People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Laghouat

Departement of computer science


Thesis Submitted in Partial Fulfillment of the requirements for the Master degree in Computer Science

***Submitted by:*** Chaima Kamri

Theme

---

# Big data frequent itemset mining

---

***Jury members:***

| | | |
|---|---|---|
| Mr Laradj Chellama | MA(A) | President |
| Mr Mohamed El Habib Maicha | MA(A) | Examiner |
| Mr Benameur Ziani | MC(A) | Advisor |

2021/2022

# شكر وعرفان

الحمد لله وكل الشكر له فسبحانه الذي خلق ثم هدى

اللهم لك الحمد على ما قدرت ورزقت وأعنت وسترت

ثم الشكر لمن قرن بي عباده سبحانه بالإحسان إليهما: أمي وأبي

شكرا على كل السعادة التي أحظى بها معكما

ثم الشكر لمن لا تتم السعادة إلا معهم: فاطمة، مصعب، عبد الجليل، يمينة ومعمر إخوتي

شكرا على كل المواقف التي لا تخلو من جمالكم

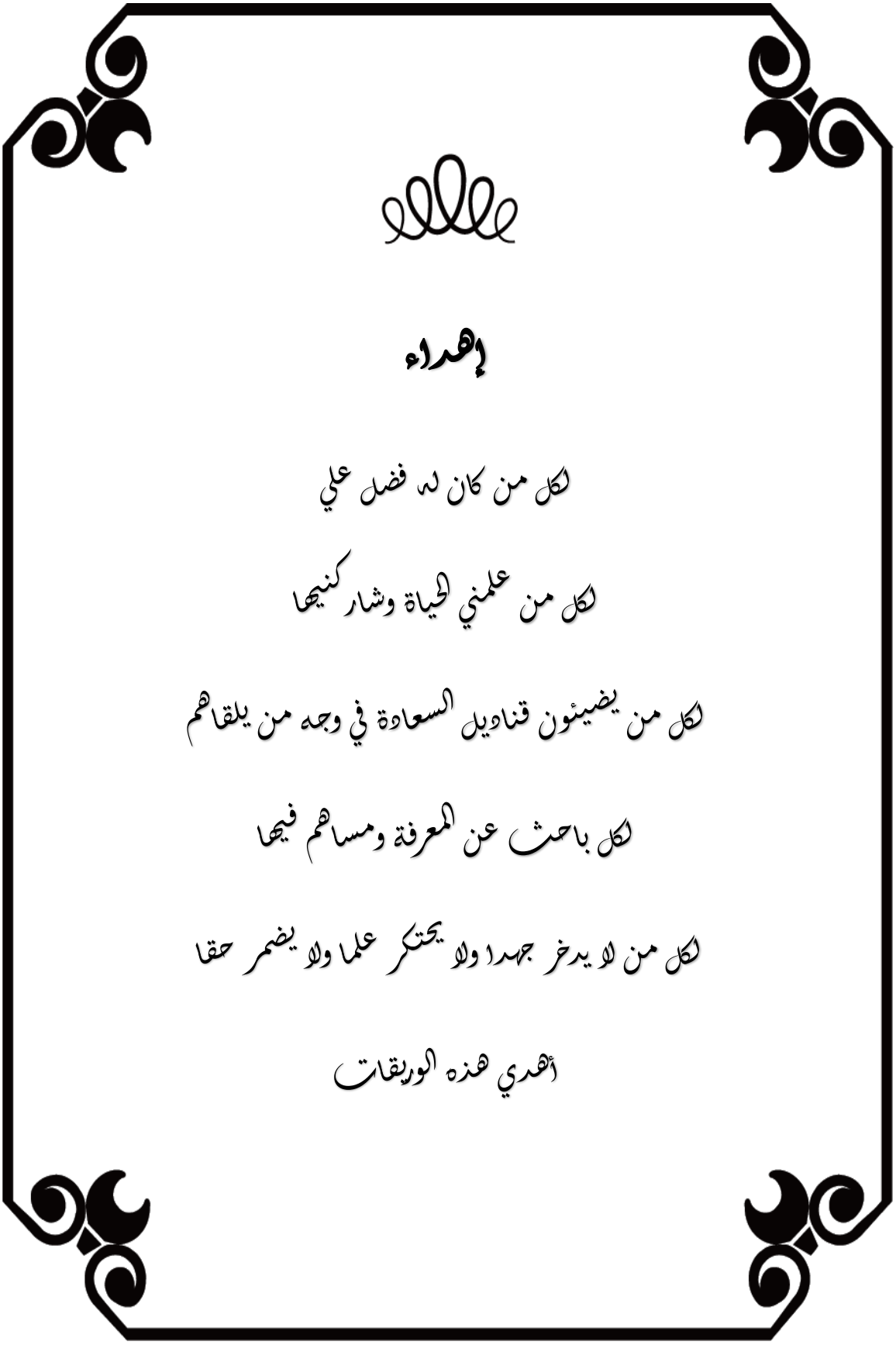
ثم شكرا لمربي الثاني وأستاذي الأول: كبير محمد

شكرا لجميع أساتذتي: على العلم الذي حملتمونيه وعلى القيم التي غرستموها فيّ

شكرا لكل الأسماء التي ما زالت في ذاكرتي

ثم شكر خاص لمن تفضل بقبول الإشراف عليّ وتوجيهي في هذا العمل

الأستاذ بن عمر زياني



# إهداء

لكل من كان له فضل علي

لكل من علمني الحياة وشاركنيها

لكل من يضيئون قناديل السعادة في وجه من يلقاهم

لكل باحث عن المعرفة ومساهم فيها

لكل من لا يدخر جهدا ولا يحتكر علما ولا يضمر حقا

أهدي هذه الوريقات

## Abstract

Big data takes its fame from analyzing the data gathered from various sources. This process is called Big data analytics that includes multiple techniques, each one differs from the others by the data used, and the results earned. But they share the same purpose, which is the improvement of decision-making. In this manuscript, we aim to present one of the famous analytic techniques, frequent itemset mining. That seeks to find the frequent patterns in a transactional database and decipher the association rules among those patterns, therefore, using this knowledge to make decisions based on those patterns. Our purpose is to introduce the big data process to mine the frequent itemset using Spark's machine-learning library, and Python.



***Keywords***— Big data, Data mining, Frequent itemset mining, Apriori, Fp-growth, Eclat, Hadoop, Spark MLlib.



## Résumé

Le big data prend sa renommée de l'analyse des données capturées de différentes sources. ce processus d'analyse est nommé par le big data analytics qui comprend beaucoup de techniques qui sont différents en terme des entrées et des sorties mais elles partagent le même but qui est l'amélioration de la prise des décisions. Dans cette manuscrit on s'intéresse de la recherche des motifs fréquents, une des technique d'analyse qui sert à trouver des items fréquents dans une base transactionnelle et après de détecter les règles d'association entre ces items et donc de construire une connaissance qui peut être significative dans la prise de décision à propos de ces items. Notre objectif est de présenter le processus utilisé dans le big data pour fouiller les motifs fréquents à l'aide de la libraire de l'apprentissage automatique Spark MLlib et Python.



***Mots clés***— Big data, La fouille de données, La recherche des motifs fréquents, Apriori, Fp-growth, Eclat, Hadoop, Spark MLlib.

# Table of contents

# List of Tables

# List of Figures

# List of Acronyms

**KDD**: Knowledge discovery from data.
**ID**: Identifier.
**FIM**: Frequent itemset mining.
**sup**: support.
**min-sup**: minimum support.
**nbr**: number.
**FP**: Frequent pattern.
**conf**: Confidence.
**tid**: transaction identifier.
**SQL**: Structured Query Language.
**JSON**: JavaScript Object Notation.
**XML**: eXtensible Markup Language.
**HDFS**: Hadoop Distributed File System.
**MB**: MegaByte.
**GB**: GegaByte.
**CPU**: Central Processing Unit.
**RAM**: Random Access Memory.
**GPU**: Graphics Processing Unit.
**JVM**: Java Virtual Machine.
**YARN**: Yet Another Resource Negotiator.
**NoSQL**: Not Only SQL.
**HQL**: Hive Query Language.
**RDD**: Resilient Distributed Datasets.
**MLlib**: Machine learning library.
**PFP**: Parallel fp-growth.
**DAG**: directed acyclic graph.
**API**: Application programming interface.

# Introduction

We all have a smart device to connect to the world wide web, to surf, to chat, or to share our routines. With this durable faithfulness; we send/receive a lot of data of different types; it can be audio or a video call, an email, or just your new account's information. Those different and enormous data are called *Big data*.

In reality, we used to do this kind of daily operations, due to the intelligence behind the systems that we deal with, which returns to big data analytics.

Big data analytics is a process that takes its input from big data sources such as social networks, online transactions...etc. And apply to it a set of algorithms, leading to extract a hidden knowledge depending on the data used, and the applied algorithm. That knowledge may be helpful for decision-making; like improving the user experience that may lead to best organizing the shop's racks. This organization of racks may be decided after a knowledge about which products are more purchased than others. This type of knowledge is treated by one of the data mining tasks, which is frequent itemset mining.

Frequent itemset mining is performed on a transactional dataset, and aims to extract a set of items that are more appeared than the others depending on a threshold fixed by the user. This technique may apply differently to big data environments, due to big data particularity. This particularity is caused by the large volume of data, that can't be treated by traditional tools; therefore it needs innovative tools which can process the nonpredictable amounts of data in a reasonable time. Actually, all such tasks (in big data environments) invoke parallelization as the first solution. One of the solutions suggested to treat the problem of frequent itemset mining in big data is parallel fp-growth[1]. It was adopted by the popular big data processing engine; Spark in its machine learning library.

In this work, we answer to big data frequent pattern mining by presenting the Spark offered tool ; to facilitate the dealing with the problem. Thus we put the reader in front of the implementation of parallel fp-growth in Spark's machine learning library MLlib using Python, as a programming language.

To present our objective in a smoothy form to the reader, we thought to organize our document into three chapters:

- Chapter one: Data mining; a road to the knowledge. Where we will introduce the data mining concept, and then some of the algorithms used to mine the frequent itemsets, will be presented.
- Chapter two: Big data : data, process & consequences. Here we will present big data, its characteristics, and its technologies.
- Chapter three: Experimentation and discussion. In this chapter, we will implement the Spark MLlib frequent itemset mining solution in Python, and measure its performance as a big data solution to the problem.

At the end of these chapters, we conclude with a conclusion and perspectives suggested for improving our work. .

Chapter 1

# Data mining: a road to the knowledge

## 1.1 Introduction

As humans, we need new knowledge to build our perceptions about things around us. This need is crucial to guarantee the life cycle development. The same need; we find in the numeric side, depicted by the data mining field, where the knowledge will be deducted from gathered data, as the data is the unique supplier in the numeric world. Data mining is based on a set of algorithms classified as descriptives or predictives, depending on the target behind the task of knowledge mining.

The aim of this chapter is: to introduce data mining as a step of knowledge discovery from data, and then present the targeted task of this thesis; which is the frequent itemset mining. Thus some of the algorithms dealing with the problem will be described.

