TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Kirill Timofejev 205934IAIB

DISTRIBUTION OF APRIORI ALGORITHM FOR LUNG CANCER DATA SET USING APOLLO FRAMEWORK

Bachelor's Thesis

Supervisor: Mahtab Shahin
MSc

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kirill Timofejev 205934IAIB

APRIORI ALGORITMI HAJUTAMINE KOPSUVÄHI ANDMEKOGUMI JAOKS KASUTADES APOLLO RAAMISTIKKU

Bakalaureusetöö

Juhendaja: Mahtab Shahin
MSc

Tallinn 2023

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Kirill Timofejev

22.05.2023

Abstract

This work aims to investigate options for distributing the existing association rule mining algorithm, Apriori. Apriori is one of the typical algorithms, which is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. It is frequently used in mining rules over relational databases.

The contribution of the work is implementing the algorithm in the distributed environment to improve its speed and decrease the memory load on the local machine. As dataset size increases over time, our computational requests also increase, so we need to find a way to distribute the load between different machines.

During the work, the theoretical material was gathered and analyzed, the basics of the problematic topic were introduced, and possible algorithm prototypes were created as abstractly as technically. Using Apollo Framework, prototypes were implemented and tested.

As the work was finished and algorithms were tested, it was possible to state that goals were achieved. Ways of algorithm distribution were found, and the total dataset mining time has been decreased, so the algorithm's speed was increased.

The thesis is in English and is 48 pages long, including 6 chapters, 16 figures and 12 tables.

Annotatsioon

Selle töö eesmärk on uurida viisi olemasoleva assotsiatiivsete reeglite kaevandamise algoritmi, Apriori, jaotamiseks. Apriori on üks tüüpilistest algoritmidest, mis oli välja töötanud R. Agrawal ja R. Srikant 1994. aastal sagedaste üksuste leidmiseks Boole'i assotsiatiivsete reeglite kaevandamiseks. Seda kasutatakse sageli reeglite kaevandamiseks relatsioonide andmebaaside üle.

Töö panus seisneb algoritmi rakendamises jaotatud keskkonnas, et kiirendada algoritmi töö ja vähendada mälukoormust masinatel. Kuna andmekogumite suurus suureneb ajaga, suurenevad ka meie arvutuslikud nõudmised. Sellepärast peame leidma viisi koormuse jaotamiseks erinevate masinate vahel.

Töö käigus koguti ja analüüsiti teoreetilist materjali, tutvustati probleemse teema põhitõdesid ning loodi võimalikud algoritmi parenduste prototüübid nii abstraktselt kui ka tehniliselt. Prototüübid rakendati ja testiti Apollo raamistiku abil.

Kuna töö oli lõpetatud ja algoritmid testitud, on võimalik öelda, et eesmärgid on saavutatud. Leiti viisid algoritmi jaotamiseks ja kogu andmekogumi kaevandamise aeg on vähenenud, seega suurenes algoritmi kiirus.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 48 leheküljel, 6 peatükki, 16 joonist, 12 tabelit.

List of Abbreviations and Terms

CSV	Comma-separated values
JSON	JavaScript Object Notation
API	Application Programming Interface
CPU	Central Processing Unit
IDE	Integrated Development Environment
VM	Virtual Machine
AWS	Amazon Web Services
AFCL	Apollo Function Choreography Language
MB	megabytes
RAM	Random Access Memory
GHz	Gigahertz

Table of Contents

List of Figures

List of Tables

1. Introduction

Nowadays, many commercial and non-commercial companies collect vast amounts of data from their daily operations. So, for example, stores collect data about the purchases made by their customers, or banks collect data on their creditors. With the increasing amount of data, these companies have found a practical way to use it. They started looking for association rules among data. Now stores can evaluate the dependence of the purchase of one item on another, or banks can assess the risk of default on a loan. For these purposes, companies use associative analysis. One of the typical algorithms for retrieving those rules is Apriori.

Although Apriori is one of the most popular algorithms for mining association rules because of its simplicity and speed, it is becoming slower as data increases. With the number of clients increasing - companies' databases are also growing, and analyzing this data takes more time, memory and computer capabilities. Among the more significant amount of data, it is possible to find rare rules, respectively, but this also complicates computer calculations, due to which the search speed decreases and, accordingly, the search cost increases.

The problem, in this case, is that it is worth finding ways to speed up the search for association rules when one computer can no longer complete the task so quickly. and it also does not make sense for a company to wait for any new kind of CPU that is going to speed up the algorithm. Therefore, it is worth developing an algorithm to distribute the algorithm's work among several computers.

1.1 Existing solutions

Many researches on the Internet provide information about distributing the Apriori algorithm.

In general, they all are similar. One was chosen for Example[1]. From this research was found one possible solution. This research states that it is possible to distribute Apriori by dividing data horizontally into approximately equal parts. After dividing it into parts, the developer should send those parts to different nodes and run Apriori using this small data. After the computation of frequent itemsets is done locally on another machine - they are gathered together on the primary node, and based on the results, frequent itemsets are

retrieved by recalculating support value globally. Then those itemsets are filtered and can be used to generate rules.

At first sight, it seems logical. Although it is the right solution, more is needed for accurate results. The paper cited above does not exist a comparison of accuracies on the plain Apriori and their solution. However, this solution can only catch some rare rules that globally are frequent enough and can send locally frequent itemset that is globally not frequent, bringing odd computations.

The bachelor's work addresses this research gap by comparing the approach with the standard Apriori algorithm. Furthermore, an additional algorithm prototype will be developed to enrich the comparative analysis further. By evaluating the performance and accuracy of these approaches, this study endeavours to contribute to the existing body of knowledge surrounding the distribution of the Apriori algorithm.

1.2 Runtime environment

Different Internet researchers use different technologies like Spark, MapReduce, Hadoop and others. For this work, Apollo Framework will be used to prove that this algorithm can work correctly.[2] The Framework will review results that can be compared.

Apollo Framework was chosen because of its option to use different sources like AWS, IBM and docker with local systems. In this case, we have different opportunities and flexibility to choose sources based on the data set and our expertise.

1.3 Dataset used for tests

Choosing a medicine dataset about lung cancer was decided to find interesting rare rules besides the main work.[3] Mining association rules on medical datasets offers valuable insights and benefits for healthcare professionals and researchers. With rich and complex data encompassing patient demographics, medical history, symptoms, diagnoses, treatments, and outcomes, medical datasets provide a fertile ground for discovering hidden relationships and patterns. Researchers can uncover previously unknown connections between medical conditions, treatments, risk factors, and patient outcomes by applying association rule mining techniques. This knowledge can support clinical decision-making, personalized medicine, quality improvement, and healthcare planning. From aiding in clinical decision support and enabling personalized treatment plans to identifying areas for improvement and optimizing resource allocation, mining association rules on medical

datasets contribute to advancing medical knowledge, enhancing patient care, and improving healthcare practices.

