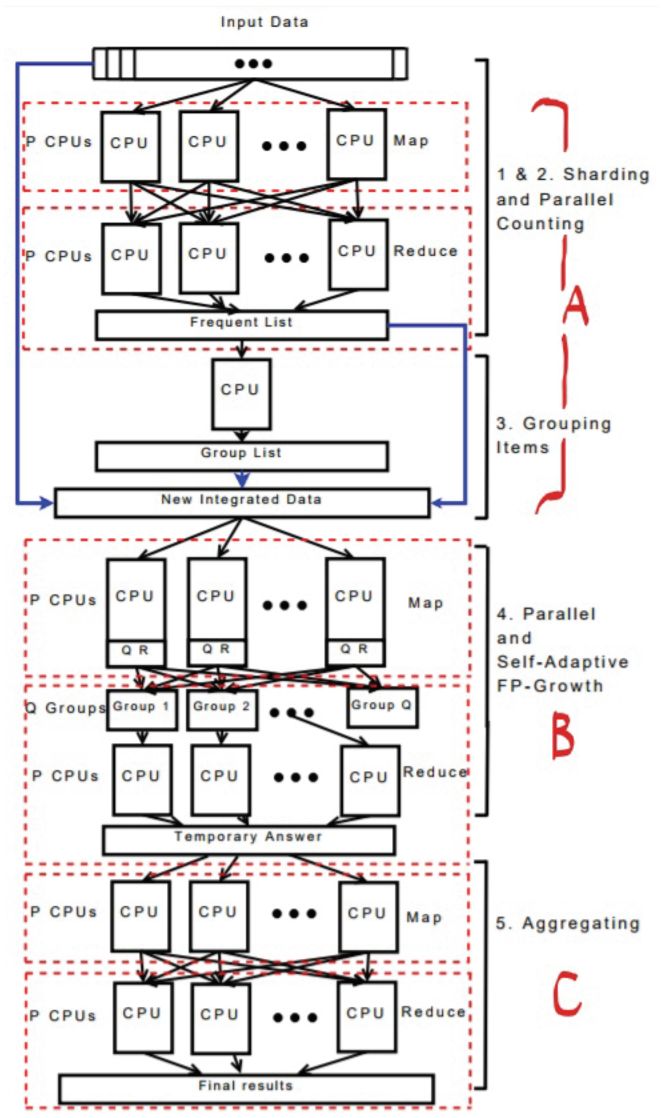


Figure 4.4 Parallel Fp-growth phases

(Source: Wang et al., 2008)

For our study, we have used the standalone version instead of managing a cluster. The driver is informed about the state of each executor and tracks the overall state at all times. Each worker node has a general executor java virtual machine, which is the core that executes specific tasks such as mapping or reducing based on the workflow's need. This structure can be seen in Figure 4.3. For our execution, we have used the standalone mode, using a single machine as a Worker, which contained multiple executors. The parallelized tasks such as counting items, MapReduce operations for various paralell Fp-growth stages such as grouping, generating group conditional transactions and finding frequent itemset sub-results by generating local Fp-trees are all independently executed by worker nodes.

A- Preparation

B- Applying parallel Fp-growth

C- Aggregating results

Finally, for fault tolerance, Spark provides tracking of individual nodes and transformation operations by using an RDD lineage graph. This graph helps for tracking failed nodes and recalculating necessary partitions when needed so that reliable results can be achieved. Steps of parallel fp growth implementation can be examined under three main phases for simplification. The phases are marked within Figure 4.4:

### 4.2.1. Preparation

Preparation phase consists of three main jobs: partitioning, parallel counting and ordering, and grouping the database at hand.

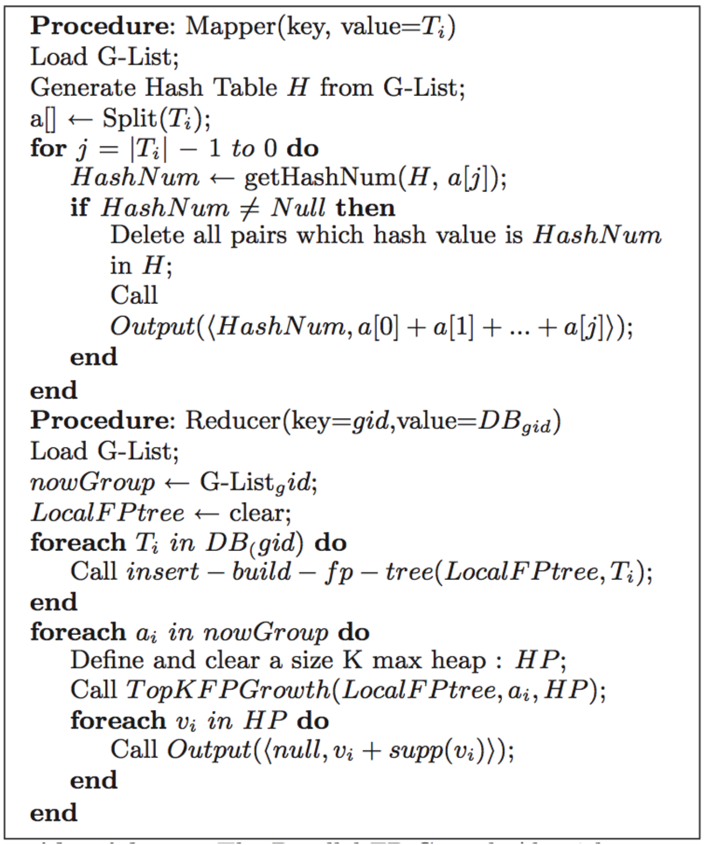
***Partitioning:*** This is also called "sharding" the database. Within this step the transactional data is partitioned into independent shards. These shards are then distributed amongst different computers.

***Parallel Counting:*** Within this step the support value of each item within partitioned transactions are counted in parallel(for each shard), by using a mapReduce pass. Within this step both a vocabulary of items(I) and their frequencies are discovered. The result is stored within the F-list. These count values are to be used for sorting the elements in itemsets in descending order. The time complexity is O(|I|).

***Grouping:*** All items are divided into seperate groups. These groups are referred to as the G-List. In order to manage seperate and parallel execution of the shards, each item in the G-list has a unique group id (gid). The time complexity of grouping can be expressed as O(|I|).

### 4.2.2. Parallel Fp-growth

The key step of parallel Fp-growth algorithm. Another mapReduce is executed within this step. The pseudocode of the phase can be seen in Figure 4.5. This phase is where Fp-growth algorithm is applied to the groups (G-List) at hand. The algorithm has two procedures; a Mapper() and a Reducer(). The output of this phase is the generated conditional local Fp-trees.



```
Procedure: Mapper(key, value=$T_i$)
Load G-List;
Generate Hash Table $H$ from G-List;
a[] ← Split($T_i$);
for $j = |T_i| - 1$ to 0 do
    $HashNum$ ← getHashNum($H$, $a[j]$);
    if $HashNum \neq Null$ then
        Delete all pairs which hash value is $HashNum$
        in $H$;
        Call
        $Output(\langle HashNum, a[0] + a[1] + ... + a[j] \rangle)$;
    end
end
Procedure: Reducer(key=$gid$,value=$DB_{gid}$)
Load G-List;
$nowGroup$ ← G-List$_g id$;
$LocalFPtree$ ← clear;
foreach $T_i$ in $DB_(gid)$ do
    Call $insert - build - fp - tree(LocalFPtree, T_i)$;
end
foreach $a_i$ in $nowGroup$ do
    Define and clear a size K max heap : $HP$;
    Call $TopKFPGrowth(LocalFPtree, a_i, HP)$;
    foreach $v_i$ in $HP$ do
        Call $Output(\langle null, v_i + supp(v_i) \rangle)$;
    end
end
```

Figure 4.5 Parallel Fp-growth algorithm
(Source: Wang et al., 2008 )

***Mapper:*** A shard of db is given to each mapper. The mapper also accesses the G-List, and processes the transactions in the shard one by one. During this phase, items are marked according to their gid's. Following this "marking" step, the algorithm handles transactions from each group's perspective, and items that do not belong to the group under examination are removed. This step is repeated for each group and each transaction that contains items from that group, causing transactions to dissect into group dependent transactions that are parallelizable without any information loss. After handling every transaction, the mapper phase finishes by returning (key,value) pairs. In these pairs, the key is the group-id and the value is the group-dependent transaction that was generated. The mapper then outputs key-value pairs such that each key is a gid (group id) and each value is its respective group dependent transaction that is generated within the mapping phase.