## 1.2 Data mining concept

People belonging to the data science field, often classify data mining as the whole process of Knowlledge Discovery from Data (KDD). Because the new knowledge jumps into the real world after applying the data mining algorithms. But in reality, data mining is just a step in the KDD process, where every step has its importance.[2]

### 1.2.1 The steps of knowledge discovery from data[2]

1. **Data cleaning:** data comes from various sources that may be noise, redundant, and/or inconsistent; so to earn a valuable analysis we must pass with this step.
2. **Data integration:** once we clean the captured data, the time is came to integrate these various sets into one repository.
3. **Data selection and transformation:** analysis tasks are different in terms of input data, as they are different in terms of results. So that is why we need to pass by this step; to select which data to be used, or to transform it into another one using aggregate operations.
4. **Data mining:** here we apply a set of algorithms (based on the target to be reached) in order to discover hidden patterns. Where pattern means the relationship between the processed data; inferred from the task.

5. **Pattern evaluation:** here we decide if the recognized pattern will be interesting or not, using different measures.
6. **Knowledge presentation:** here we present the knowledge (pattern) in a comprehensible format.

In other words; data mining is an interdisciplinary subject that invokes statistics, mathematics, and computer science; aims to find a small significant set from raw data. This set represents a knowledge that can be used in decision making.

### 1.2.2 The kinds of data that can be mined

"data mining can be applied to any kind of data as long as the data are meaningful for a target application"[2]

#### Data types :

- **Relational data:** refers to the database and data warehouse data.
- **Transactional data:** the previous type stands on a respective schema; which is a set of rows including columns values. But in this type, we have a different schema that involves for each transaction ID the list of invoked data; called items, like customer's purchase, or user's clicks on a web page. . . etc.
- **Data streams:** captured from real-time streams like (video surveillance and sensor data).
- **Time-related or sequence data:** like historical records.
- **Spatial data:** like maps; data coming from satellites.
- **Hypertext and multimedia data**.
- **Graph and networked data:** social networks data.

### 1.2.3 Data mining tasks

Data mining tasks divide in two families: predictive tasks and descriptive tasks.

- **Predictive tasks:** aims to make predictions on the data depending on what we inferred from the recognized patterns.
- **Descriptive tasks:** "...characterize properties of the data in a target dataset..."[2]

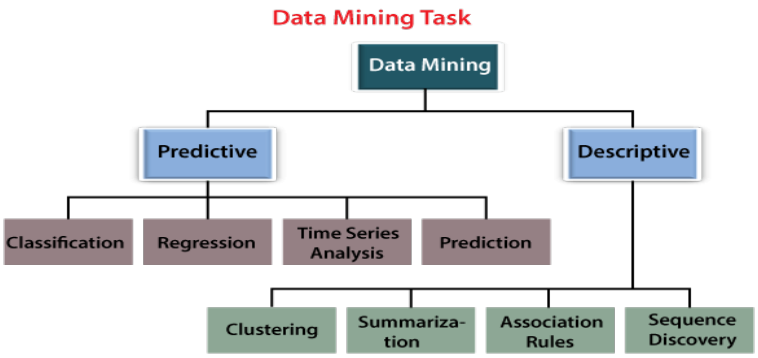



Figure 1.1: Data mining tasks [3]

Depending on Figure 1.1; there is a classification of tasks. But in the real practical environment, the nature of the task will be defined based on the task's objective which is the answer of : why do you want to analyze your data ? There are no conditions on how to deal with your data ? or what you decide from the algorithm's results ?

In our thesis, we focus on frequent itemsets (patterns) mining; which is the first step in association rules mining, therefore we won't discuss the other tasks types.

## 1.3 Frequent itemsets mining (FIM)

Often called market basket analysis. The technique refers to finding a set of items that are frequently occurred together in a dataset, for example, milk and bread are frequently bought together.

The technique stands on the traditional algorithmic problem of occurrence calculation. Hence in order to decide if an item is frequent or not, we must calculate its occurrence in a transactional dataset. The decision of which items are frequent and which are not, led us to fix a threshold to compare it with the item's occurrence value. This value is called minimum support which is in general cases a percentage defined by the user.

### 1.3.1 Problem's vocabulary

- **Item:** it is the subject of the study meaning; if i want to know what products are frequently bought together ? The item here refers to a product. If you want to know which pages are more visited ? Items here refer to pages and so on.
- **Support:** item's support is: how much the item occurs in the dataset.
  With mathematical language;

$$sup(item) = \frac{\text{nbr of transactions involving this item}}{\text{the total nbr of transactions}}$$

- **Minimum support:** a percentage fixed by the user.

- **Itemset:** is a set of items. Where its support refers to the percentage of the transactions, that involve the itemset members together.
- **K-itemset:** a set of K items.
- **Frequent itemset:** an itemset is frequent; if and only if its support is greater or equal to the minimum support.
- **Closed itemset:** a frequent itemset is closed; if all its supersets have a lower support value than its one.
- **Maximal itemset:** a frequent itemset is maximal; if all its supersets are not frequent.

### 1.3.2 Problem solutions

To treat the problem of frequent itemset mining, many solutions are suggested, each one of them has its methodology to mine the frequent itemset. We were interested in three algorithms Apriori, Fp-growth and Eclat.

#### Apriori

It is the naive algorithm to find the frequent itemsets; by scanning the database iteratively until found all the frequent itemsets. Apriori depends on an intelligent property in order to reduce the search space. This property (called antimonotonicity or apriori property ) says that all non-empty subsets of a frequent itemset must also be frequent.

**Apriori steps [2]**

1. Find the frequent 1-itemsets by scanning the database and comparing each item's support with the minimum support.
2. Generate the 2-itemset candidates by combining the frequent 1-itemsets for example; let us say that { milk, bread, butter } are frequent 1-itemset. the 2-itemset candidates will be {milk, bread},{milk, butter},and {bread, butter}
3. Build the frequent 2-itemsets by comparing the 2-candidates support with the minimum support.
4. And so on, until we cannot build any frequent itemset.

**Note:** in K-itemset candidate generation where K$>$ 2; two (K-1)-itemsets can be joined if and only if, they share the same K-2 itemset (prefix).
Example: let's take the previous example and say that all the candidates are frequent:
{milk, bread}, {milk, butter}, {bread, butter} $\Longrightarrow$ the only join that we are able to do is between {milk, bread} & {milk, butter} and the result will be {milk, bread, butter}.

The condition of prefix sharing, enables the reduction of the generated candidates.

**Apriori's algorithm [2]:**
**Input**:

- D, a database of transaction.
- min-sup

**Output**: L, frequent itemsets in D.
**Method:**

(1) $L_1 = find\_frequent\_1 - itemsets(D)$;
(2) **for**$(k = 2; L_{k-1} \neq \phi; k++)\{$
(3) $C_k = apriori\_gen(L_{k-1})$;
(4) **for each** *transaction* $t \in D\{$ //scan D for counts
(5) $C_t = subset(C_k, t)$;//get the subsets of t that are candidates
(6) **for each** *candidate* $c \in C_t$
(7) $c.count++$;
(8) $\}$
(9) $L_k = \{c \in C_k | c.count \geq min - sup\}$
(10) $\}$
(11) **return** $L = \cup_k L_k$;

**procedure apriori_gen($L_{k-1}$:frequent (k-1)-itemsets)**

(1) **for each** *itemset* $l_1 \in L_{k-1}$
(2) **for each** *itemset* $l_2 \in L_{k-1}$
(3) **if**$(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge ... \wedge (l_1[k-2] = l_2[k-2])$
$\wedge (l_1[k-1] = l_2[k-1])$ **then**$\{$
(4) $c = l_1$ *join* $l_2$; //join step: generate candidates
(5) **if** $has\_infrequent\_subset(c, L_{k-1})$ **then**
(6) **delete** $c$ //prune step: remove unfruitful candidate
(7) **else add c to** $C_k$;
(8) $\}$
(9) **return** $C_k$

**procedure has_infrequent_subset**$(c : candidate\ \ k-itemset;$
$L_{k-1} : frequent(k-1) - itemsets)$;//use prioir knowledge
(1) **for each**$(k-1) - subset\ \ s$ **of** $c$
(2) **if** $s \notin L_{k-1}$ **then**
(3) **return** $TRUE$;
(4) **return** $FALSE$;

**Example [2]**
We will use this sample dataset for all the algorithms that we speak about.The minimum support is equal to 2. Which is about 22% from the dataset.

| TID | list of items_IDs |
|---|---|
| T100 | I1, I2, I5 |
| T200 | I2, I4 |
| T300 | I2, I3 |
| T400 | I1, I2, I4 |
| T500 | I1, I3 |
| T600 | I2, I3 |
| T700 | I1, I3 |
| T800 | I1, I2, I3, I5 |
| T900 | I1, I2, I3 |

Table 1.1: Sample dataset [2]

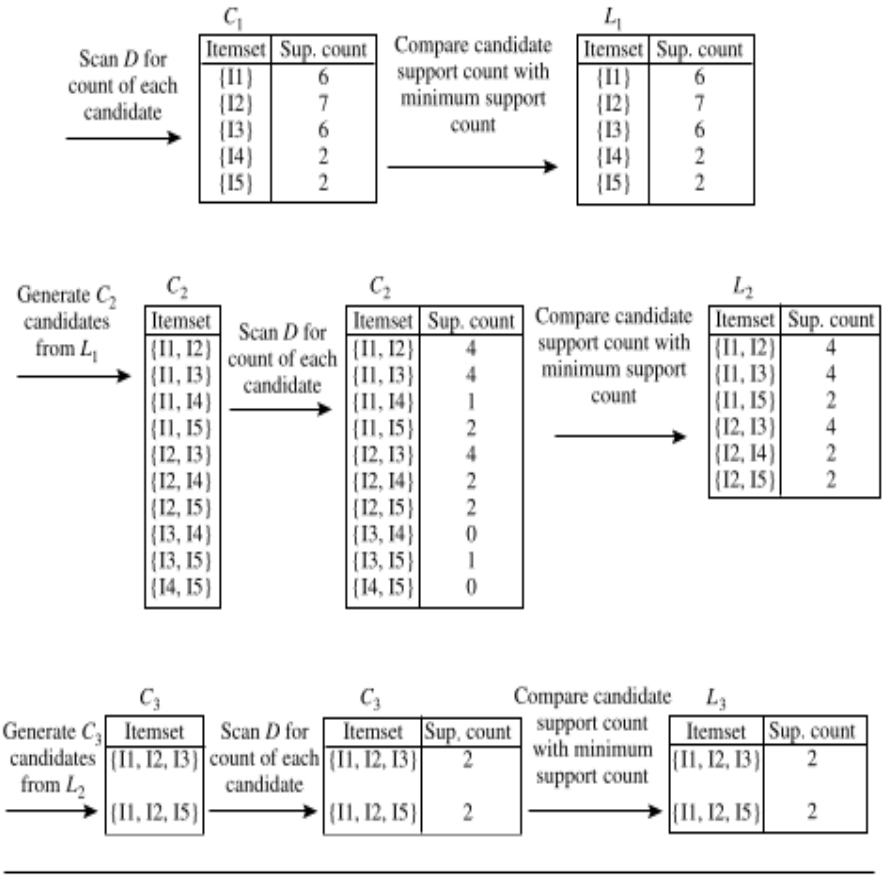



Figure 1.2: Frequent itemset mining using Apriori on dataset presented in Table 1.1 [2]

### Frequent pattern growth (FP-growth)

The two weaknesses of apriori are :

- Scanning iteratively the whole database.
- The cost of candidates generation.