1.4 Goals

After analyzing alternative solutions, the author investigated ways to improve an existing algorithm. For this work, such goals were defined as

- Gather and analyze theoretical material on the "Association rule mining ", "Apriori algorithm", and related topics.
- Describe a new algorithm prototype, which parts can be integrated into a distributed computing framework.
- Implement prototypes using the Apollo framework.
- Run multiple performance tests using the Apollo framework and existing medicine dataset to measure the completion time and compare them to each other.

2. Theoretical part

2.1 What is Apriori?

The Apriori algorithm is an influential algorithm for discovering frequent itemsets and association rules from large datasets. It is widely used in data mining and market basket analysis. The algorithm works by iteratively generating candidate itemsets and pruning those that do not meet a minimum support threshold. It employs a breadth-first search strategy to efficiently explore the space of itemsets. The Apriori algorithm's strength lies in its ability to handle datasets with a large number of items and transactions. It provides valuable insights into item associations, allowing businesses to make informed decisions and improve various aspects such as product placement, cross-selling, and recommendation systems.

E.g. Almost every hospital gather their patients and their symptoms data. So this data can be analyzed later to make some common conclusions to improve medicine quality. An example table with symptoms can be seen in Table 1.

Table 1. *Example of medicine dataset.*

PID	**Items**
1	{Male, 70, Smoking, Lung Cancer}
2	{Female, 30}
3	{Male, 60, Consuming Alcohol, Yellow Fingers, Lung Cancer}
4	{Male, 25, Consuming Alcohol, Shortness of Breath}
5	{Female, 45, Smoking, Consuming Alcohol, Shortness of Breath, Lung Cancer}

In the table above, each row is a unique patient and a set of his/her symptoms that were present at the inspection. This information can be useful for any doctor to look for similar symptoms. In this paper, the focus is made on finding associations between those symptoms using Association Analysis.

With Association Analysis, the outputs are association rules that show the relationship between symptoms. A classic example {Smoking} -> {Lung Cancer}, which states that transactions containing Smoking usually tend to contain Lung Cancer too. So from this rule, we can make a conclusion that usually, the person who smokes tends to have cancer

too. With this information, doctors can usually use it to define the diagnosis of the patient.

That is why we need Apriori to find interesting itemsets. Let's dive in to understand how it works. Firstly, we will look at metrics used inside the algorithm to generate interesting rules.

2.2 Association Rules Metrics

Association Analysis uses some metrics to generate these associations. To make it easier to understand the table with the binary presentation of Table 1 can be seen in Table 2

Table 2. *Binary representation of medicine dataset.*

PID	**Sex**	**Age**	**Smoking**	**Consuming Alcohol**	**Yellow Fingers**	**Shortness of Breath**	**Lung Cancer**
1	Male	70	1	0	0	0	1
2	Female	30	0	0	0	0	0
3	Male	60	0	1	1	0	1
4	Male	25	0	1	0	1	0
5	Female	45	1	1	0	1	1

Each row corresponds to a patient, but each column now corresponds to a symptom. Then the value in each cell is one if the symptom is present and 0 if not. Let

$$I = \{i_1, i_2, i_3, ..., i_d\}$$

be the set of all symptoms in a hospital database and

$$T = \{t_1, t_2, t_3, ..., t_N\}$$

be the set of all patients. Each patient t_i contains a subset of symptoms chosen from I. A collection of zero or more symptoms is called an itemset.

2.2.1 Support

An important property of an itemset is its support. It basically refers to the number of transactions that contain a particular itemset divided by the number of all transactions and informs the percentage of transactions containing itemset. Mathematically, the support count of itemset X can be stated as follows:

$$support(X) = \frac{\textit{Number of transactions containing X}}{\textit{Number of all transactions}}$$

2.2.2 Confidence

Another important property is confidence. It shows the probability of Y being present in a transaction with X. Mathematically; confidence can be stated as follows;

$$confidence(X->Y) = \frac{\textit{support(}X \cup Y\textit{)}}{\textit{support(X)}}$$

2.2.3 Lift

This says how likely item Y is purchased when item X is purchased while controlling for how popular item Y is. A lift of 1 implies no association between items. A lift value greater than one means that item Y is likely to be bought if item X is bought (positively correlated), while a value less than one means that item Y is unlikely to be bought if item X is bought (negatively correlated).[4]. Mathematically, the lift can be stated as follows:

$$lift(X->Y) = \frac{\textit{confidence(X -> Y)}}{\textit{support(Y)}}$$

2.3 Implementation

Apriori consists of 2 steps. The first is finding frequent itemsets, and the second is generating association rules out of frequent itemsets.

2.3.1 Finding frequent itemsets

At this step, we want to define frequent itemsets used to generate rules. In general, a data set that contains k items can potentially generate up to $2k - 1$ frequent itemsets, excluding the null set. Because k can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large.[5] That is why The Apriori principle is used. The Apriori principle states:

> If an item set is frequent, then all of its subsets must also be frequent.[5]

To see why this rule can be helpful, suppose we now know items a and b are already infrequent. Using the rule, we can automatically drop any supersets containing a and b because we know they will not surpass the minimum support threshold we've set. In the diagram below, we can see that a large part of the diagram (Figure 1) can be pruned (we now only need to compute the support for the grey node).[4]