***Reducer:*** After the mapper instance finishes, mapReduce performs a grouping on all of the transactions to contain them in a shard. Following the generation, these shards are handled one by one, and a local Fp-tree is grown for each. Afterwards, these local trees are handled individually and conditional sub-trees of each are generated recursively.

### 4.2.3. Aggregation

This is the final phase where the output of parallel Fp-growth reducer phase is combined to get the final result set. This phase consists of only one step, Aggregating. The pseudocode of the aggregation algorithm can be seen in in Figure 4.6.

***Aggregating:*** The discovered patterns are aggregated to display a final result. This is achieved by executing another MapReduce pass. Aggregation receives the result of the Fp-growth phase as input. The algorithm stores the support values as data pairs in the Map. A data pair contains a key and the corresponding total support value of a particular item. After obtaining all data pairs the reducer selects the patterns with highest support values and returns. The final output of the aggregation phase is the top K patterns according to their support values for each item, which are the final list of frequent itemsets.

```
Procedure: Mapper(key, value=$v$ + supp($v$))
foreach item $a_i$ in $v$ do
    Call Output($\langle a_i, v + \text{supp}(v)\rangle$);
end
Procedure: Reducer(key=$a_i$, value=$S(v + \text{supp}(v))$)
Define and clear a size K max heap : $HP$;
foreach pattern $v$ in $v$ + supp($v$) do
    if $|HP| < K$ then
        insert $v$ + supp($v$) into $HP$;
    else
        if supp($HP[0].v$) < supp($v$) then
            delete top element in $HP$;
            insert $v$ + supp($v$) into $HP$;
        end
    end
end
Call Output($\langle null, a_i + C\rangle$);
```

Figure 4.6 Aggregation
(Source: Wang et al., 2008 )

The main space consumption of aggregation is caused by managing the heap structure, which is bound by K, resulting in a complexity of O(K). With |I| being number of items, K size of max heap and P number of machines that the load will be distributed amongst, is O(|I|*Max(NumOfItemRelatedPatterns)*log(K)/P) denotes the time complexity.

In order to observe how the algorithm performs in accordance with various characteristics of data, the parallel Fp-growth algorithm is applied on five different datasets: a dataset taken from a busy web server, a content based website, a transaction based website, a social media website and a product website. Each of the datasets has different characteristics that represent a different business model. The following chapter discusses the experiment results in detail.

# CHAPTER 5

# EMPRICAL STUDY

Five different datasets were used for experiments. One of the datasets was taken from an actual server, the other four were generated to represent different business models. Different types of websites were represented by synthetic datasets were modeled by using various number of transactions, length of transactions, correlation between patterns and number of individual items. The models represented are: a content based website, a transaction based website, a social media website and a product website. Parallel Fp-growth algorithm was executed to observe how well the algorithm scaled for each model with changes in file size, minimum support threshold value and number of transactions. Experiments were repeated using different numbers of executors to observe parallelization performance.

Within this chapter, the first section describes our simulation environment. Section 5.2 describes the datasets at hand. Experiments are given for Nasa weblogs and synthetic weblogs under section 5.3. Finally, the experiment results are discussed under section 5.4.

## 5.1. Simulation Environment

During knowledge discovery phase, we executed the Fp-growth algorithm in two versions: one to be executed serially, the second to be executed in a parallel manner. In order to compare performances, we have applied both approaches using various file sizes between 25MB and 2GB and various minimum supports for performance comparison. The experiments were conducted on a machine with 16GB of RAM, Intel(R) Core(TM) i5-6500 CPU @ 3.2GHz processor with 4 cores and 6 MB SmartCache, running on a 64-bit operating system. Because of our hardware constraints, we were able to test the parallel execution up to 4 nodes, but the implementation can easily be executed with more nodes by providing a larger cluster. Since FP growth has the largest constraint regarding memory because of the in-memory FP-Tree, each node had a memory of 2GB. For larger file sizes, scaling up would possibly be needed.

## 5.2. Datasets

Two kinds of datasets are used for experimentation. NASA weblogs, taken from a real web server, and synthetic datasets, generated with IBM's synthetic data generator, provided by QUEST big data research group (IBM QUEST, 1996). We have applied minor modifications to the generator in order to be able to execute it on a more recent version of a development environment, and to better fit our parser's needs. Since a single dataset is unable to represent various scenarios that we aim to inspect, we have used IBM's data generator to include different business models in our study. Another reason that required generation of synthetic datasets is that real data is hard to obtain by nature. There are various causes for not being able to capture usage logs completely: Browsers generally offer mechanisms that cache last visited pages. These mechanisms might prevent capturing some of the browsing behaviour. This case also occurs when users hit backspace to return to the previously viewed page. Another scenario that prevents capturing real browsing behaviour is shared computers. For these machines it is not possible to discriminate individual users unless a logging mechanism is introduced. Although logging mechanisms are able to capture realistic data, they cause user to enter information and log in every visit, which might effect user behaviour or prevent visits or logging in altogether because of the time consumption factor. Another factor causing distortion in real data is users that regularly switch machines or use multiple at the same time. Since there is no other mechanism to prevent this other than logging in scenario, it is highly possible that same consequences will apply here as well. The other factor preventing to capture realistic web traverse data even when using real server logs is that user might get distracted for a time and considered a new session altogether during the same visit. The length of this time frame is an approximation and up to the data analyst, hereby effects the results. As a solution to capture better logs, cookies or specific browsers have been proposed, but these structures rely completely on user cooperation and raise privacy concerns.

### 5.2.1. NASA Weblogs

We acquired web server logs from a busy log server in order to mine for informative patterns. During preprocessing, we have identified different users and urls by parsing the server logs. After preprocessing, we sessionized the server log data with the assumption that a single session has the maximum duration of thirty minutes and categorized all requests from a single IP within a this time interval as a single session. As cleaning phase, we have considered unsuccessful requests with code 404, requests other than UTF-8 and items -request urls for our domain- with very rare (less than 500) occurrence as outliers and eliminated them from our set of items. Although they are still included for counting page hit counts, sessions that contain only one request are not eligible for pattern discovery so they are removed. Basic characteristics of the logs at hand can be examined in Table 5.1. After grouping sessions and ordering itemsets by hit count, the number of valid sessions for Fp-tree generation is 124,406 as seen in Table 5.1.

Table 5.1 Number of frequent itemsets for each minimum support

| | |
|---|---|
| number of requests | 1,509,890 |
| number of requests after cleansing | 1,498,369 |
| number of distinct transactions | 272,230 |
| number of sessions | 146,521 |
| longest session length | 250 |
| valid sessions for Fp-tree generation | 124,406 |

## 5.2.2. Synthetic Datasets

To be able to achieve the metrics of each business model's requirements, we have determined four different types of websites.