In order to not scan the whole database iteratively, fp-growth supposes another data structure called fp-tree. Which enables to compress the whole database into a tree; thus frequent items can be easily mined.

#### Fp-Growth Steps [2]

1. Extract the frequent 1-itemsets as Apriori.
2. Sort this set in descending order depending on the item's support.
3. Construct the fp-tree: the method is clearly described in the algorithm.
   **Note:** the tree's nodes are merely the frequent 1-itemsets.
4. Extract all the frequent itemsets.
   - Start mining the tree from its leaves.
   - Extract for each item leaf the conditional pattern base, which is about the possible branches to the item's label.
   - Construct the conditional fp-tree, which is about the prefix items led to this leaf. Where their supports $\geq$ min-sup, meaning the times of pass with this path nodes to the leaf $\geq$ min-sup .
   - Finally, generate frequent patterns by combining this item leaf with the frequent patterns generated from the conditional fp-tree.

**Fp Growth's algorithm [2]:**
**Input**:

- D, a database of transaction.
- min-sup

**Output**: the complete set of frequent patterns.
**Method:**

1. The Fp-tree is constructed in the following steps:
   (a) Scan the transaction database $D$ once. Collect $F$, the set of 1-frequent items and their support counts.Sort $F$ in support count decending order as $L$, the list of frequent items.
   (b) Create the root of an FP-tree, and label it as "null". For each transaction *Trans* in $D$ do the following:
   Select and sort the frequent items in *Trans* according to the order of $L$. Let the sorted frquent item list in *Trans* be [p|P], where p is the first element and $P$ is the remaining list. Call **insert_tree([p|P],T)**, which is performed as follows.If $T$ has a child $N$ such that *N.item-name = p.item-name*, then increment $N$'s count by 1; else create a new node $N$, and let its count be 1, its parent link be linked to $T$, and its node-link to the nodes with the same item-name via the node-link stucture. If $P$ is nonempty, call insert_tree($P$,$N$) recursively.
2. The FP-tree is mined by calling **FP-growth**(FP-tree, null) which is implemented as follows

**procedure FP-growth**($Tree, \alpha$)
(1) **if** *Tree contains a single path P* **then**
(2) **for each** *combination* (*denoted as* $\beta$) *of the nodes in the path P*
(3) *generate pattern* $\beta \cup \alpha$ *with support_count = minimum support count of nodes in* $\beta$;
(4) **else for each** $\alpha_i$ *in the header of Tree*{
(5) *generate pattern* $\beta = \alpha_i \cup \alpha$ *with support_count* $= \alpha_i$*.support_count*;
(6) *construct* $\beta$*'s conditional pattern base and then* $\beta$*'s conditional FP_tree* $Tree_\beta$
(7) **if** $Tree_\beta \neq \phi$ **then**
(8) *call* **FP_growth**($Tree_\beta, \beta$);}

**Example [2]**

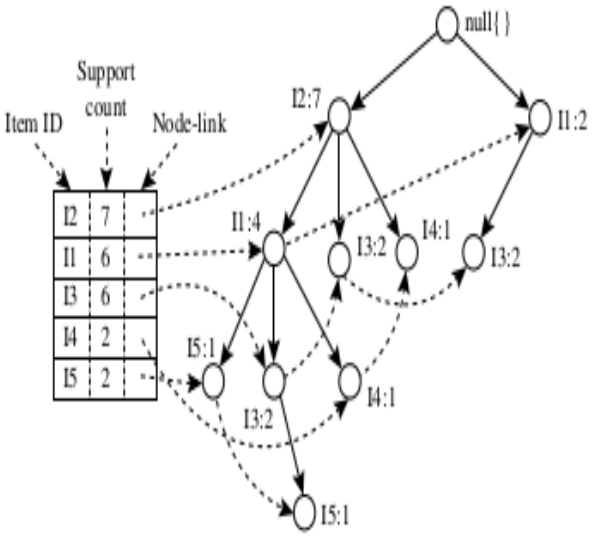



Figure 1.3: Fp-tree of the sample dataset 1.1 [2]

| **Item** | **Conditional Pattern Base** | **Conditional FP-tree** | **Frequent Patterns Generated** |
|---|---|---|---|
| I5 | {{I2,I1:1},{I2,I1,I3:1}} | {I2:2,I1:2} | {I2,I5:2},{I1,I5:2},{I2,I1,I5:2} |
| I4 | {{I2,I1:1},{I2:1}} | {I2:2} | {I2,I4:2} |
| I3 | {{I2,I1:2},{I2:2},{I1:2}} | {I2:4,I1:2},{I1:2} | {I2,I3:4},{I1,I3:4},{I2,I1,I3:2} |
| I1 | {{I2:4}} | {I2:4} | {I2,I1:4} |

Table 1.2: Frequent itemset mining using fp-growth on the sample dataset 1.1

### Eclat

Eclat in the opposite of Fp-growth and Apriori, uses vertical database in order to reduce the scan times, to facilitate the support calculation, and the candidate generation.

#### Vertical database vs horizontal database [2]

- Horizontal database: rows represent the transactions, for each transaction we find the list of items invoked by this transaction.
- Vertical database: rows are items, and for each item, we find the transactions IDs list which invoked this item.

#### Eclat steps [4]

1. Build the vertical database from the original dataset.
2. Delete the non-frequent single itemsets. Where the item's support can be deducted from the length of the transaction set of this item.
3. Generate the 2-itemset candidates by intersecting the tid-set for every pair of frequent 1-itemset.
4. Keep the frequent 2-itemset and repeat the process until no more frequent itemset can be mined.

#### Eclat's algorithm

**Input:** dataset, min-sup
**Output:** $L$: list of all frequent itemsets

```
1. VDB = build_vertical_database(dataset)
2. k = 1
3. k_frequent_VDB = [ ] // Vertical database from k frequent itemset
4. for each row in VDB: // mine 1-frequent itemset
5.    if len(row.TransactionList)/len(dataset)≥ min-sup :
6.       L.append(row.item)
7.       k_frequent_VDB.append(row)
8. k+1_candidates = [ ]:
9. while(len(k_frequent_VDB)>0):
10.   for each row in k_frequent_VDB:
11.      for each element in row.successors: // candidates generation
12.         k+1_candidates.append((row.item∪element.item), row.TransactionList∪element.TransactionList)
13.   k+1_frequent_VDB = [ ]
14.   for each row in k+1_candidates:
15.      if len(row.TransactionList)/len(dataset)≥ min-sup :
16.         L.append(row.item)
17.         k+1_frequent_VDB.append(row)
18.   k++
```

**Example [2]**

| itemset | TID_set |
|---|---|
| I1 | {T100, T400, T500, T700, T800, T900} |
| I2 | {T100, T200, T300, T400, T600, T800, T900} |
| I3 | {T300, T500, T600, T700, T800, T900} |
| I4 | {T200, T400} |
| I5 | {T100, T800} |

Table 1.3: The Vertical Database Format of the Transaction dataset of Table1.1

| itemset | TID_set |
|---|---|
| {I1,I2} | {T100,T400,T800,T900} |
| {I1,I3} | {T500,T700,T800,T900} |
| {I1,I5} | {T100,T800} |
| {I2,I3} | {T300,T600,T800,T900} |
| {I2,I4} | {T200,T400} |
| {I2,I5} | {T100,T800} |

Table 1.4: Frequent 2-Itemsets with Eclat

| itemset | TID_set |
|---|---|
| {I1,I2,I3} | {T800,T900} |
| {I1,I2,I5} | {T100,T800} |

Table 1.5: Frequent 3-Itemsets with Eclat

### 1.3.3 Comparison among Apriori, Fp-growth, and Eclat

We saw how different is the methodology of each algorithm. Now we put the different points to compare among them.

| **Evaluation Criteria** | **Apriori** | **FP Growth** | **Eclat** |
|---|---|---|---|
| **1. Techniques** | Breadth first search | Divide and Conquer | Depth first search and intersection of transaction id |
| **2. Database Scan** | Database is scanned each time a candidate itemset is generated | Database is scanned two times only | Database is scanned few times |
| **3. Advantages** | - Easy to implement.<br>- Use Large itemset property. | Database scanned two times only | No need to scan database each time. |

| | | | |
|---|---|---|---|
| **4. Disadvantages** | - Require large memory space.<br>- Too many candidate itemset. | FP tree is expensive to build, consumes more memory. | It requires virtual memory to perform the transaction. |
| **5. Data format** | Horizontal | Horizontal | Vertical |
| **6. Storage format** | Array | Tree(FP tree) | Array |
| **7. Time** | More execution time | Less time than Apriori | Less time than Apriori |

Table 1.6: Comparison among Apriori, Fp-growth, and Eclat [5]

### 1.3.4 Association rules mining

The aim of this work turning about frequent itemset mining. However, we will take a peek into association rules mining.

Once we finish the FIM, we will be interesting to decipher the relations among the frequent K-itemset members, where K $\geq$ 2. Which items led to invoke the others in implication format(A$\longrightarrow$ B, A & B are itemset, A $\neq \emptyset$, B $\neq \emptyset$ and A$\cap$B = $\emptyset$).
Here we use another measure called confidence; to decide the strong association rules by comparing the rule's confidence with the min-conf's value fixed by the user.
With natural language, confidence refers to how much it's true that A items invoke B items.

$$sup(A \longrightarrow B) = \frac{\text{nbr of transactions involving both A \& B items}}{\text{the total nbr of transactions}}$$

$$conf(A \longrightarrow B) = \frac{sup(A \longrightarrow B)}{sup(A)} = \frac{\text{nbr of transactions involving both A \& B items}}{\text{nbr of transactions that inolve only A items}}$$

Knowing that the problem inputs are: transactional database, minimum support, and minimum confidence.

## 1.4 Conclusion

In this chapter, we have discovered data mining as a step in the KDD process, we have defined the frequent itemset mining technique, and its solution using Apriori, Fp-growth, and Eclat. We saw also that data mining tasks take their input from several data sources. Most of those data sources are present in big data, which will be discussed in the next chapter.