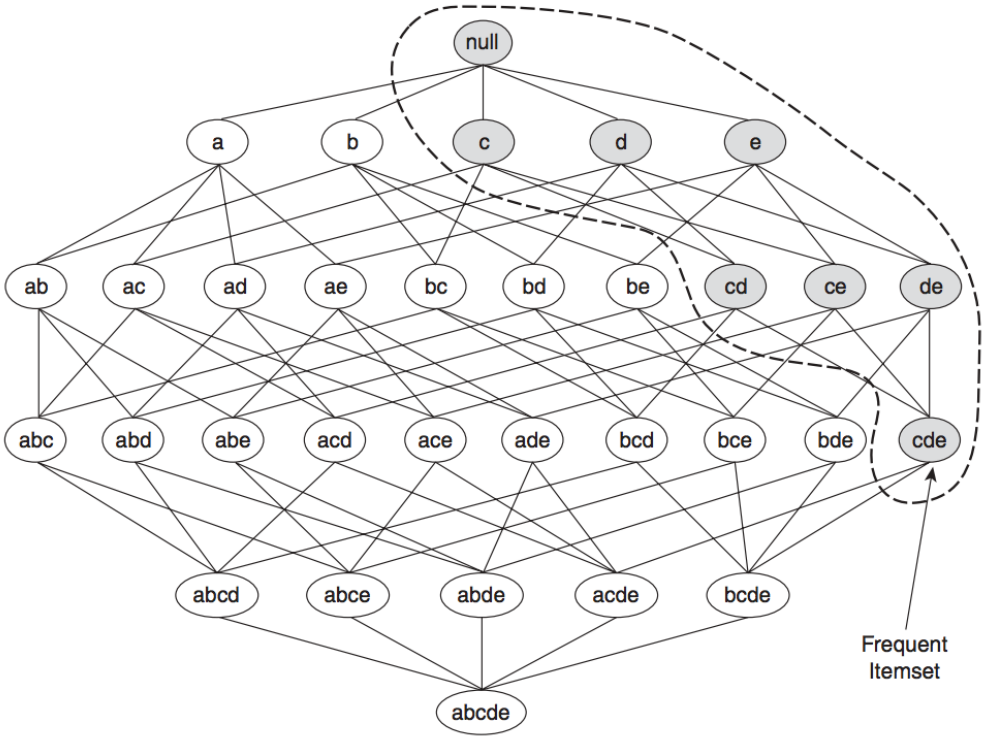

Figure 1. *An illustration of the Apriori principle.*

The pseudocode for the frequent itemset generation part of the Apriori algorithm is shown in Figure 2. Let C_k denote the set of candidate k-itemsets and F_k denote the set of frequent k-itemsets.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: $C_k =$ apriori-gen(F_{k-1}). {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t =$ subset(C_k, t). {Identify all candidates that belong to t}
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$. {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result $= \bigcup F_k$.

Figure 2. *Frequent itemset generation of the Apriori algorithm.*
[5]

Generating k-candidates

$F_{k-1} \times F_{k-1}$ Method The candidate generation procedure in the apriori-gen function merges a pair of frequent $(k-1)$-itemsets only if their first $k-2$ items are identical. Let $A = a_1, a_2, ..., a_{k-1}$ and $B = b_1, b_2, ..., b_{k-1}$ be a pair of frequent $(k-1)$-itemsets. A and B are merged if they satisfy the following conditions[5]:

$$a_i = b_i (\textit{for i} = 1, 2, ..., k-2) \textit{ and } a_{k-1} = b_{k-1}.$$

2.3.2 Generating association rules

This part is easier. You should take every generated frequent itemset, and for each item there, calculate its confidence depending on the rest of the itemset. X is one of the found itemsets, then we should find confidence of $X - Y -> Y$ where Y is any one of the items present in X. Pseudocodes are shown in Figure 3 and Figure 4.

```
1: for each frequent k-itemset f_k, k ≥ 2 do
2:    H_1 = {i | i ∈ f_k}          {1-item consequents of the rule.}
3:    call ap-genrules(f_k, H_1.)
4: end for
```

Figure 3. *Generating rules pseudocode.*
[5]

```
1: k = |f_k|    {size of frequent itemset.}
2: m = |H_m|    {size of rule consequent.}
3: if k > m + 1 then
4:    H_{m+1} = apriori-gen(H_m).
5:    for each h_{m+1} ∈ H_{m+1} do
6:       conf = σ(f_k)/σ(f_k − h_{m+1}).
7:       if conf ≥ minconf then
8:          output the rule (f_k − h_{m+1}) ⟶ h_{m+1}.
9:       else
10:         delete h_{m+1} from H_{m+1}.
11:      end if
12:   end for
13:   call ap-genrules(f_k, H_{m+1}.)
14: end if
```

Figure 4. *Calculating confidence of k-itemset pseudocode.*
[5]

3. Algorithm prototypes

3.1 Choosing approach for distribution work

This study investigates two distinct approaches for the distribution of the Apriori algorithm, each addressing the challenge of calculating frequent itemsets independently on distributed nodes. The main objective is to efficiently distribute the computation while ensuring accurate and reliable results.

The first approach involves dividing the dataset at the initial stage of the algorithm. The dataset is partitioned into smaller, approximately equal portions, and these subsets are then assigned to different nodes within the distributed system. By doing so, each node can independently execute the Apriori algorithm on its assigned subset of data. This approach allows for parallel processing and harnesses the computational power of multiple nodes, thereby potentially speeding up the overall execution time.

The second approach explores the concept of setting up communication and creating a distributed database among the nodes. In this scenario, the support map, which keeps track of the occurrence of itemsets, is stored and shared among the nodes. However, it is important to note that this approach introduces additional complexities and overhead. The distributed database needs to handle concurrent updates and continuously synchronize the data, which can impact the overall system performance. In particular, the process of blocking mutex on a distributed map takes longer compared to the synchronization between local threads in a single-machine multi-threaded implementation.

Given the potential challenges and performance implications associated with maintaining a distributed database, the decision was made to adopt the approach of dividing the dataset. This approach allows each node to work independently on its allocated subset of data. By minimizing the need for constant synchronization and inter-node communication, this approach offers improved efficiency and reduced computational overhead.

Here is shown example Table 3 on which prototypes will be explained further.

Table 3. *Binary medicine dataset.*

PID	**Sex**	**Age**	**Smoking**	**Consuming Alcohol**	**Yellow Fingers**	**Shortness of Breath**	**Lung Cancer**
1	Male	80	1	0	0	0	1
2	Female	44	0	0	0	0	0
3	Male	62	0	1	1	0	1
4	Male	85	0	1	0	1	0
5	Female	87	1	1	0	1	1
6	Male	23	1	0	0	0	1
7	Female	30	0	0	0	0	0
8	Male	56	0	1	1	0	1
9	Male	39	0	1	0	1	0
10	Female	20	1	1	0	1	1
11	Female	25	0	0	0	0	0
12	Male	38	0	1	1	0	1
13	Male	43	0	1	0	1	0
14	Female	62	1	1	0	1	1
15	Female	67	1	1	0	1	0
16	Male	57	1	0	0	0	1

3.2 Horizontally divided dataset

The idea of this approach was taken from research paper that was chosen for Example[1]. In this approach the main idea is to divide data horizontally. The workflow of the algorithm is shown in Figure 5.