Table 5.2 Characteristics of the logs from different platforms

| Type | Transaction count | Transaction length | Correlation between patterns | Number of items |
|---|---|---|---|---|
| Content based | high/very high | long/very long | high (for a given time) | high |
| Transaction based | average | short | very high | low |
| Social media | very high | long/very long | low | very high |
| Product website | low | very short | very high | very low |

As seen in Table 5.2, environments that we intend to inspect are: (1) Content based website: In this structure content is produced by professionals. The main objective is to capture attention of as many users as possible for as long as possible. The contents tend to be long, such as long texts or videos. Page hits are expected to be high in an ideal case, and the itemsets tend to be longer since capturing user attention is wanted. News websites, online publishing platforms, entertainment websites could be listed under this type. (2) Transaction based website: These platforms are designed for specific processes, such as an online system to manage accounts or a university website designed to manage lecture enrollments etc. Other examples could be library, hotel or sports club websites. In these websites the objective is highly defined and limited, which makes them simpler in structure in most cases. Page hits may vary from average to high and peaks might occur in visits depending on the domain. Ideally, itemsets tend to be shorter since the goal is to help user reach and complete actions as quick and easily as possible. Number of individual items may vary, again depending on the domain. (3) Social media : These are platforms where users are able to produce and share content quickly.

Table 5.3 Characteristics of synthetic web usage logs

| Type | Transaction length (10) | Correlation between patterns (0.25) | Number of items (100.000) |
|---|---|---|---|
| Content based | 50 | 0.35 | 250 |
| Transaction based | 8 | 0.5 | 50 |
| Social media | 25 | 0.1 | 500 |
| Product website | 3 | 0.5 | 10 |

The main goal is to encourage users to produce content as frequent as possible and to capture user attention as much as possible in order to get high page hits. In these websites the number of items (pages) tend to be higher since content is produced very frequently by users. The website is designed for attracting user attention so itemsets are usually longer. (4) Advertisement/Product website : A small and focused platform, the goal is to direct users to a single sales or information page. The number of items (pages) can be very low in order to direct user attention to the designated product page. Because of the focused structure, very high similarity between different visits and shorter transactions are expected. Similar to transaction based websites, the goal is to make it easy for the users to access the relative content as quick as possible. The actual numbers that correspond to representations from Table 5.2 are as shown in Table 5.3. The values were determined by taking average values and the needs of the dataset into consideration. These values can be seen in Table 5.3.

## 5.3. Experiments

Experiments were conducted on five datasets, divided into two types: Web usage logs taken from a real server and synthetic web usage logs. The two will be examined under separate sections, 5.3.1 and 5.3.2. For each type two different approaches have been pursued in order to observe how the algorithm scales for NASA weblogs: First set of experiments were done by using a constant file size and a variant minimum support value. A change in the minimum support value directly effects the number of frequent itemsets and the size of the Fp-tree. Thus, as the size of the tree increases an increase in execution time is expected. Second set of experiments were performed by changing the file size for the same value of minimum support. An increase in the file size naturally results in a higher volume of data, similar to dropping the minimum support value. Experiments were performed for ten different file sizes in order to observe how the algorithm scales. Finally, all experiments were repeated for different levels of parallelization to be able to observe the difference between parallel and serial executions.

### 5.3.1. NASA Weblog Results

The size of the Fp-tree is in relation with the number of the frequent items in the database. To estimate the approximate size of the FP Tree, Table 5.4 can be examined for numbers of frequent itemsets for each minimum support value. The height of the Fp-tree depends on the length of the maximal frequent itemset within the database (250 in this case). Keep in mind that with an increase in the number of common itemsets, which completely depends on the data set instance at hand, the size of the tree would drop.

***Support Impact***: The difference in execution times for various minimum support values varying between 0.01 and 0.09 have been observed as in Figure 5.1. The dramatic increase in time consumption with lower minimum support is caused by the significant increase by 28473 itemsets between 0.01 and 0.02 minimum supports. Minimum supports that are above 0.1 decrease the number of frequent itemsets drastically, causing the remaining data lose its ability to represent details. In order to examine meaningful changes considering time consumption, we have used minimum support values between 0.01 and 0.09 for testing.

Table 5.4 Number of frequent itemsets for each minimum support

| minimum support | frequent itemsets |
|---|---|
| 0.01 | 36500 |
| 0.02 | 8027 |
| 0.03 | 3459 |
| 0.04 | 1552 |
| 0.05 | 964 |
| 0.06 | 777 |
| 0.07 | 508 |
| 0.08 | 283 |
| 0.09 | 137 |
| 0.1 | 114 |
| 0.2 | 47 |
| 0.3 | 33 |
| 0.4 | 11 |
| 0.5 | 2 |

***Transaction Number Impact***: A set of experiments were performed to observe the relatioship between number of transactions and the execution time of the algorithm. Since the characteristics of data is stable for this single dataset, effects of number of transactions is directly in accordance with the file size. The initial log file consisted of 1,5 M lines of server logs. Following the preprocessing phase, in which error responses and non UTF-8 formatted requests were cleansed, 1,49M lines of requests remained. These requests were sessionized using our sessionization algorithm which took 30 minutes as a time window for a single session. This phase resulted with 272K transactions. This base file has been used to obtain log files that contain various numbers of transactions between 1K and 6M The file size growth in accordance to the number of transactions contained has been observed as seen in Figure 5.2.

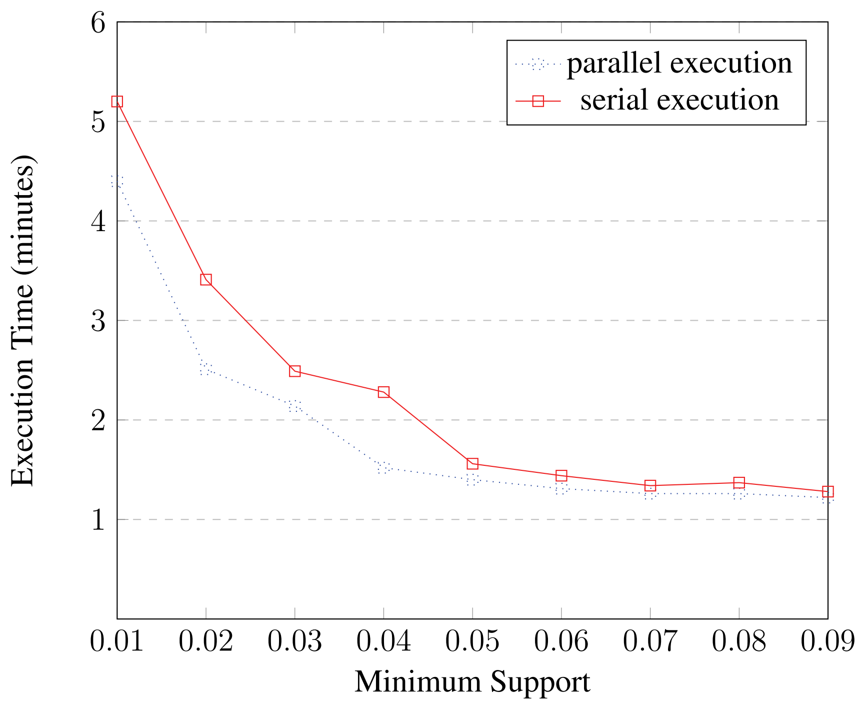


Figure 5.1 Execution time with support change (0.01-0.09)