Chapter 2

# Big data : data, process & consequences.

## 2.1 Introduction

Around the world, there are millions of data transfers, as there were more than 181,331,340 Emails sent, and 5,500,560 Google searches done,...etc[1]. With this kind of operations, the concept of big data had seen the light.
This chapter comes with the aim to unveils some concepts related to big data

## 2.2 Definition

Here are some definitions of big data:

- Gartner; which is an American technological research and consulting corporation, was one of the firsts who define big data as:" Big data is high-volume, high-velocity and or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation. " [6]
- IBM also has another definition by keeping the 3V's: "Big data is a term applied to data sets whose size or type is beyond the ability of traditional relational databases to capture, manage and process the data with low latency. Big data has one or more of the following characteristics: high volume, high velocity or high variety." [7]
- Another source defined it as consisting of "...extensive datasets -primarily in the characteristics of volume, velocity, and/or variability- that require a scalable architecture for efficient storage, manipulation, and analysis." [8]

[1] `https://www.sas.com/en_us/insights/big-data/what-is-big-data.html`

## 2.3 Big data sources

Big data has several sources; such as online transactions, data captured from sensors, emails, relational data, data gathered from social networks, and all the data transferring in the world no matter its format (audio, video, text...etc).
These data are divided depending on their format into three families:

- Structured data: data has a respective and regular schema. In this family we found: SQL tables, Excel sheets,...etc.
- Semi-structured data: data can be depicted by a flexible schema for example, JSON files, XML schemas.
- Unstructured data: data can't be presented by a schema due to their complex nature like; videos, and audio.

## 2.4 Big Data characteristics

Usually, people in the big data field mention these characteristics by the V's of big data, that we have discovered some of them in section 2.2. In order to put the reader in front of some of those V's; we brought these clear definitions :

1. **Volume:** "refers to the production of high-volume data." [9]
2. **Velocity:** "the data production rate is unpredictable." [9]
3. **Variety:** "it relates to the diversity of data and its various formats." [9]
4. **Veracity:** "ambiguity within data is typically from noise and abnormalities within the data."[10]
5. **Value:** the anticipated benefit in business by using this technology (big data).
6. **Visualization:** "refers to how present data to the user."[9]
7. **Validity:** "correctness or accuracy of data used."[9]
8. **Versatility:** "the ability of big data to be flexible enough to be used differently for different contexts."[9]
9. **Verbosity:** "the redundancy of the information available at different sources."[9]
10. **Viscosity:** "related to Velocity; how difficult is the data to work with ? "[11]

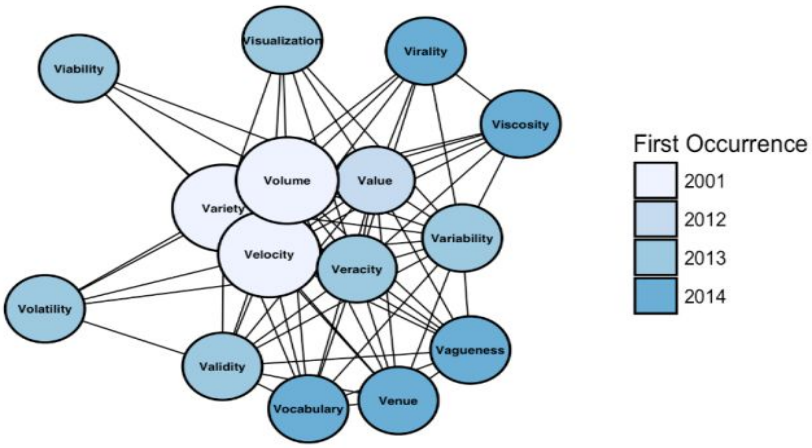



Figure 2.1: Big Data V's [11]

Depending on what we saw in section 2.4 we can easily observe that the 7V's by except (volume, velocity, variety) are results of dealing with big data, so that is why almost of field participants accept Gartner's definition as the accredited one.

## 2.5 Big data advantages

1. **Better decision-making:** it comes from the analysis of various data sources. Where you will have a look at what people say of your services from social network data, hence you will decide to improve the service quality. Or you may change for example; your website layout after studying visitors' clicks data.

2. **Best marketing strategy:** if you understand your clients (due to customer behavior analysis), you may know what they can be attracted to? So you offer your services depending on their tendencies. That will make your offering strategy attracts your clients.

3. **Sales and loyalty increasing:** you make good decisions, and you know how to offer your products; thus you will guarantee the loyalty of your clients and augment the sales ratio.

## 2.6 Big data Technologies

Back to the third definition that we saw; the authors focus on how to deal with big data by taking scalability as a solution to that problem. Hence the technologies that may serve big data processing, storage, and analysis must respect this trait.
Scalability is invoked every time when we are in front of a huge performance of a system. The idea of scalability stands on parallel processing that may resolve complex tasks in a reasonable time.

### 2.6.1 Hadoop the salient agent of big data

Hadoop at the best of our knowledge, is the first open-source platform that makes the hypothesis of process and storing big data with commodity hardware (a set of computers) possible and realizable by its tools.

### Architecture

Hadoop as a parallel architecture depends on two layers, master and slaves (workers).

- The master layer takes the responsibility of tasks scheduling, data distribution, client queries, data localization,...etc. This layer is represented -depending on Hadoop version- by name node(s), secondary name node, job tracker,and resource manager.
- The slaves (workers) layer takes the responsibility of doing the work. Here where Hadoop stores the data and achieves the tasks. It is represented by -depending on Hadoop version-: data nodes, task trackers, application managers,and containers.

Knowing that, in these layers there are two sides: hardware (name node, secondary name node,and data nodes) and software (job tracker, task trackers, ressource manager, application manager, and containers)

Earlier in its first days, Hadoop as a platform comes with two general purposes :

- **Data storage :** storing the data (no matter its format) by Hadoop Distributed File System (HDFS) . It takes the original voluminous file, divides it into blocks; each one takes a size limit of 64MB or 128MB, and then sends each block to one data node.
  To resolve the problem of wasting data due to the data node breakdown, Hadoop uses the replication technique. Where it replicates the chunks more of one time, respecting the replication factor[1] and sends these replicas to data nodes. Hence two replicas do not exist at the same where. Figure2.2

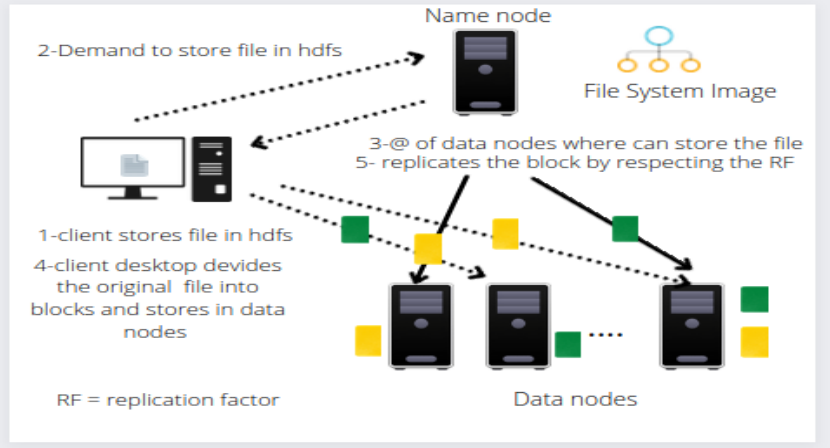



Figure 2.2: Data storage in HDFS

[1] A constant fixed by the admin of the system.

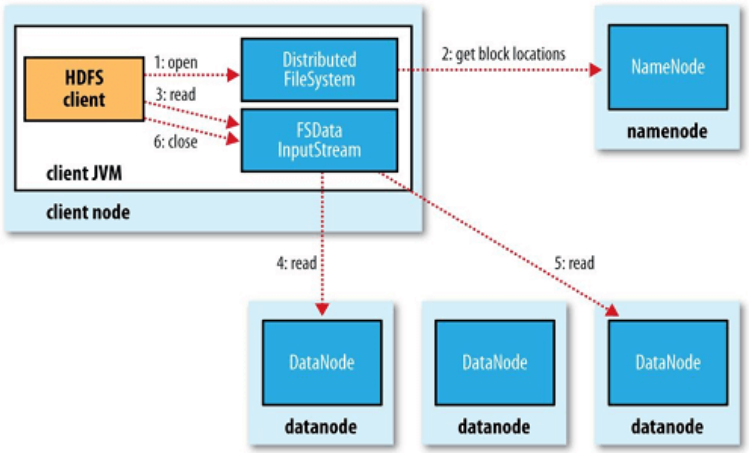



Figure 2.3: Data reading from HDFS [12]

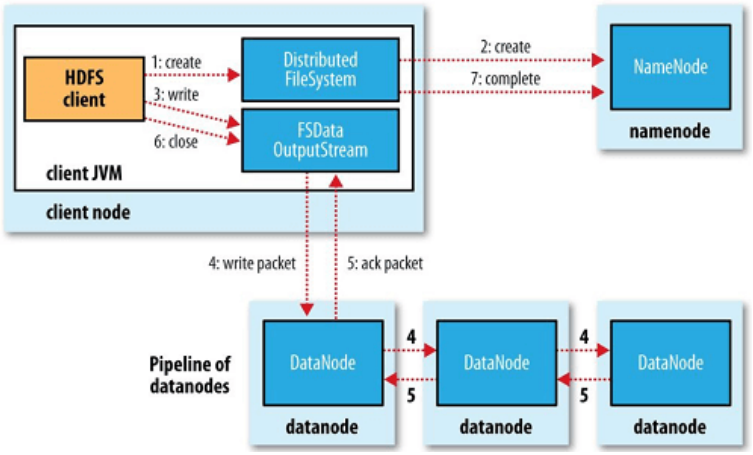



Figure 2.4: Data writing in HDFS [12]

- **Data processing**: process the data with Map-Reduce; a parallel programming paradigm composes of two stages :

  1. **Map stage:** here the process (mapper) takes the input file and transforms it into (key, value) pairs depending on how the programmer writes the task. Knowing that this stage is distributed among the responsible data nodes (which host the chunks of the file) so we will have a group of mappers.
  2. **Reduce stage:** before the output of mappers transforms to the reducer (in some times it may be a group of reducers), it must pass by another task containing two functions:
     - **Sort:** to sort the results in ascending order of the keys
     - **Shuffle:** to transfer the results of sorting into the reducer(s).

     Now the reduce stage will be ready to realize the task (the objective of the task), depending on how the programmer writes it, and products the result in (key, value) pairs.

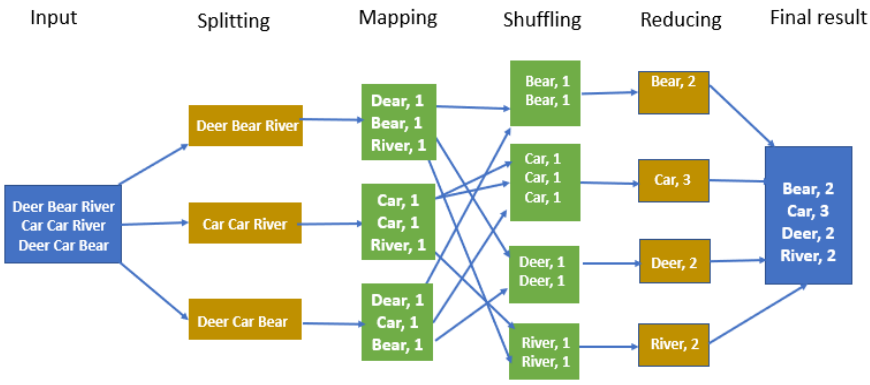



Figure 2.5: Word Counting example with map-reduce paradigm [13]

### Hadoop ecosystem

Actually, Hadoop proved its efficiency as a big data platform with two general purposes as we mentioned previously. But it had some weak sides that led to being supported by other tools; called Hadoop ecosystem.