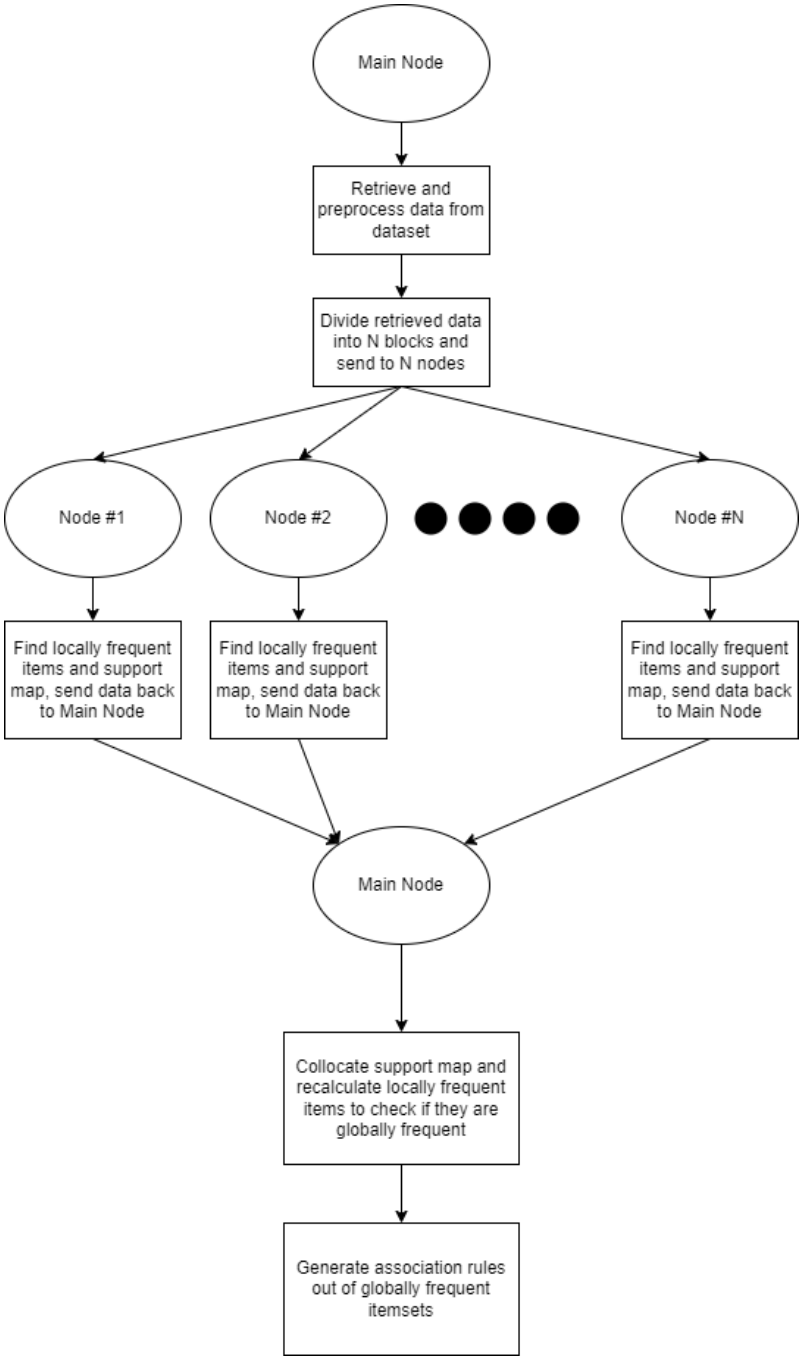

Figure 5. *Horizontally divided dataset approach.*

3.2.1 Dividing data

Main goal in this part is to divide data. Corresponding to Table 3 it would look like this in case we had 4 nodes that can do calculations. Tables 4, 5, 6, 7 are referred as an example of dividing dataset into equal blocks of data.

Table 4. *Divided dataset part 1.*

PID	**Sex**	**Age**	**Smoking**	**Consuming Alcohol**	**Yellow Fingers**	**Shortness of Breath**	**Lung Cancer**
1	Male	80	1	0	0	0	1
2	Female	44	0	0	0	0	0
3	Male	62	0	1	1	0	1
4	Male	85	0	1	0	1	0

Table 5. *Divided dataset part 2.*

PID	**Sex**	**Age**	**Smoking**	**Consuming Alcohol**	**Yellow Fingers**	**Shortness of Breath**	**Lung Cancer**
5	Female	87	1	1	0	1	1
6	Male	23	1	0	0	0	1
7	Female	30	0	0	0	0	0
8	Male	56	0	1	1	0	1

Table 6. *Divided dataset part 3.*

PID	**Sex**	**Age**	**Smoking**	**Consuming Alcohol**	**Yellow Fingers**	**Shortness of Breath**	**Lung Cancer**
9	Male	39	0	1	0	1	0
10	Female	20	1	1	0	1	1
11	Female	25	0	0	0	0	0
12	Male	38	0	1	1	0	1

Table 7. *Divided dataset part 4.*

PID	Sex	Age	Smoking	Consuming Alcohol	Yellow Fingers	Shortness of Breath	Lung Cancer
13	Male	43	0	1	0	1	0
14	Female	62	1	1	0	1	1
15	Female	67	1	1	0	1	0
16	Male	57	1	0	0	0	1

3.2.2 Apriori algorithm

In this step, the conventional Apriori algorithm, which was described in Chapter 2, is applied to the block of data obtained in the Chapter 3.2.1.

The objective of this step is to generate association rules based on the frequent itemsets discovered within the specific block of data. The Apriori algorithm follows a systematic approach, involving the generation of candidate itemsets, the evaluation of their support, and the pruning of infrequent itemsets.

During the execution of the Apriori algorithm on the block of data, a local hash map is created. This hash map serves as a crucial data structure that stores the support metrics for each mined itemset. The support metric represents the frequency or occurrence count of an itemset within the given block of data. By maintaining this local hash map, the algorithm efficiently tracks the support values of the itemsets, enabling subsequent steps to utilize this information for further analysis. Example of reuslt shown on Figure 6.

The local hash map generated in this step becomes essential in the subsequent phase, where the results from multiple blocks of data need to be combined. By having a local hash map for each block, the integration of the results becomes more manageable.

This step of applying the Apriori algorithm to the block of data contributes to the creation of association rules and the generation of a local hash map containing the support metrics for each mined itemset.