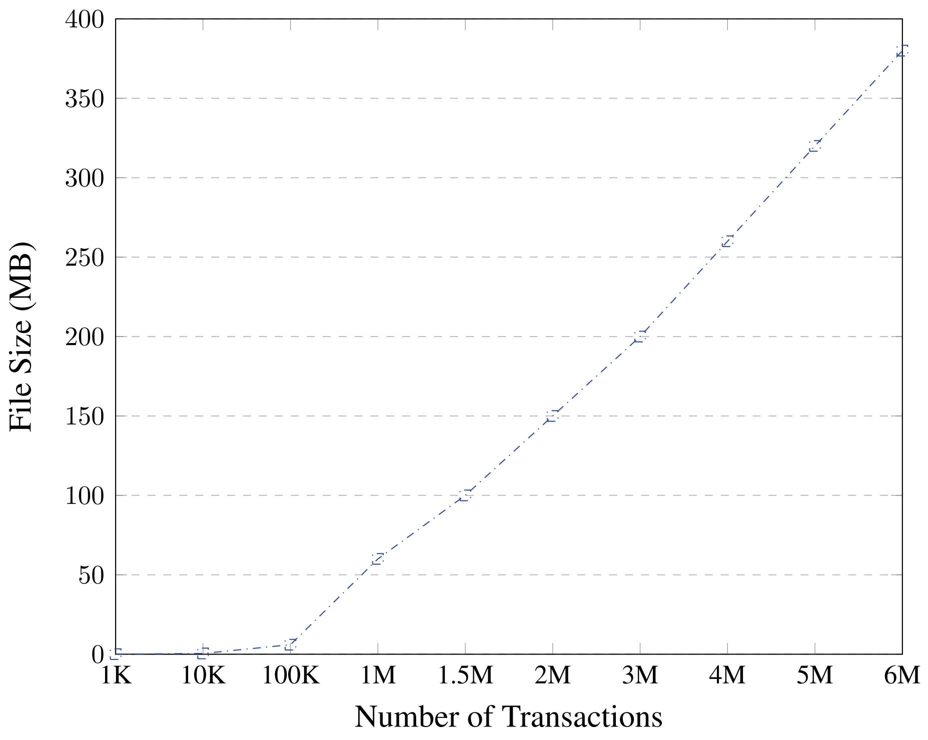


Figure 5.2 Log file sizes for different transaction volumes

The average length of transactions contained in a sessionized NASA weblog file is around 15. This fact causes both the growth rate of the size of the log files and the execution times in accordance with the number of transactions to remain small. However, number of unique items are high for NASA dataset. This causes the size of the Fp-tree to grow, reducing compression, and also factoring in a minor increase in execution time of the algorithm. The increase in execution times have been observed as close to linear, as seen in Figure 5.3.

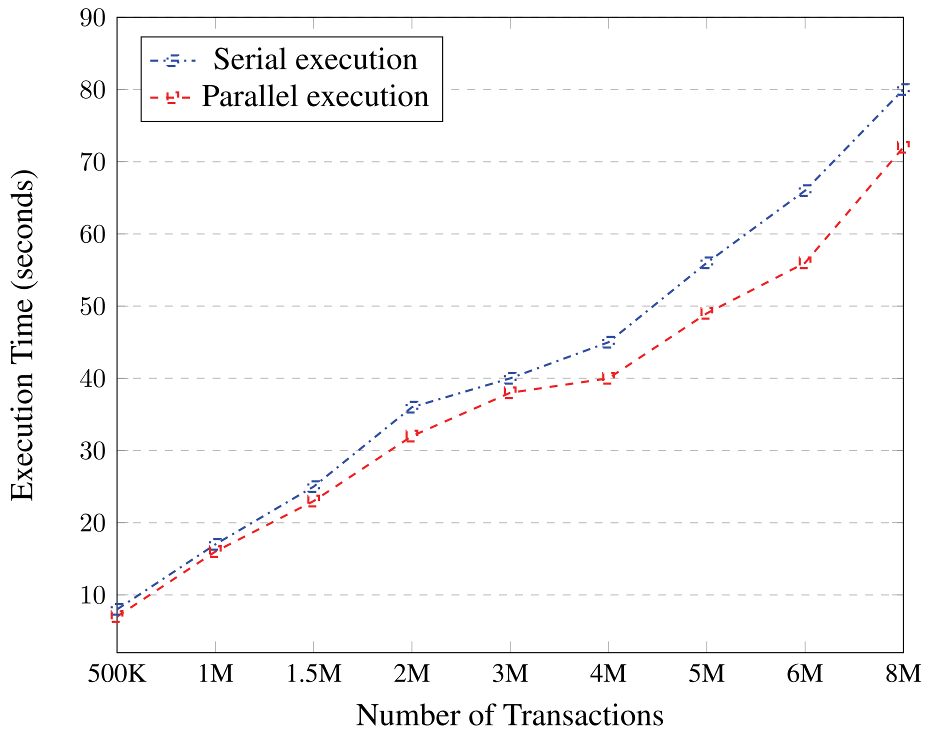


Figure 5.3 Execution time with transaction volume change

***File Size Impact***: Experiments were conducted to observe the difference between serial execution and different levels of parallelization for various file sizes. For these experiments, the lowest minimum support, 0.01 has been used to obtain the largest frequent item space. Time consumption has been observed as scaling slightly higher than linear for file sizes 15 MB, 25 MB, 50 MB, 100 MB, 220 MB, 450 MB, 700 MB, 900 MB, 1GB, 1.5 GB and 2GB, as seen in Figure 5.4. The maximum improvement regarding time consumption is observed between serial and 4-parallel execution for 2GB file as expected, approximately differing by 30 seconds. Note that since Fp-growth is a memory-intensive algorithm, increasing file size too much so that memory of nodes are not sufficient anymore would increase the performance of the algorithm, possibly causing dramatic execution time increases caused by high I/O cost from switching to disk read and write operations.

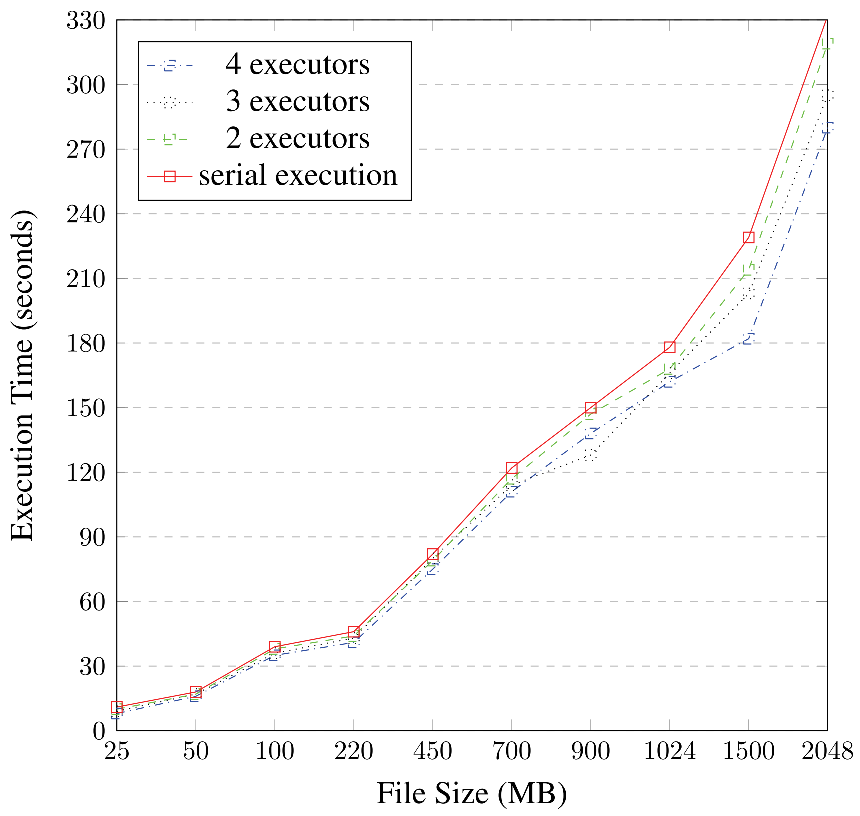


Figure 5.4 Execution time with file size change

Comparisons between parallel and serial execution time for different file sizes has been observed as in Figure 5.5. The decrease in execution time is about 10 seconds for the smallest file, whereas the decrease for a 2GB file is about a minute. It can be seen from the results that as the file size grows, the enhancement in the execution time gets higher.