1. **YARN or Yet Another Resource Negotiator**[14] [15] :before its coming, Hadoop organizes the task (client query) by two process types; job tracker for the name node and task tracker for the data node. The job tracker was the responsible for job(the whole operation)'s monitoring and scheduling, meaning that it takes the whole charge. In otherwise the task tracker was the responsible for the task(portion of job it can be a map function or a reduce function) achievement.
   In YARN; the job tracker replaced by the ressource manager, and the task tracker by the application manager and containers,sometimes they both are called node manager to refer the task of the data node.
   By this anatomy, YARN fairly manages resources in the cluster, where it saves the poise of charge among the data nodes, and the name node.

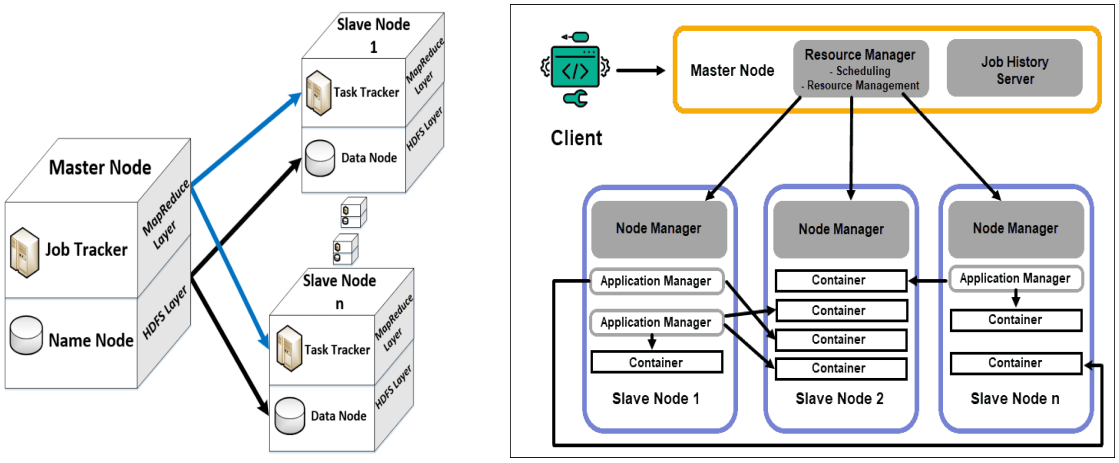



Figure 2.6: Hadoop v1 [16] VS Hadoop v2 (YARN) [17]

2. **NoSql databases** [14]: refers to not only SQL. It's a data management system without any conditions on the data's schema, therefore there is no normalization like RDBMS which conducts to get a lot of redundancy.

3. **Sqoop** [14]: aims to transfer the data from a structured environment such as relational databases, into Hadoop cluster. It can also serve the opposite operation.

4. **Zookeeper** [14]: aims for coordinating and scheduling nodes in a cluster.

5. **Pig** [14]: comes to facilitate large data sets analysis by a high programming language. That comes to escape the java map-reduce programming difficulties.

6. **Hive** [14]: is a data warehouse project, that can facilitate data querying by a SQL like which is Hive Query Language (HQL).

7. **Flume** [14]: it's a tool that enables to collect, aggregate, and transport of data from several sources and transfers it into HDFS.

8. **Kafka** [14]: it seems like a huge queue; that captures streaming data coming from various sources, and transforms them into data processing applications.

9. **Oozie** [14]: is a workflow scheduling system. It can link several tasks from several different systems.

10. **Mahout** [14]: is primarily used for creating machine learning projects. It implements recommendation, classification, and clustering algorithms.

### 2.6.2 Spark

One of the weaknesses of Hadoop treatment is that it takes too much time, because it uses disk storage to store the job results. Hence the task results were also stored on disk.(Figure 2.7)

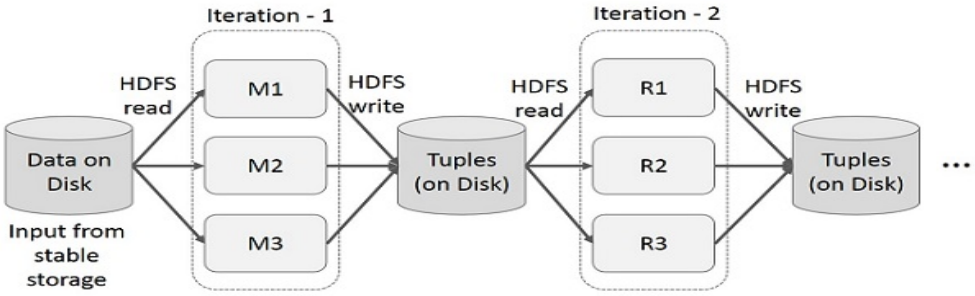



Figure 2.7: Data processing in Hadoop [18]

That technique was adopted by Hadoop to face possible failures for example; the power off, hence starting the task from its endpoint instead of repeating the whole job. The read/write from/in disk led to high latency, that may be resolved by the new alternative Spark.

Some of the field participants classify Spark as a Hadoop ecosystem belonging tool. This is true when we think of Hadoop as a storage repository for Spark [19].

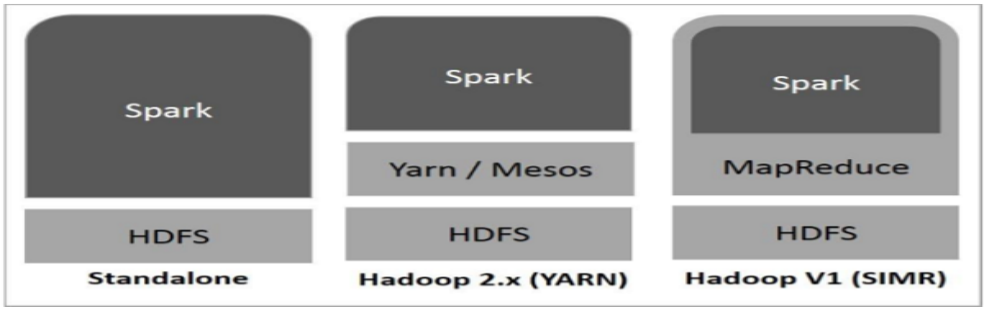



Figure 2.8: Spark use levels.[19]

Otherwise, others classify Spark as an alternative to Hadoop, because it has its own strategy of working, that may be different from Hadoop's one (Table 2.1 ). Especially in how spark deals with tasks; it uses the main memory to store results that is why it's faster than Hadoop.

| **Category of comparison** | **Hadoop** | **Spark** |
|---|---|---|
| **Performance** | Slower performance, uses disks for storage and depends on disk read and write speed. | Fast in-memory performance with reduced disk reading and writing operations. |
| **Cost** | An open-source platform, less expensive to run. Uses affordable consumer hardware. Easier to find trained Hadoop professionals. | An open-source platform, but relies on memory for computation, which considerably increases running costs. |
| **Data Processing** | Best for batch processing. Uses MapReduce to split a large dataset across a cluster for parallel analysis. | Suitable for iterative and live-stream data analysis. Works with RDDs and DAGs to run operations. |
| **Fault Tolerance** | A highly fault-tolerant system. Replicates the data across the nodes and uses them in case of an issue. | Tracks RDD block creation process, and then it can rebuild a dataset when a partition fails. Spark can also use a DAG to rebuild data across nodes. |
| **Scalability** | Easily scalable by adding nodes and disks for storage. Supports tens of thousands of nodes without a known limit. | A bit more challenging to scale because it relies on RAM for computations. Supports thousands of nodes in a cluster. |
| **Security** | Extremely secure. Supports LDAP, ACLs, Kerberos, SLAs, etc. | Not secure. By default, the security is turned off. Relies on integration with Hadoop to achieve the necessary security level. |

| | | |
|---|---|---|
| **Ease of Use and Language Support** | More difficult to use with less supported languages. Uses Java or Python for MapReduce apps. | More user friendly. Allows interactive shell mode. APIs can be written in Java, Scala, R, Python, Spark SQL. |
| **Machine Learning** | Slower than Spark. Data fragments can be too large and create bottlenecks. Mahout is the main library. | Much faster with in-memory processing. Uses MLlib for computations. |
| **Scheduling and Resource Management** | Uses external solutions. YARN is the most common option for resource management. Oozie is available for workflow scheduling. | Has built-in tools for resource allocation, scheduling, and monitoring. |

Table 2.1: Hadoop & Spark differencies [20]

Spark is classified as in-memory big data processing engine, used for both batch and stream processing. It bases on a specific immutable data structure (its content can't be changed like: constants, and tuples ) called RDD, which is the abbreviation of Resilient Distributed Datasets. It is a distributed collection of objects across cluster nodes which makes it possible to achieve iterative (multiple) and interactive (synchronous, interact with the user) tasks faster than Hadoop map-reduce.

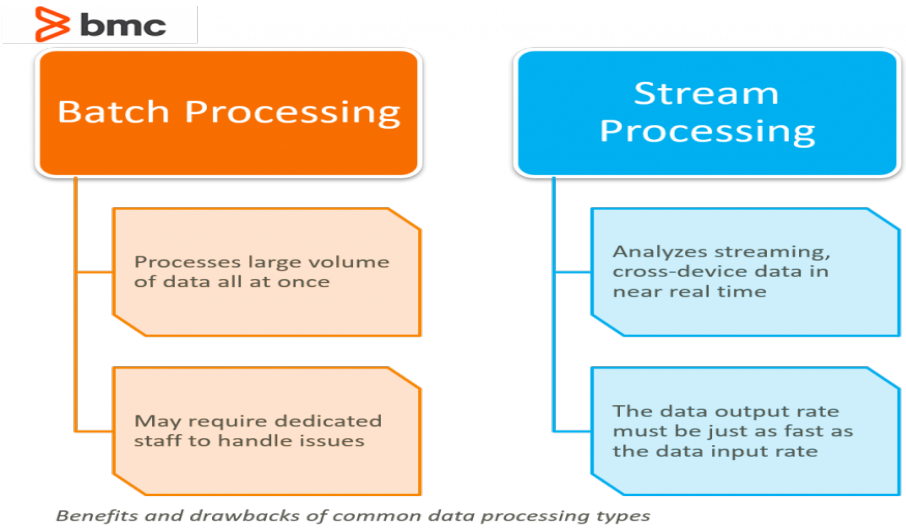



Figure 2.9: Difference between batch & stream processing.[21]

**Components:**

- **Spark core:** is the primary layer of Spark, with this component the in-memory processing happens.
- **Spark SQL:** is a top component of Spark core, that enables users to work with SQL and HQL queries. Thus structured and unstructured data can be easily treated.

- **Spark Streaming:** aims to deal with live data streams coming from different sources. Knowing that Spark uses micro-batching for real-time streaming. Figure 2.10 .