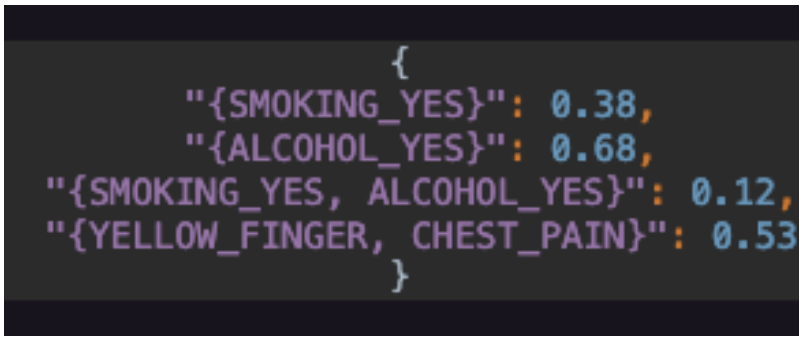

Figure 6. *Example of calculated support hash map.*

3.2.3 Collocating result

This is the most important step in this approach as it gives the final result of distributed computing. In this step hash maps, obtained in previous step, can be merged or synchronized to restore the collective support metrics for the complete dataset. This process ensures that all the necessary information regarding the support of itemsets is consolidated, enabling a comprehensive analysis of association rules across the entire dataset. Example of collocating is shown on Figure 7.

Using the collocated hash map, each rule mined is evaluated to determine whether it satisfies the predefined support and confidence metrics. This evaluation process is crucial for filtering out irrelevant or insignificant rules and retaining only the rules that exhibit significant associations between items.

For each rule, the support and confidence metrics are calculated by referencing the support map stored in the collocated hash map. During the evaluation process, any rule that does not meet the minimum support or confidence thresholds is considered not significant and is subsequently removed from further consideration. This filtering step ensures that only rules that possess sufficient support and demonstrate a strong association between the antecedent and consequent are retained. The rules that successfully pass the support and confidence criteria are accumulated in the final rules array, which serves as the output of the association rule mining process.

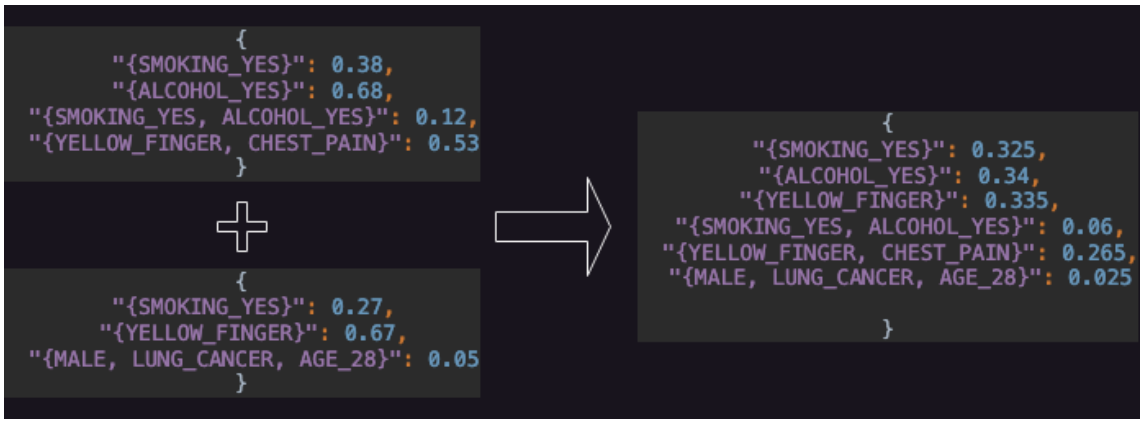

Figure 7. *Example of collocated hash map.*

3.2.4 Pros and cons

An examination of the theoretical pros and cons of the approach reveals potential advantages and disadvantages. From a theoretical perspective, this distributed approach holds promise in terms of significantly improving the speed of the Apriori algorithm compared to its non-distributed counterpart.

The advantage of improved speed arises from the ability to distribute the computational workload among multiple nodes, enabling concurrent execution of the Apriori algorithm on each block of data. This parallel processing capability has the potential to reduce the overall execution time, particularly when dealing with large datasets.

However, it is important to acknowledge that this approach's theoretical advantages may come at a cost in terms of algorithm accuracy, particularly when dealing with smaller datasets. By dividing the dataset into smaller blocks, there is a possibility that infrequent itemsets, which may still be relevant and significant in the overall dataset, could be missed.

3.3 Grouped by attributes dataset

This implementation is based on grouping some data rows by common attributes. The workflow of the algorithm is shown if Figure 8.