About %10 enhancement in time consumption has been observed for all file sizes by using 4-Parallelization istead of serial execution. A ratio can be seen between the enhancement in time consumption and parallelization. This ratio has been examined in Figure 5.6. While 2-Parallelization resulted in a %1-4 reduce in time consumption for different sized files varying between 25 MB and 2 GB, 3-parallelization has resulted in a %5-9 reduced time consumption, and 4-parallelization resulted in an enhancement between %10-11 regarding time consumption. The results suggest that a greater improvement could be achieved either by increasing the file sizes, which would result in a time improvement with the current fixed improvement percentage, or by increasing the number of parallel nodes, which would enhance the efficiency by a new ratio that is above %10.

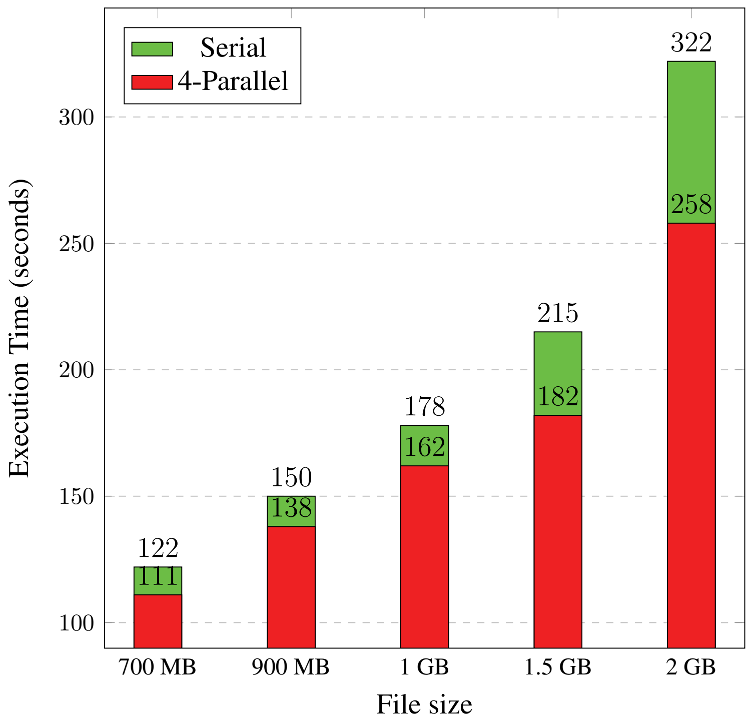


Figure 5.5 Execution time with file size and executor number change

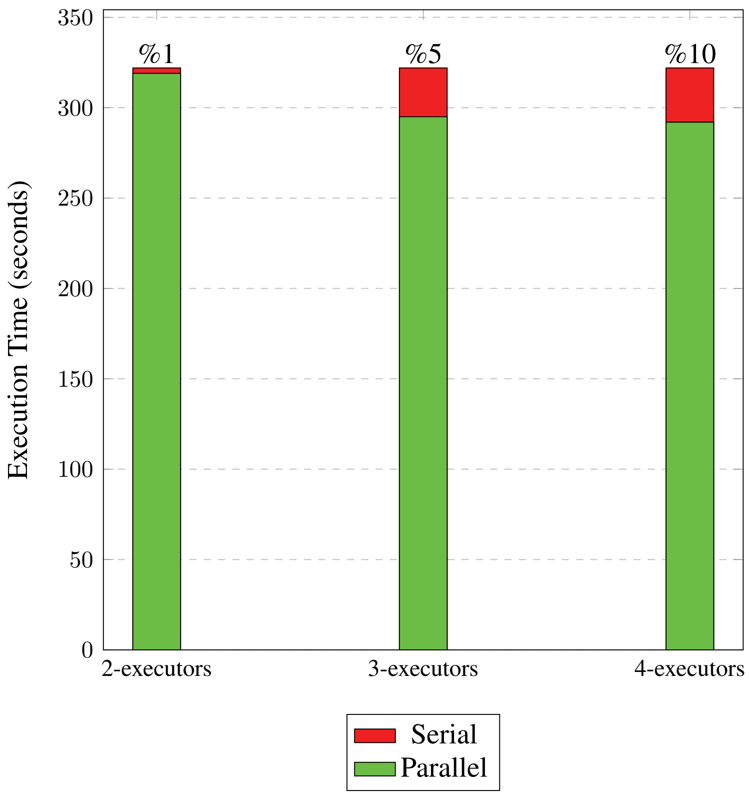


Figure 5.6 Relative execution time with executor number change

### 5.3.2. Synthetic Dataset Results

In order to observe how the nature of the data at hand effects performance and scalability, experiments for both support and file size impact were performed for different levels of parallelization for all four types of synthetic datasets.

***Support Impact***: Each web usage log was tested using various minimum support values varying between 0.01 and 0.06. Files of size 2GB were used. All other characteristics remained in accordance with the characteristics of the dataset.

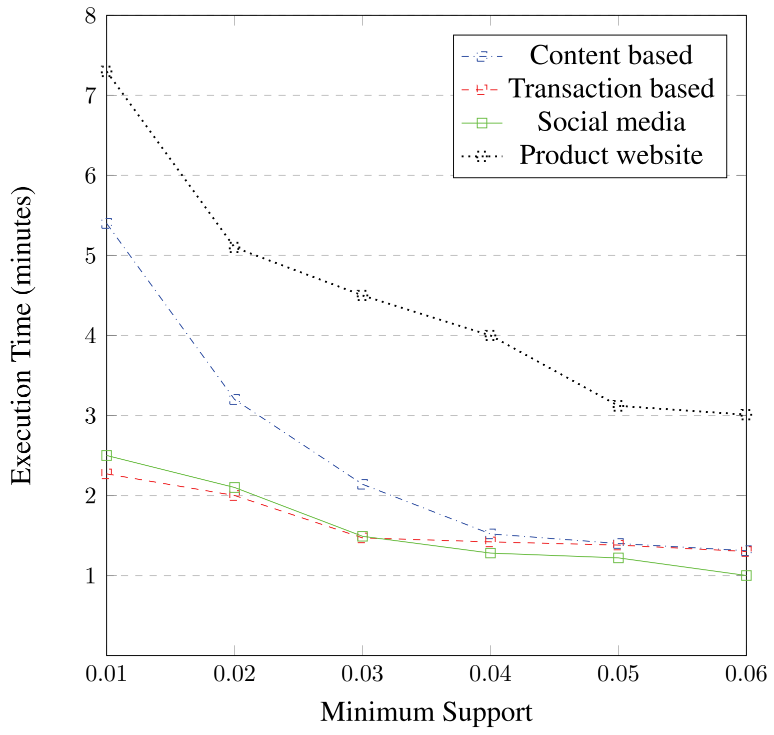


Figure 5.7 Execution time with support change (0.01-0.06)

As seen in Figure 5.7, product website has a much higher execution time, almost 4 times of social media and transaction based websites. Content based website has scaled second worst, following product website. Product website has a very low number of items, which casuses intersections and results in a dramatic increase in number of frequent itemsets. As number of frequent itemsets increase, the tree size grows and the algorithm consumes more space and time in return. In contrast, the second type of website that consumed most time is content based website. Unlike product website that has 10 different items, content based website has a higher number of items, which is 250. Another major cause for the increase in execution time for content based website is the transaction length, 50, which is the highest among all types.

***Transaction Number Impact***: We created synthetic log files that contain different number of transactions varying between one thousand and six million. The relationship between number of transactions and size of log data depends very much on the length of transactions, as observed in Figure 5.8. For content websites, which have the longest transactions, 50 as average transaction length, the file size increases faster as the number of transactions grows.