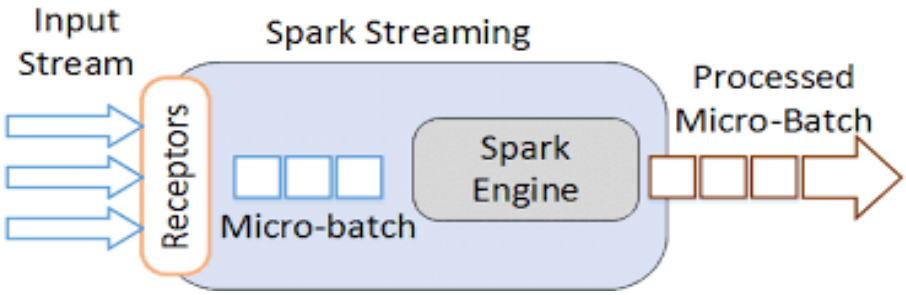



Figure 2.10: Spark streaming stands on micro-batching[21]

- **Apache Spark MLlib:** machine learning library that offers a lot of implementations of machine learning algorithms to be easily used in a scalable environment.
- **GraphX:** is a graph scalable processing engine.

### 2.6.3 Hadoop or Spark

We inferred the different use cases of Hadoop and spark from [20]:
Hadoop use cases include:

- When we have humble material especially in terms of memory size.
- Batch processing.
- Execution time isn't crucial.
- Not interactive tasks.
- Security is required.

Spark is good for:

- Machine learning applications.
- Streaming.
- Iterative and interactive tasks.
- Fast tasks.

## 2.7 Big Data analytics

### 2.7.1 Definition

"is a process used to extract meaningful insights, such as hidden patterns, unknown correlations, market trends, and customer preferences " [22]

### 2.7.2 Steps of big data analytics

These steps were brought from [23]

1. **Data collection:** collect data from various sources such as web server logs, social media content...etc
2. **Data storage:** store what you collect in repositories such as NoSQL databases, Data lake, data warehouse...
3. **Data processing:** it's about 2 subprocesses
   (a) Data cleaning: clean what you store from anomalies and uncertainty...
   (b) Data analysis: by using several techniques called in some cases machine learning; such as data mining, text mining...etc
4. **Data visualization:** here we shape the results in a smoothy form; to facilitate the decision making.

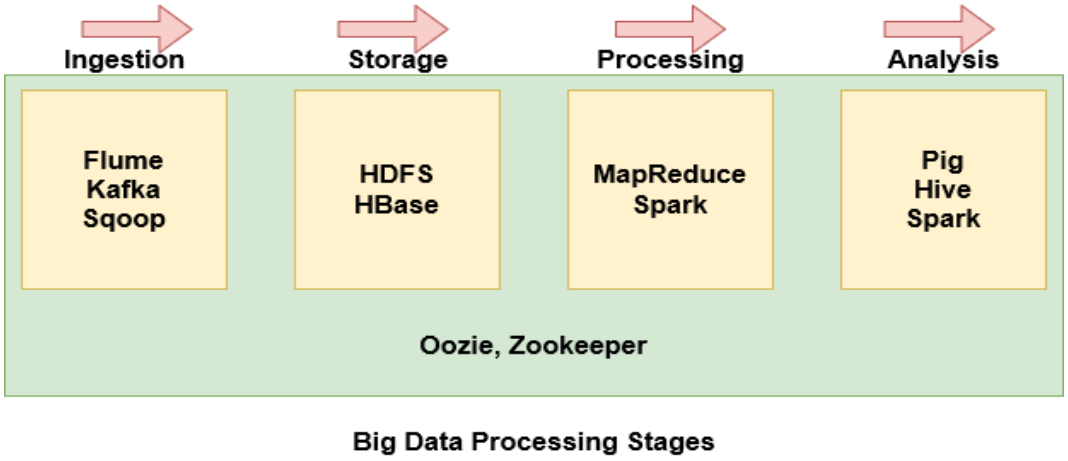



Figure 2.11: Big data analytics steps & tools [24]

It's easily to observe that the big data analytics phases (Figure 2.11) are inspired from the KDD steps.

## 2.8 Big data challenges

This section is inferred from[25]

- **Managing large volume of data**: this obligates to offer and effectively use the appropriate platforms; from data capturing until earn benefits from this data.
- **Define the tools**: depending on the business nature, the benefit wanted, and the cost.
- **Security**: big data means non-accurate data. Which will lead to security alerts, knowing that a scalable environment needs innovative security tricks.
- **The effective exploit of earned knowledge**: the real gain of big data is neither its innovation nor pattern discovery but it is the effective use of its benefits.

## 2.9 Conclusion

We met in this chapter; big data from different sides: as data, it's about various types of data. As a process, it's an innovative type of processing that stands on scalability. As consequences, we live them, as we see how marketing was developed? How do the ads that we are seeing depend on our search queries...etc. In the next chapter; we will know how to achieve FIM, one of the analysis tasks; in big data.

Chapter 3

# Experimentation & discussion

## 3.1 Introduction

In this chapter we will introduce our experimentation using Python, and Spark MLlib as it offers the implementation of parallel fp-growth. Finally, we will compare the performance of the MLlib FIM implementation to that of the *mlxtend* library's fp-growth implementation to corroborate the results.

## 3.2 Spark machine learning library

Spark machine learning library or MLlib comes to enhance the big data analytics treatment by a set of several tools like[1]:

- Machine learning algorithms: an implementation of a set of machine learning algorithms.
- Featurization: it aims to transform the data no matter its format into numerical data that can be used to make predictions or to facilitate the algorithm's application on the data.
- Pipelines: is a road of data processing means we have raw data and we will apply to it a lot of treatments.
- Persistence: means saving the model of your application on the disk in order to reuse it.
- and more.

MLlib is based on RDDs as low level data structures, can be manipulated by only two operations type [26]:

- Transformation: to product new RDD from an existed one.
- Action : to calculate a result from an RDD.

[1] https://spark.apache.org/docs/latest/ml-guide.html

Spark is an in-memory processing engine, and in order to not waste the main memory. Spark invokes the concept of lazy evaluation. This refers to there being no real calculation until a real obligation comes. That means: Spark waits for the coming of an action to release the set of transformations captured. Which were translated and recapped by a directed acyclic graph (DAG). The DAG comes to guarantee: firstly, the fault-tolerant of the RDD, and secondly the amelioration of the transformations done by the user.
Spark adopts other data structures: data frames, and datasets. All Spark's data structures keep immutability as the main characteristic.[1]

## 3.3 Frequent itemset mining With MLlib

Because Spark is a distributed platform. Therefore, all the solutions offered are parallel. For the FIM problem; Mllib adopted parallel fp-growth which is described in [1] as a solution to the problem.
The idea behind the parallelization described by parallel fp-growth is to divide the large set of transactions into independent small sets, among the workers.
That means: dividing the original transactional database depending on shared items between transactions, transferring for each node a set of transactions not related to the other workers in order to reduce the communication time, treat the problem locally as the traditional fp-growth so we will have a set of fp-trees, and finally aggregate the result between the workers to earn the whole frequent itemsets.

### 3.3.1 Parallel FP-growth steps

Li et al [1] have implemented their parallelization of fp-growth with three Map-reduce phases involve five steps:

- **Phase 1**: involves step 1 and step 2.
- **Phase 2**: represents the fourth step.
- **Phase 3**: represents the final step.

1. **Step 1: (Sharding)** dividing the original database into P successive parts, and storing them in P different computers (nodes).
2. **Step 2: (Parallel counting)** we have P mappers to detect the whole itemset list and calculate their supports by passing the P shards in order to build the list of frequent 1-itemset called F-list.
3. **Step 3: (Grouping items)** divide F-list into Q groups called G-list, where every group has its own id (Gid), knowing that this step can be achieved by one node.
4. **Step 4: (Parallel FP-growth)**
    - Mapper phase: generating group dependent transactions; it relates every G-list with the group of transactions that involves the members of G-list, thus this phase outputs a set of key-values pairs where the key is Gid and the value is the group dependent transactions.

[1] `https://data-flair.training/blogs/spark-tutorial/`

- Reducer phase: for every fragment of group dependent transactions, build its fp-tree & its conditional fp-tree.

5. **Step 5: (Aggregating)** aggregating the result from the resulted fp-trees.

## 3.4 Experimentation

There are two choices to use Spark :

- Installing Spark and using its tools from the shell.
- Installing PySpark; which "*is an interface for Apache Spark in Python*"[1].

we have used the second choice due to:

- The ease of Python use.
- We aim to translate the process into an application. Which is impossible with the first choice as the shell is an interactive environment.

### 3.4.1 Programming environment

**The hardware used in this experimentation** :

- Hp notebook
- RAM: 6,7 Gib
- Disk : 81 Gb
- CPU : AMD®E2-7110 apu with amd radeon r2 graphics × 4

**The software used in this experimentation** :

- Operating system : ubuntu 18,04
- Anaconda platform to use pandas and other machine learning libraries without installation.
- Jupyter notebook[2] as an IDE.

Figure 3.1 depicts the process followed in this experimentation:

[1] https://spark.apache.org/docs/latest/api/python/
[2] `https://github.com/jupyter/`

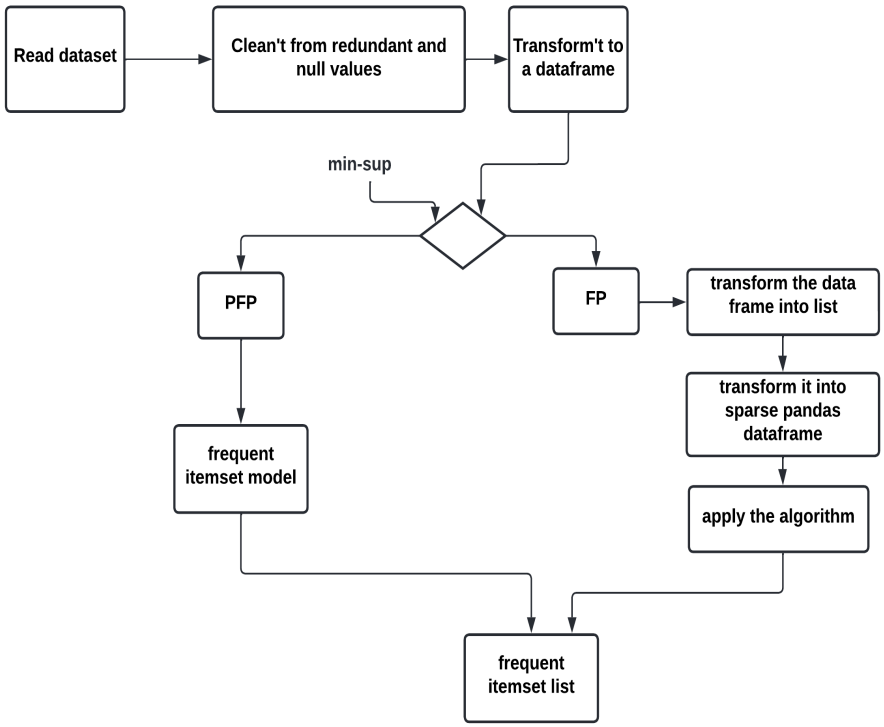



Figure 3.1: Experimentation plan

### 3.4.2 Dataset

For the dataset we have been used three datasets, are all available on the interenet Table 3.1.