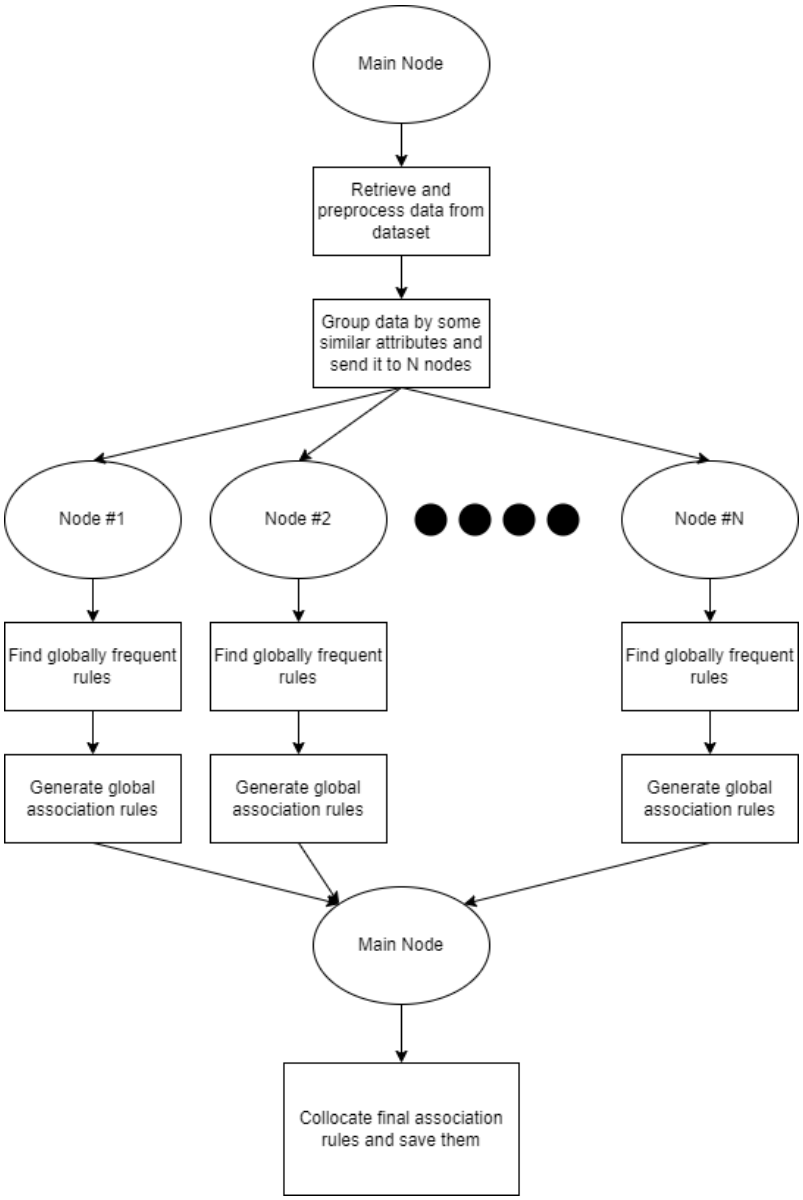

Figure 8. *Grouped by similar attributes approach.*

3.3.1 Grouping data

The grouping step represents a pivotal and intricate phase within this approach, significantly influencing the effectiveness and performance of the distributed algorithm. The careful and precise grouping of data into blocks based on a selected attribute is crucial to ensure that the resulting data blocks are approximately equal in size. This step requires meticulous consideration of several factors to achieve optimal grouping. But in this case it decreases dimensionality of dataset on each node and the size of dataset which causes very big increase in Apriori speed.

One crucial aspect is the careful selection of columns from which attribute values are chosen to form the basis for grouping the data. It is imperative to choose columns that do not include the target columns for association rule mining. Including target columns in the grouping process would result in transactions that do not consist items with those attributes, thereby distorting the calculation of support. Hence, it is essential to adhere to the principle that any column can be chosen for grouping except for the target columns.

The selection of appropriate attributes for grouping and the exclusion of target columns pose challenges for the practical application of this algorithm. Users must possess a clear understanding of the target columns and their significance within the dataset to ensure accurate and meaningful association rule mining. Consequently, the algorithm's applicability may be constrained if users are not adequately aware of the target columns or if such columns cannot be determined reliably.

Corresponding to Table 3 it would look like this in case we had 4 nodes that can do calculations. Tables 8, 9, 10, 11 are referred as an example of grouping dataset into blocks of data. This data is grouped by some general attributes like sex and age.

Table 8. *Grouped by male and age >= 55 dataset part 1.*

PID	**Smoking**	**Consuming Alcohol**	**Yellow Fingers**	**Shortness of Breath**	**Lung Cancer**
1	1	0	0	0	1
3	0	1	1	0	1
4	0	1	0	1	0
8	0	1	1	0	1
16	1	0	0	0	1

Table 9. *Grouped by male and age < 55 dataset part 2.*

PID	Smoking	Consuming Alcohol	Yellow Fingers	Shortness of Breath	Lung Cancer
6	1	0	0	0	1
9	0	1	0	1	0
12	0	1	1	0	1
13	0	1	0	1	0

Table 10. *Grouped by female and age >= 55 dataset part 3.*

PID	Smoking	Consuming Alcohol	Yellow Fingers	Shortness of Breath	Lung Cancer
5	1	1	0	1	1
14	1	1	0	1	1
15	1	1	0	1	0

Table 11. *Grouped by female and age < 55 dataset part 4.*

PID	Smoking	Consuming Alcohol	Yellow Fingers	Shortness of Breath	Lung Cancer
2	0	0	0	0	0
7	0	0	0	0	0
10	1	1	0	1	1
11	0	0	0	0	0

3.3.2 Apriori step

This step entails the execution of the Apriori algorithm, as elucidated in Chapter 2, on the datablocks derived from the preceding step. By applying the Apriori algorithm to each datablock, a set of association rules is generated. These rules, which capture the relationships and dependencies between items in the dataset, form the foundation for the subsequent step where they will be aggregated and consolidated.

3.3.3 Collocating result

This step basically takes all rules obtained in previous step and combine them all together to common array. Common array is final result of this approach that contains all mined rules.

3.3.4 Pros and cons

The most important advantage of this algorithm is speed which is increased because of decreasing size of the algorithm and its dimensionality, as one column is deleted from dataset, which brings high speed and lower memory load. Besides that it discovers more rare and detailed rules. They are more detailed because each node mines rules on dataset with predefined group of attributes and they added to final result. So for example if user wants to discover each rules with its dependence on age or sex he/she can use this approach and group data based on such wishes. In this case every rule will contain this column and will be very carefully described.

On the other hand, this approach has disadvantage that you can not group data on target columns and you should carefully think of which columns you want to use for grouping. It makes user to explore dataset before implementing algorithm to choose correct columns.

4. Proofs of Concepts

In the previous chapter two different prototypes were proposed for Apriori algorithm distribution. A method was described for every prototype.

In this chapter proofs of concepts for each of prototypes are implemented using Apollo Framework for distributing a workflow according to its graph representation and Python for preprocessing dataset, implementation of Apriori algorithm, collocating results and dividing it into blocks of data. For each of implementation AWS account was required as AWS Lambda was used for serverless functions deploying.

4.1 Dataset preprocessing

Dataset preprocessing is important because it improves the quality and suitability of the data for analysis, handles missing values and outliers, enables feature selection and dimensionality reduction, transforms and encodes variables, ensures compatibility with analysis techniques, reduces noise and biases, and enhances efficiency and performance during analysis. Ultimately, dataset preprocessing contributes to more accurate and reliable results, saves computational resources, and maintains the integrity of findings[6].

For the chosen dataset couple of techniques like data quality improvement and data normalization are used to improve dataset for the given algorithm. Other techniques as handling missing values, feature selection and dimensionality reduction, data transformation and encoding, noise reduction and outlier handling were not used as this dataset was full and did not require any of these to use[6].

CSV file[3] was parsed into JSON format as it is used for communication between nodes. Age column values were grouped into 8 different groups to reduce amount of possible itemsets and make it more convenient to combine with other attributes in Apriori. Illustration of plain dataset is shown in Figure 9. Illustration of preprocessed dataset is shown in Figure 10.