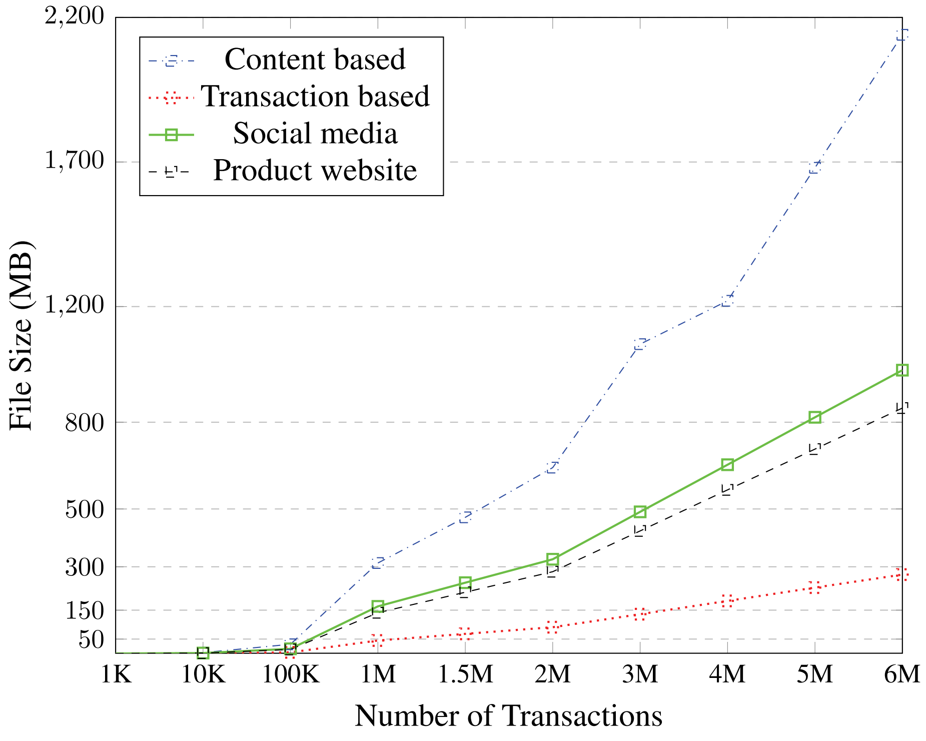


Figure 5.8 Log file sizes for different transaction volumes

Content based website is followed by product website, which has 30 as average transaction length, and social media, which has 25 as average transaction length. Transaction based websites have the least average transaction length, as 8, causing them to grow the slowest as the number of transactions increase. For web usage logs that contain longer traversal information file size grows faster as number of transactions increase since each transaction will be containing more data. Logs of content based website grow the fastest with the increase in number of transactions, followed by social media and product websites, and lastly transaction based websites. The increase in volume directly applies to the execution time as an additional cost. Consistent with the volumes seen in Figure 5.8, a similar increase in execution times for the algorithm has been observed, which is shown in Figure 5.9. Content based website logs consume the most time, followed by social media and product websites, and lastly transaction based websites. There is a difference between the execution times and volume increases of social media and product websites, which is caused by the very low number of items, 10, for product website in contrast to social media, which has 500 different items.

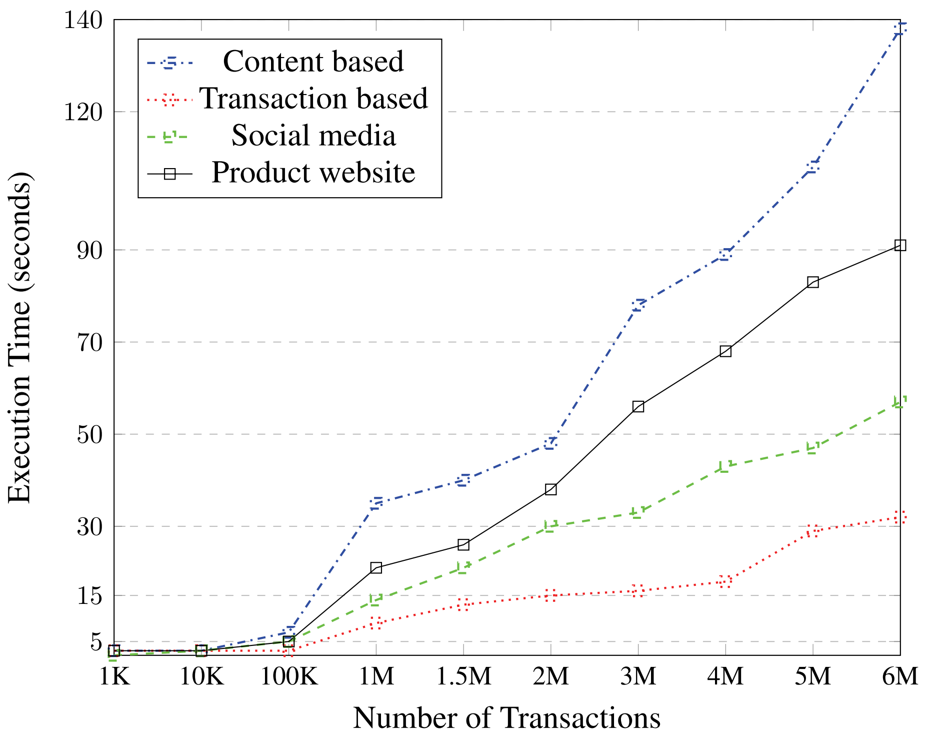


Figure 5.9 Execution time with transaction volume change

***File Size Impact***: A second observation for scalability was made, this time taking file sizes into account. We have created log files for all types of websites with different sizes varying between 25 MB to 2 GB to examine how much time each type of log consumed. As seen in Figure 5.10, product website scales the worst amongst all types, followed by content based, transaction based and social media. Product website scales the worst in execution time because of the fact that it has a very low number of items, causing a peak in transactions that contain frequent itemsets, which causes the data to be handled to grow. The second type of website that has the highest execution time is content based, which is mostly caused by it's huge transaction length, 50. The difference between execution times based on file size also highly depends on the differences between transaction lengths, since log types with short transaction length require much more transactions to satisfy the needed file size. This relationship has been explained in detail in figure 5.8.

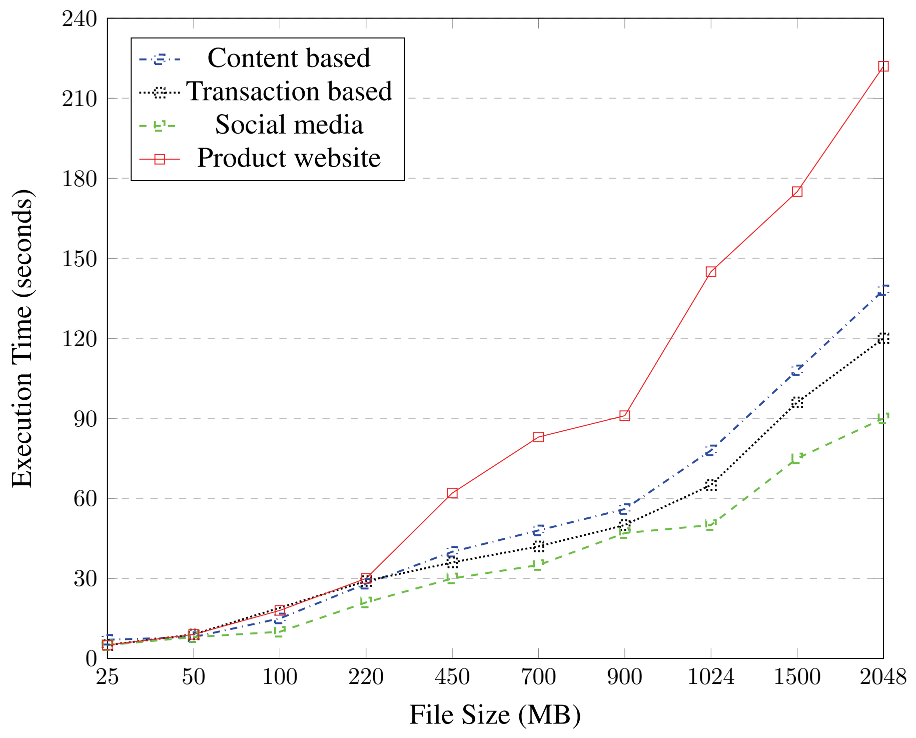


Figure 5.10 Execution time with file size change

## 5.4. Discussion

The experiments suggest that the performance of the algorithm is highly effected by the number of items, number of frequent items, transaction length, correlation between patterns, minimum support threshold value and file size. This can be observed in experiments with varying minimum support values.