| Dataset | Description | Cause of use |
|---|---|---|
| **Grocery store dataset**[1] | it is about 20 transactions. The length of a transaction is between two and four items. With items total equal to 11. It's a portion of a real transactional dataset of a grocery shop. | it is the smallest dataset, we have used with mere test reason. |
| **Mushroom dataset**[2] | it is about 8124 transactions. The length of a transaction is 24 items. The total items are equal to 119. It's a set of transactions containing the characteristics of various species of mushrooms | it's widely used in big data FIM test like [27],[28] |

[1] `https://www.kaggle.com/datasets/shazadudwadia/supermarket?select=GroceryStoreDataSet.csv`

[2] `https://github.com/zhang943/Spark-Apriori/tree/master/data`

| | | |
|---|---|---|
| **Chess dataset**[2] | it is about 3196 transactions. The length of a transaction is 38 items. The total items are equal to 75. It is a set of examples of chess positions described only by the coordinates of the pieces on the board. | it's widely used in big data FIM test like [27],[28] |

Table 3.1: Datasets used in the experimentation

### 3.4.3 Program development

Every Spark application consists of two main types of processes: one driver and a set of executors. The essential part is the driver as it takes the responsibility of responding to the user's program and scheduling the work among the workers. Which are responsibles for achieving the portion of work received from the driver. The whole will be managed by a cluster manager. Figure 3.2.

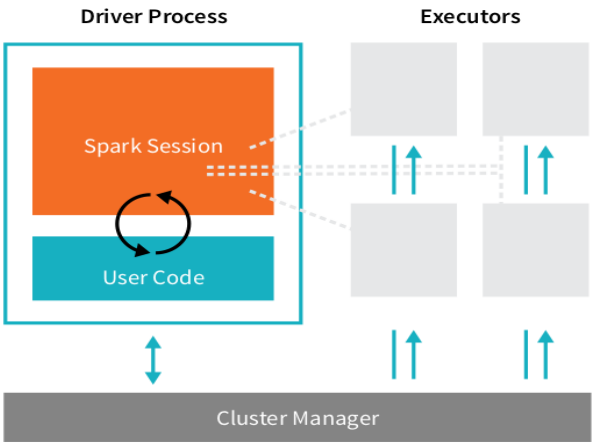



Figure 3.2: Spark application anatomy [29]

To start the driver's work we must import a SparkSession and instantiate it. Figure3.3.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("FIM").master("local[*]").config("spark.driver.memory", "4g").getOrCreate()
```

Figure 3.3: Starting a SparkSession

We named the application by FIM, and enabled it to work locally with as many threads as possible (depending on the CPU cores). We gave the driver a memory space equal to 4GB. This way is the simplest manner of Spark configuration.[1]

[1] https://spark.apache.org/docs/latest/configuration.html

After that, we load the dataset, which can be entered by a local path or from HDFS. We must clean it from null values, and redundant ones. Because the general FIM does not accept duplication of an item in a transaction. And then transform it to a data frame composed of two columns (transaction-id, and the list of items of that transaction).

### a) MLlib FIM

In order to use the model offered by MLlib, we must first import fp-growth. Figure 3.4.

```
from pyspark.ml.fpm import FPGrowth
```

Figure 3.4: Importing MLlib fp-growth

#### a.1) Problem parameters

The MLlib FIM implementation takes the following parameters:

- **min-support**: in our case will be defined by the user.
- **min-confidence**: we will not use it. Because we are focused on frequent itemsets mining.
- **items's column**: we must index the name of the column that involves the items. Because the FIM implementation will apply on a data frame.
- **numPartitions**: refers to the number of the portions of the dataset. We will not use it. Because the experimentation occurs locally in one node, so we don't need this parameter.

The MLlib FIM implelemtation gives the possibility to achieve:

- Frequent itemsets model.
- Association rules model.
- Do some predictions depending on the association rules mined.

We have used the first model. Figure 3.5.

```
fpGrowth = FPGrowth(itemsCol="items", minSupport=minsup)
model = fpGrowth.fit(df)
```

Figure 3.5: Instantiate the Fp-growth model and fit it on the dataframe

With the aim of accrediting the performance of MLlib FIM, we used the fp-growth model offered by mlxtend library.

### b) Mlxtend fp-growth

**MLxtend**[1]: is a python library for machine learning tasks. It enables the use of many analytic techniques, such as clustering, frequent pattern mining, regression, and more.
In its models, mlxtend uses pandas data frames.

**Pandas**: is a python library destinated to data science tasks.

Firstly, we must import Panads; as we have to use its data frames, we must import the *TransactionEncoder* to transform the transactional dataset, into a sparse dataset[2]. Because *mlxtend* fp-growth requires the use of a sparse dataset, and then we have to import the fp-growth model. Figure 3.6.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent patterns import fpgrowth
```

Figure 3.6: Necessary importation to use *mlxtend*'s fp-growth

After that, We transform the original dataframe into a list, transform it into a sparse dataset using *TransactionEncoder* object, and transform it into a pandas data frame, at the end we will be able to apply fp-growth on the resulted dataframe. Figure 3.7

```
dataset = []
for i in df.collect():
    dataset.append(i[1])
te=TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df=pd.DataFrame(te_ary, columns=te.columns_)
data = fpgrowth(df, min_support=self.min_sup, use_colnames=True)
```

Figure 3.7: *mlxtend* fp-growth's code

## 3.4.4 Application screenshots

In the following, we present the main windows of the application.

[1] http://rasbt.github.io/mlxtend/

[2] the column names are the the items' names, the rows represent the transactions. if the item is present in the transaction the column value will be True, else it gets False

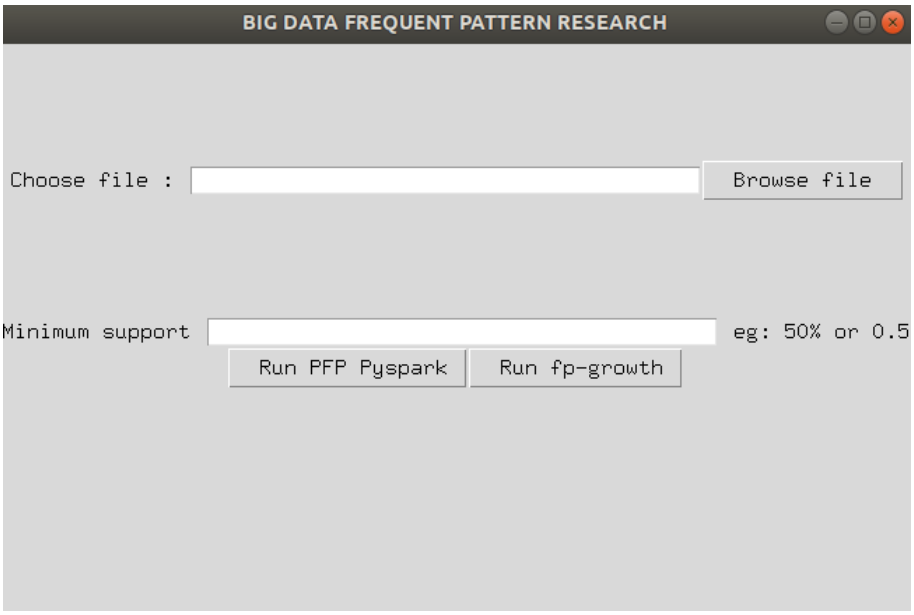



Figure 3.8: The main window

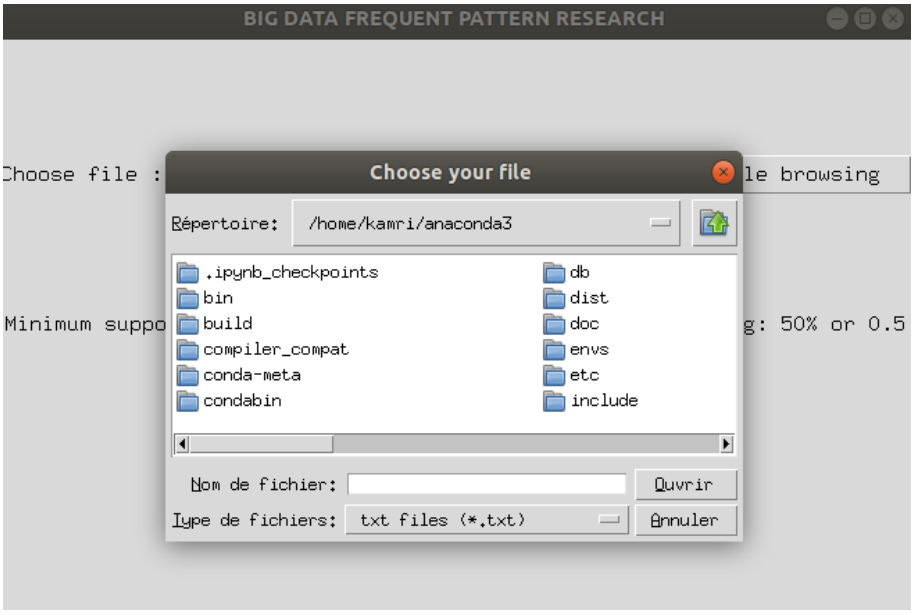



Figure 3.9: The main window :local file browsing

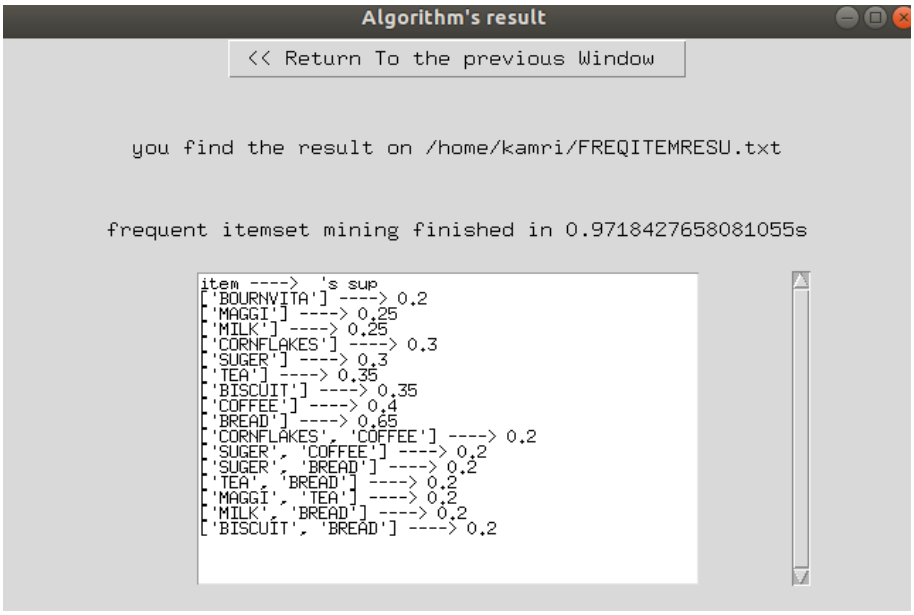



Figure 3.10: The window of the final result

### 3.4.5 Results and discussion

We have been registered the results of our experimentation in the below charts. Figure 3.11 & Figure 3.12.