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_CANCER
0	0	69	1	2	2	1	1	2	1	2	2	2	2	2	2	1
1	0	74	2	1	1	1	2	2	2	1	1	1	2	2	2	1
2	1	59	1	1	1	2	1	2	1	2	1	2	2	1	2	2
3	0	63	2	2	2	1	1	1	1	1	2	1	1	2	2	2
4	1	63	1	2	1	1	1	1	1	2	1	2	2	1	1	2
5	1	75	1	2	1	1	2	2	2	2	1	2	2	1	1	1
6	0	52	2	1	1	1	1	2	1	2	2	2	2	1	2	1
7	1	51	2	2	2	2	1	2	2	1	1	1	2	2	1	1
8	1	68	2	1	2	1	1	2	1	1	1	1	1	1	1	2
9	0	53	2	2	2	2	2	1	2	1	2	1	1	2	2	1
10	1	61	2	2	2	2	2	2	1	2	1	2	2	2	1	1
11	0	72	1	1	1	1	2	2	2	2	2	2	2	1	2	1
12	1	60	2	1	1	1	1	2	1	1	1	1	2	1	1	2
13	0	58	2	1	1	1	1	2	2	2	2	2	2	1	2	1
14	0	69	2	1	1	1	1	1	2	2	2	2	1	1	2	2
15	1	48	1	2	2	2	2	2	2	2	1	2	2	2	1	1
16	0	75	2	1	1	1	2	1	2	2	2	2	2	1	2	1
17	0	57	2	2	2	2	2	1	1	1	2	1	1	2	2	1
18	1	68	2	2	2	2	2	2	1	1	1	2	2	1	1	1

Figure 9. *Plain dataset in CSV format.*


```
{"min_support": "0.05", "min_confidence": "0.51", "dataset_length": 55394, "transactions": [["MALE", "AGE_[66.3-87.0)", "YELLOW_FINGERS_YES", "ANXIETY_YES", "FATIGUE_YES",
"WHEEZING_YES", "ALCOHOL_CONSUMING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "SWALLOWING_DIFFICULTY_YES", "CHEST_PAIN_YES"], ["MALE", "AGE_[66.3-87.0)",
"SMOKING_YES", "CHRONIC_DISEASE_YES", "FATIGUE_YES", "ALLERGY_YES", "SHORTNESS_OF_BREATH_YES", "SWALLOWING_DIFFICULTY_YES", "CHEST_PAIN_YES"], ["FEMALE", "AGE_[52.5-59.4)",
"PEER_PRESSURE_YES", "FATIGUE_YES", "WHEEZING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "CHEST_PAIN_YES", "LUNG_CANCER_YES"], ["MALE", "AGE_[59.4-66.3)",
"SMOKING_YES", "YELLOW_FINGERS_YES", "ANXIETY_YES", "ALCOHOL_CONSUMING_YES", "SWALLOWING_DIFFICULTY_YES", "CHEST_PAIN_YES", "LUNG_CANCER_YES"], ["FEMALE", "AGE_[59.4-66.3)",
"YELLOW_FINGERS_YES", "WHEEZING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "LUNG_CANCER_YES"], ["FEMALE", "AGE_[66.3-87.0)", "YELLOW_FINGERS_YES",
"CHRONIC_DISEASE_YES", "FATIGUE_YES", "ALLERGY_YES", "WHEEZING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES"], ["MALE", "AGE_[45.6-52.5)", "SMOKING_YES", "FATIGUE_YES",
"WHEEZING_YES", "ALCOHOL_CONSUMING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "CHEST_PAIN_YES"], ["FEMALE", "AGE_[45.6-52.5)", "SMOKING_YES", "YELLOW_FINGERS_YES",
"ANXIETY_YES", "PEER_PRESSURE_YES", "FATIGUE_YES", "ALLERGY_YES", "SHORTNESS_OF_BREATH_YES", "SWALLOWING_DIFFICULTY_YES"], ["FEMALE", "AGE_[66.3-87.0)", "SMOKING_YES",
"ANXIETY_YES", "FATIGUE_YES", "LUNG_CANCER_YES"], ["MALE", "AGE_[52.5-59.4)", "SMOKING_YES", "YELLOW_FINGERS_YES", "ANXIETY_YES", "PEER_PRESSURE_YES", "CHRONIC_DISEASE_YES",
"ALLERGY_YES", "ALCOHOL_CONSUMING_YES", "SWALLOWING_DIFFICULTY_YES", "CHEST_PAIN_YES"], ["FEMALE", "AGE_[59.4-66.3)", "SMOKING_YES", "YELLOW_FINGERS_YES", "ANXIETY_YES",
"PEER_PRESSURE_YES", "CHRONIC_DISEASE_YES", "FATIGUE_YES", "WHEEZING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "SWALLOWING_DIFFICULTY_YES"], ["MALE", "AGE_[66.3-87.0)",
", "CHRONIC_DISEASE_YES", "FATIGUE_YES", "ALLERGY_YES", "WHEEZING_YES", "ALCOHOL_CONSUMING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "CHEST_PAIN_YES"], ["FEMALE",
"AGE_[59.4-66.3)", "SMOKING_YES", "FATIGUE_YES", "SHORTNESS_OF_BREATH_YES", "LUNG_CANCER_YES"], ["MALE", "AGE_[52.5-59.4)", "SMOKING_YES", "FATIGUE_YES", "ALLERGY_YES",
"WHEEZING_YES", "ALCOHOL_CONSUMING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "CHEST_PAIN_YES"], ["MALE", "AGE_[66.3-87.0)", "SMOKING_YES", "ALLERGY_YES",
"WHEEZING_YES", "ALCOHOL_CONSUMING_YES", "COUGHING_YES", "CHEST_PAIN_YES", "LUNG_CANCER_YES"], ["FEMALE", "AGE_[45.6-52.5)", "YELLOW_FINGERS_YES", "ANXIETY_YES",
"PEER_PRESSURE_YES", "CHRONIC_DISEASE_YES", "FATIGUE_YES", "ALLERGY_YES", "WHEEZING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES", "SWALLOWING_DIFFICULTY_YES"], ["MALE",
"AGE_[66.3-87.0)", "SMOKING_YES", "CHRONIC_DISEASE_YES", "ALLERGY_YES", "WHEEZING_YES", "ALCOHOL_CONSUMING_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES",
"CHEST_PAIN_YES"], ["MALE", "AGE_[52.5-59.4)", "SMOKING_YES", "YELLOW_FINGERS_YES", "ANXIETY_YES", "PEER_PRESSURE_YES", "CHRONIC_DISEASE_YES", "ALCOHOL_CONSUMING_YES",
"SWALLOWING_DIFFICULTY_YES", "CHEST_PAIN_YES"], ["FEMALE", "AGE_[66.3-87.0)", "SMOKING_YES", "YELLOW_FINGERS_YES", "ANXIETY_YES", "PEER_PRESSURE_YES", "CHRONIC_DISEASE_YES",
"FATIGUE_YES", "COUGHING_YES", "SHORTNESS_OF_BREATH_YES"], ["FEMALE", "AGE_[59.4-66.3)", "CHRONIC_DISEASE_YES", "FATIGUE_YES", "SHORTNESS_OF_BREATH_YES",
"LUNG_CANCER_YES"], ["FEMALE", "AGE_[38.7-45.6)", "SMOKING_YES", "YELLOW_FINGERS_YES", "ANXIETY_YES", "PEER_PRESSURE_YES", "CHRONIC_DISEASE_YES", "FATIGUE_YES",
```


Figure 10. *Preprocessed data in JSON format.*

4.2 Apollo Implementation in Python

In this thesis, the Apriori algorithm was implemented using the Python programming language to mine association rules from a given dataset. The implementation aimed to discover interesting patterns and relationships among items based on their co-occurrence frequencies.