For the minimum support experiments, results are also highly effected from the fact that the size of the log files are all 2 GB. With very low transaction length, number of transactions need to be very high in order to compensate and achieve the same file size. Traversal of all transactions and counting costs tend to increase. Branches of the tree tend to be shorter but the number of branches might increase, depending on the number of different items, which would increase the execution time. Similarly, an increase in minimum support effects social media dataset worse, since it has the lowest correlation thus the lowest compression of the data at hand. As a result, Product website, which has very low transaction length and number of items, results with the worst execution time. The bigger the transaction length is, the faster the log files grow with each transaction. When evaluating the performance of an algorithm using execution time to number of transaction ratio, datasets with higher transaction length have a higher probability to execute for a longer period of time. Since the Fp-tree depth is bound by the longest transaction at hand, the tree size might increase as transactions get longer. The execution time also depends on file size. Experiments suggest that the execution time is much bigger for datasets with short transactions. More transactions are needed in order to achieve the same file size for shorter transactions. The results of execution time with file size change experiments support the results previously found in execution time with support change experiments.

# CHAPTER 6

# CONCLUSION

The World Wide Web is being used more than ever, causing an explosive growth in website traffics and the amount of information consisting user activities on websites. Moreover, the popularity of www has also caused an increased competition between websites with similar content. Gaining valuable user attraction and becoming the preferred alternative has become primary concern. For this purpose, websites are analyzed and enhanced continuously. Enhancements can vary between updating content to reorganizing the structure of the website. Although the enhancement decisions can be made implicitly; results of implicit designs are rarely optimal and the risk of losing precious user attention arises. Enhancement decisions made scientifically by evaluating how users actually perceive and interact with the website have a far better potential to succeed. Deriving information from actual usage data such as web server logs could be inspected in order to propose better improvements. This could be achieved by finding frequent, informing patterns otherwise unknown to gain insight of how the website operates in practice. The methodology for finding frequent patterns should require as little manual interference as possible. Since the volume of web server log data is enormous, the proposed scientific methods should be automatable. Automation is advantageous both for reusability and for obtaining realistic results for longer durations.

For all the purposes stated above, we have investigated some of the previous approaches for frequent pattern mining, and propose a solution that is based on the Fp-growth algorithm, which is a scalable and efficient approach (Han et al., 2000). By removing the costly phases of candidate generation, candidate testing and repetitive db scans, Fp-growth algorithm performs better then it's ancestors. The algorithm also offers a divide and conquer based approach, making it easy to parallelize, scale and to automate.

In this study, Fp-growth algorithm (Han et al., 2000) is applied to the domain of web usage mining. The initial goal of applying data mining methodologies to this domain is to discover meaningful insights from web usage logs. This information could be directive for achieving a better website structure, utilizing resources efficiently, dropping maintenance costs, improving the user experience and engagement rates. Although there are various studies using Fp-growth for the task of web usage mining, it is hard to evaluate the relative performance of an application for different types of websites. An inspection of the relationship between the characteristics of data at hand and the success of the mining process have yet to be studied. As the primary motivation of this study, we applied the algorithm on five different datasets with varying characteristics in order to examine this relationship.

Firstly, in order to execute the Fp-growth algorithm in a parallel manner, the database is sharded into smaller pieces using conditional sub-trees to execute independently. Implementing the Fp-growth algorithm in a parallel manner is expected to improve the overall performance of the application. Our experimental results also support that parallelizing the mining process of web usage data enhances the efficiency of extracting meaningful website traversal paths among website usage logs. We have observed between %1-6 reduced time consumption between serial execution and 2-parallelization, %6-9 reduced time consumption between serial execution and 3-parallelization and %8-10 reduced time consumption between serial execution and 4-parallelization for different sized files varying between 25 MB and 2 GB. Secondly, we applied the algorithm on five different datasets with varying characteristics. We used one dataset that is extracted from HTTP logs of a busy WWW server (Wang et al., 2008), and generated four different synthetic datasets, each designed to represent a different business model. The algorithm is applied on usage logs of all types and results have been observed in accordance with each dataset's characteristics.

The experiments suggest that the performance is highly correlated with the number of items, number of frequent items, transaction length, similarity between frequent patterns, minimum support value and size of the log file. Log file size, by directly effecting the size of the tree, has an immediate effect on execution time. Although number of items is important when mining for frequent patterns, either by effecting the size of the tree and the table structure, it is also important how similar the frequent patterns are. The higher the correlation between patterns, the smaller the tree structure, which in return causes a faster execution time. For datasets that contain a very low number of items, the transactions are very high in similarity, which is expected to increase performance. However, as our experiments suggest, when given a low minimum support or a huge file size, these features might cause an explosive increase in number of frequent itemsets and number of processed transactions, drastically dropping performance. Similar to this problem, with very low transaction length, file sizes tend to get smaller. In order to achieve the same file size and to compensate for the shortness of the transactions, the number of transactions need to be very high, causing the execution time to increase. An increase in number of items is prone to cause an increase in the width of the tree by introducing separate frequent paths. Similarly, as correlation between patterns drop or number of items increase, the size of the tree structure grows, causing the execution time to increase.

As a future study, larger log files might be inspected with a larger scale of parallelization. Secondly, the same principles can be applied to evaluate different business models. Moreover, evaluations could be enhanced to observe the relative effects of different types of characteristics. Furthermore, the results obtained from mining web usage logs can be made use of when observing specifics about web usage information of a website, such as when executing A/B testing on users to compare results for each website version. The logs collected from different groups of users can be examined seperately to compare results in a scientific manner. Lastly, mining sets of frequent pages can be used in combination with another methodology such as clustering or classification to categorize frequent sets into frequent set classes or clusters, which then can be used to enhance user experience and provide personalized suggestions or adaptations specifically for related users.

# REFERENCES

Agarwal, R. and Psaila, G. (1995). Active data mining. in proceedings on knowledge discovery and data mining. *KDD-95*.

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. (1996). Fast discovery of association rules. *Advances in Knowledge Dis-covery and Data Mining*.

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. *In-VLDB*.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. *ICDE*.

Apte, C. and Hong, S. (1996). Predicting equity returns from security data. in advances in icnowledge discovery and data mining. *AAAI Press and the MIT Press*.

Brin, S., Motwani, R., and Silverstein, C. (1997). Generalizing association rules to correlations. *InSIGMOD*.

Burdick, D., Calimlim, M., and Gehrke, J. (2001). Mafia: A maximal frequent itemset algorithm for transactional databases.

Canada, C. I. M. (2015). Microsoft attention spans. *Spring*.

Cheeseman, P. and Stutz, J. (1996). Bayesian classification (auto class): Theory and results. *Advances in Knowledge Discovery and Data Mining, American Association for Artificial Intelligence, Menlo Park*.

Chen, D., Lai, C., Hu, W., Chen, W., Zhang, Y., and Zheng, W. (2006). Tree partition based parallel frequent pattern mining on shared memory systems.