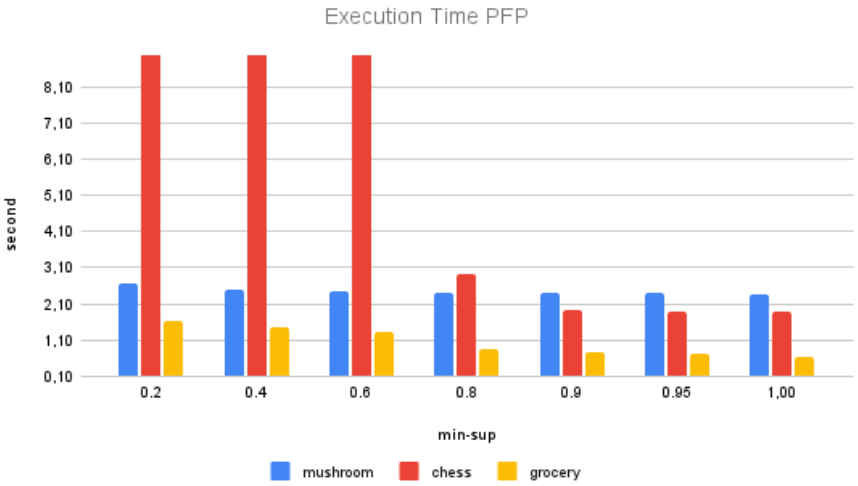



Figure 3.11: PFP execution time chart

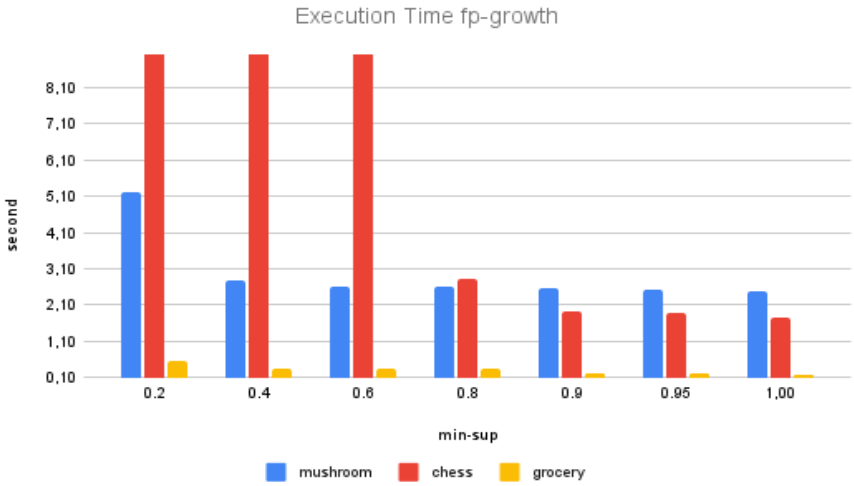



Figure 3.12: Fp-growth execution time chart

In these charts, we can see the behavior pulled from both algorithms. Where the x-axis represents the min-sup values, and the y-axis represents the time in seconds.

1. We can observe that the higher the min-sup values, the shorter the execution time. We have seen that with all the datasets.
2. We can detect that the larger the dataset, the more power PFP. We have seen it with the mushroom dataset with min-sup equal to 0.2; fp-growth took more than 5 seconds, while PFP took less than 3 seconds.
3. And with a deep remark, because the chess dataset has a big transaction length(38 items), it had registered the longest execution time (Especially with min-sup $\leq$ 0.6). Though the mushroom dataset has the biggest number of transactions (8124 transactions).

From this experimentation, and the remarks captured, we can infer that:

- Higher min-sup values accelerate the FIM treatment (from the first remark ).
- PFP MLlib implementation, can fit with big data frequent itemset mining (from the second remark ).
- The length of the transaction has a major role in the execution time, as it can indicate the length of the largest frequent itemset (from the third remark ).
- The transaction length, the min-sup value, and the dataset size are the major effect factors of FIM.

**Note**: The experimentation occured locally with a humble material so that why we cannot prove and generalize the results.(return to 3.4.1)

## 3.5 Conclusion

In this chapter, we saw the PFP MLlib implementation with python. And we have compared its performance with *mlxtend* fp-growth. Where we inferred that PFP MLlib can fit with big data frequent itemset mining.

# Conclusion

Nowadays, there is a revolution in the world. Since the introduction of big data, many fields have developed rapidly; like marketing, commerce, and science fields. This evolution is actually due to big data analytics.

In this thesis, we were interested in big data frequent itemset mining, one of the analytics techniques. Where we dealt with the theme from different sides. Theoretically, we saw data mining as a step towards knowledge, and we introduced the famous solutions of FIM; this was the first. Secondly, we defined big data as an enormous variant amount of data, that cannot predict its coming as it depends on velocity. Where we must invoke innovative techniques to store it, process it, and analyze it.
Practically, we saw how the problem can be solved with Apache Spark MLlib FIM implementation. That achieved a good performance as a big data solution to the problem.
Personally, this theme added to us many things such as:

- We have enhanced our knowledge about data mining.
- We had the opportunity to know more about big data.
- Dealing with Apache Spark and its machine learning library. Where we met a lot of new terms and techniques.
- We had the chance to meet the Python data science tools.
- This experience enabled us to be more aware of data analytics tools.

As perspectives on the subject we suggest the below points:

- Trying the Spark FIM MLlib implementation in a real cluster, to capture its real performance.
- Continue the task to reach the association rules, and thus decipher the hidden relations among the frequent itemsets.
- Solve the problem with Apache Pig, and compare the results with Spark MLlib FIM.
- Build a decision support system based on the association rules technique. By using the Spark MLlib association rules model.

# Bibliography


[1] Haoyuan Li, Yi Wang, Dong Zhang, Edward Chang, and Ming Zhang. Pfp: Parallel fp-growth for query recommendation. In *ACM Recommendation Systems*, 2008.

[2] Han Jiawei, Kamber Micheline, and Pei Jian. *Data Mining Concepts and Techniques*. Morgan Kaufmann, 2011. Third Edition.

[3] Tutorial and example. Data mining tasks. `https://www.tutorialandexample.com/data-mining-tasks`, 2021. last consultation : 14/05/2022.

[4] Jackson Munyai. Getting started with eclat algorithm in association rule mining. `https://www.section.io/engineering-education/eclat-algorithm-in-python/`, 2021. last consultation : 14/05/2022.

[5] Mohammod Akib Khan, Kazi Mohammad Solaiman, and Touhid Hossain Pritom. Market basket analysis for improving the effectiveness of marketing and sales using apriori, fp growth and eclat algorithm. 2017.

[6] Gartner. Big data definition. `https://www.gartner.com/en/information-technology/glossary/big-data`. last consultation : 14/05/2022.

[7] IBM. Big data analytic. `https://www.ibm.com/uk-en/analytics/hadoop/big-data-analytic`. last consultation : 14/05/2022.

[8] NIST wg National Institute of Standards and Technology US working group. What is big data? `https://datasciencedegree.wisconsin.edu/data-science/what-is-big-data/`, 2015. last consultation : 14/05/2022.

[9] Nazari Elham, Afkanpour Marziyeh, and Tabesh Hamed. Big data from a to z. *Frontiers in Health Informatics*, 2019.

[10] Taylor-Sakyi Kevin. Big data: Undrestanding big data. `https://arxiv.org/abs/1601.04602`, 2016.

[11] Tom Shafer. The 42 v's of big data and data science. `https://www.kdnuggets.com/2017/04/42-vs-big-data-data-science.html`, 2017. last consultation : 14/05/2022.

[12] Tom White. *Hadoop the definitive guide*. OREILLY, 2015. fourth edition.

[13] Priya Pedamkar. Mapreduce word count. `https://www.educba.com/mapreduce-word-count/`, -. last consultation : 14/05/2022.

[14] Aniruddha Bhandari. Introduction to the hadoop ecosystem for big data and data engineering. https://www.analyticsvidhya.com/blog/2020/10/introduction-hadoop-ecosystem, 2020. last consultation : 14/05/2022.

[15] krishnaprasad k. Yet another resource negotiator(yarn). https://medium.com/nerd-for-tech/yet-another-resource-negotiator-yarn-internals-and-architecture-af33da73e7a, 2021. last consultation : 14/05/2022.

[16] Mohammed Hadi, Ahmed Lawey, Taisir El-Gorashi, and Jaafar Elmirghani. Big data analytics for wireless and wired network design: A survey. *Computer Networks*, 132:180–199, 2018.

[17] Vladimir Kaplarevic. Apache hadoop architecture explained (with diagrams). https://phoenixnap.com/kb/apache-hadoop-architecture-explained, 2020. last consultation : 14/05/2022.

[18] Mèhdi Ben Hamida. Big data management for manufacturing intelligence application. https://www.researchgate.net/publication/326572328_Big_Data_Management_for_Manufacturing_Intelligence_Application, 2017.

[19] Tutorialspoint. Apache spark introduction. https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm, -. last consultation : 14/05/2022.

[20] Goran Jevtic. Hadoop vs spark - detailed comparison. https://phoenixnap.com/kb/hadoop-vs-spark, 2020. last consultation : 14/05/2022.

[21] Laura Shiff. Batch processing: An introduction. https://blogs.bmc.com/what-is-batch-processing-batch-processing-explained, 2020. last consultation : 14/05/2022.

[22] Simplilearn. What is big data analytics and why it is important? https://www.simplilearn.com/what-is-big-data-analytics-article, 2022. last consultation : 14/05/2022.

[23] Chai Wesley, Mark Labbe, and Craig Stedman. big data analytics. https://www.techtarget.com/searchbusinessanalytics/definition/big-data-analytics, -. last consultation : 14/05/2022.

[24] Aniruddha Bhandari. Introduction to the hadoop ecosystem for big data and data engineering. https://www.analyticsvidhya.com/blog/2020/10/introduction-hadoop-ecosystem/, 2020. last consultation : 14/05/2022.

[25] George Lawton. 10 big data challenges and how to address them. https://www.techtarget.com/searchdatamanagement/tip/10-big-data-challenges-and-how-to-address-them, 2022. last consultation : 14/05/2022.

[26] Spark documentation. Rdd operations. https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-operations, -. last consultation : 29/06/2022.

[27] Hamdad Leila and Benatchba Karima. Association rules mining exact, approximate and parallel methods: A survey. *Springer nature computer science*, 2021.

[28] Djenouri Youcef, Djenouri Djamel, Chaun-wei lin Jerry, and Belhadi Asma. Frequent itemset mining in big data with effective single scan algorithms. *IEEE*, 2018.

[29] Databricks team. A gentle introduction to apache spark, 2017.