Cohen, E., Krishnamurthy, B., , and Rexford, J. (1998). Improving end-to-end performance of the web using server volumes and proxy filters. *In Proceedings of ACM*

*SIGCOMM-98'*, pages 241–253.

Cooley, R., Mobasher, B., and Srivastava, J. (1997). Web mining: Information and pattern discovery on the world wide web. *In Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence-ICTAI' 97, IEEE Computer Society*.

Cooper, M., Foote, J., Adcock, J., and Casi, S. (2003). Shot boundary detection via similarity analysis. *In Proceedings of TRECVID 2003 workshop*.

Dean, J. and Ghemawat, S. (2004). Simplified data processing on large clusters. *OSDI*.

Dunham, M. H. (2003). Data mining introductory and advanced topics. *Pearson Education*.

Fayyad, U., Shapiro, G. P., and Smyth, P. (1997). From data mining to knowledge discovery in databases. *AI Magazine*, 3(17):37–54.

Fayyad, U. M., Djorgovski, S. G., and Weir, N. (1996). From digitized images to on-line catalogs: Data mining a sky survey. *AI Magazine*, 17:51–66.

Galitsky, B., de la Rosa, G. D. J., and Kuznetsov, S. (2011). Using generalization of syntactic parse trees for taxonomy capture on the web. *ICCS*, (5):1163–1177.

Gassama, A., Camara, F., and Ndiaye, S. (2017). S-fpg: A parallel version of fp-growth algorithm under apache spark.

Glymour, Clark, Richard, Peter, and Kevin (1987). Discovering causal structure: Artificial intelligence, philosophy of science, and statistical modeling. *Journal of Educational Statistics. 14. 10.2307/1164612*.

Han, J. and Kamber, M. (2006). Data mining: Concepts and techniques. *The Morgan Kaufmann Series in Data Management Systems, 2nd edition.*

Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation.

Hasan, M. Z., Chisty, K. A., and Ayshik., N.-E.-Z. (2012). Research challenges in web data mining. *International Journal of Computer Science and Telecommunications*, 3(7):37–54.

IBM QUEST, I. i. s. (1996).

Inamdar, S. A. and Shinde (2010). An agent based intelligent search engine system for web mining. *International Journal on Computer Science and Engineering*, 2(3).

Jain, K., A., Dubes, and C., R. (1988). Algorithms for clustering data.

Kaliyaperumal, D. and Dorairangaswamy, M. (2016). Web usage mining: Improve the user navigation pattern using fp-growth algorithm.

Kleinberg, J. (1998). Authoritative sources in a hyperlinked environment. *Proc. 9th Ann. ACM–SIAM Symp. Discrete Algorithms, ACM Press*, pages 668–677.

Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., and Verkamo., A. (1994). Finding interesting rules from large sets of discovered association rules. *CIKM*.

Kosla, R. and Blockeel (2000). Web mining research: A survey. *SIG KDD Explorations*, 2:1–15.

Kumar and S.Nandan (2015). World towards advanced web mining: A review. *American Journal of Systems and Software 3.2*, pages 44–61.

Kumar, B. and K.V.Rukman (2010). Implementation of web usage mining using apriori and fp growth algorithms.

Kumar, B. and K.V.Rukmani (2010). Implementation of web usage mining using apriori and fp growth algorithms.

Li, N., Zeng, L., He, Q., and Shi, Z. (2012). Parallel implementation of apriori algorithm based on mapreduce.

Moens, S., Aksehirli, E., and Goethals, B. (2013). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *IEEE International Conference on Big Data*.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Previous number = SIDL-WP-1999-0120.

Pal, S. K., V.Talwar, and Mitra, P. (2002). Web mining in soft computing framework: Relevance, state of the art and future directions. *IEEE transactions on neural network*, 13(5):1163–1177.

Poovammal, E. and Cigith, P. (2011). Mining web path traversals based on generation of fp tree with utility.

R.Kousalya, K.Suguna, and Saravanan, V. (2013). Improving the efficiency of web usage mining using k-apriori and fp-growth algorithm. *International Journal of Scientific Engineering Research Volume 4, Issue3, March-2013*.

Salleb, A. and Vrain, C. (2000). An application of association knowledge discovery and data mining. *PKDD, LNAI 1910*, pages 613–618.

Shahbazi, N., Soltani, R., Gryz, J., and An, A. (2016). Building fp-tree on the fly: Single-pass frequent itemset mining. pages 387–400.

Shaikh, A. (2015). Web usage mining using apriori and fp growth algorithm.

Sharma, P., Khan, S., Singh, S., and Tiwari, P. (2015). An analysis on web usage mining for internet users. *(IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (5), 4765-4767 www.ijcsit.com*.

Shenoy, P., Haritsa, J., Sudarshan, S., Bhalotia, G., Bawa, M., and Shah, D. (2000). Viper: A vertical approach to mining association rules. *SIGMOD 2000*.

Shi, X., Chen, S., and Yang, H. (2017). Dfps: Distributed fp-growth algorithm based on spark. *IEEE*.

Silverman, B. W. (1986). Density estimation for statistics and data analysis.

Silverstein, C., Brin, S., Motwani, R., and Ullman, J. (1998). Scalable techniques for mining causal structures. *VLDB*.

Singh, A. K., Kumar, A., and Maurya, A. K. (2014). An empirical analysis and comparison of apriori and fp- growth algorithm for frequent pattern mining. *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference*.

Sisodia, D. S., Khandal, V., and Singhal, R. (2016). Fast prediction of web user browsing behaviours using most interesting patterns.

Srivastava, J., Cooley, R., Deshpande, M., and P.N.Tan (2000a). Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):12–23.

Srivastava, J., Cooley, R., Deshpande, M., and Tan, P. (2000b). Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):12–23.

Stummea, G., Hothoa, A., and B.Berendtb. (2006). Semantic web mining: State of the art and future directions. *SIG KDD Explorations*, 4(2):124–143.

Titterington, D. M., Smith, A. F. M., and Makov, U. E. (1985). Statistical analysis of finite mixture distributions.

T.Srivastava, P.Desikan, and V.Kumar (2005). Web mining – concepts, applications and research directions. *IEEE Transactions on Computers*.

Wang, Y., Li, H., Zhang, D., Zhang, M., and Chang, E. (2008). Pfp: Parallel fp-growth for query recommendation. *ACM 2008*.

Weiss, S. M. and Kulikowski, C. A. (1991). Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning, and expert systems. *M. Kaufmann Publishers*.

Xia, D., Zhou1, Y., Rong, Z., and Zhang, Z. (2013). Ipfp: An improved parallel fp-growth algorithm for frequent itemsets mining. *Proceedings 59th ISI World Statistics Congress, 25-30 August 2013, Hong Kong (Session CPS026)*.

YANG, Q., Fei-Yang, ZHU, X., and JIANG, C.-G. (2016). Improved balanced parallel fp-growth with mapreduce. *Joint International Conference on Artificial Intelligence and Computer Engineering (AICE 2016) and International Conference on Network and Communication Security (NCS 2016).*

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. *HotCloud 2010.*

Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. . (1997). New algorithms for fast discovery of association rules.

Zembowicz, R. and Zytkow, J. M. (1996). Advances in knowledge discovery and data mining. pages 328–349.

Zhang, F., Xiao, Y., and Long, Y. (2017). Research and improvement of parallelization of fp growth algorithm based on spark.

Zhou, L., Zhong, Z., Chang, J., Li, J., Huang, J. Z., and Feng, S. (2010a). Balanced parallel fp-growth with mapreduce. *IEEE.*

Zhou, L., Zhong, Z., Chang, J., Li, J., Huang, J. Z., and Feng, S. (2010b). Balanced parallel fp-growth with mapreduce. *Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference*.