The Apriori algorithm implementation followed the classic steps of candidate generation and pruning. Initially, the dataset was preprocessed to ensure its compatibility with the algorithm. This involved transforming the data into a suitable format, such as a transactional representation, which is a requirement for the Apriori algorithm.

The Python programming language provided a flexible and efficient environment for implementing the Apriori algorithm. The algorithm was implemented using custom Python functions and data structures.

The implementation began with generating the initial set of frequent itemsets with a minimum support threshold. The algorithm then iteratively generated higher-level candidate

itemsets and pruned those that did not meet the support threshold. This process continued until no further candidate itemsets could be generated.

To optimize the implementation, techniques such as storing itemsets support count in hash map, were utilized to efficiently access and update itemsets and their support counts. This helped improve the runtime performance, especially when dealing with larger datasets.

To evaluate the implemented Apriori algorithm, experiments were conducted using preprocessed dataset[3]. The performance of the algorithm was measured using built-in timer. Results are shown in Figure 11.

```
['{ANXIETY_YES, FEMALE, SHORTNESS_OF_BREATH_YES} => {LUNG_CANCER_YES}', 0.5121207722456089, 1.0322920584321262]
['{ANXIETY_YES, CHRONIC_DISEASE_YES, WHEEZING_YES} => {LUNG_CANCER_YES}', 0.5104375178724622, 1.028899089007939]
['{ANXIETY_YES, CHRONIC_DISEASE_YES, SHORTNESS_OF_BREATH_YES} => {LUNG_CANCER_YES}', 0.5131691955339249, 1.0344053861724916]
['{ANXIETY_YES, ALLERGY_YES, CHRONIC_DISEASE_YES} => {LUNG_CANCER_YES}', 0.5116345877621373, 1.0313120466684558]
['{SWALLOWING_DIFFICULTY_YES, CHRONIC_DISEASE_YES, CHEST_PAIN_YES} => {LUNG_CANCER_YES}', 0.5134358384825405, 1.0349428636840672]
135.76581406593323
```

Figure 11. *An illustration of local Apriori result.*

The generated association rules from the Apriori algorithm were evaluated based on various interestingness measures, including support and confidence. These metrics were used to identify meaningful and statistically significant rules from the dataset. The results were presented and discussed to provide insights into the relationships between items and their potential applications in different domains.

Overall, the implemented Apriori algorithm in Python proved to be effective in discovering association rules from the given datasets. The flexibility and efficiency of the Python programming language enabled the development of a robust implementation that could handle datasets of varying sizes. This implementation is going to be used in next prototypes implementations to mine association rules.

4.3 Single Node Apriori

This section is written to describe the simplest usage way of Apollo Framework and to compare results that are made on the same machine which is AWS Lambda server. The workflow is shown in Figure 12.

To implement this workflow in Apollo Framework three things are required: preprocessed data, AFCL model and Apollo implementation script in Python.

Preprocessing of data was described in Chapter 4.1 and Apollo Python script was described in Chapter 4.2. The left part is AFCL model. The tutorial for AFCL was found here[7].

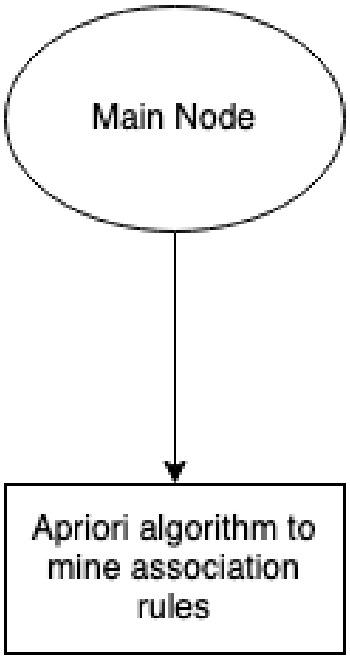

Figure 12. *Single node apriori AFCL model.*

Model was written using special IDE provided by Apollo[8]. The result is shown in Figure 13. According to this Figure to Run Single Node Apriori it is required to send to the node transactions list, minimum support metric and minimum confidence metric. Then after computation the Node will send rules back which are possible to explore.

```
Workflow: SingleApriori

  Input:
    transactions: Array = JSON(transactions)
    min_support: Number = JSON(min_support)
    min_confidence: Number = JSON(min_confidence)

  Body:
    Apriori():
      type: Apriori
      Input:
        transactions = SingleApriori.transactions
        min_support = SingleApriori.min_support
        min_confidence = SingleApriori.min_confidence
      Output:
        rules: Array


  Output:
    final_rules = Apriori.rules
```

Figure 13. *Single node Apriori AFCL model.*

4.4 Horizontally divided dataset prototype implementation

The first prototype that was implemented is "Horizontally divided dataset" which was described in Chapter 3. Basically it divides original dataset into approximately equal blocks of data. The workflow of this prototype was shown in Figure 5.

To implement this workflow in Apollo Framework five things are required: preprocessed data, AFCL model, Apollo implementation script in Python, Python script that divides data into block and Python script that combines rules and data received from serverless functions. Preprocessing of data was described in Chapter 4.1 and Apollo Python script was described in Chapter 4.2. For the implementation AFCL model and 2 more Python scripts are required.

Model was written using special IDE provided by Apollo[8]. The result is shown in Figure 14. According to this Figure to Divided Horizontally Apriori, it is required to send to the node transactions list, minimum support metric, minimum confidence metric and total size of dataset.

First step in this model is to divide data. For this goal separate function is created. This function name is "DivideData". On input it takes all dataset transaction and divide it into approximately equal by size blocks of data. On output it gives Array where each item is datablock that contains approximately same amount of data.

Second step in this model is to mine rules using Apriori. Here is used the same Apriori function that was written in Chapter 4.2. On the input it takes earlier created block of data, minimum support value and minimum confidence value. On output after computation it gives array that consists locally mined rules, JSON representation of hash map that contains support value for each itemset that was calculated and size of datablock which was used in this step. Apollo Framework combines all these results and add them into common arrays. So after all nodes are finished their work, all their results are located in array. On output we get array of arrays with rules, array of hash maps JSON representations and array of data block sizes.

Final step is to collocate all these results together. As was described in prototype here is required to combine all maps together and check every locally calculated rule to fit under chosen metrics like minimum confidence and minimum support. That is why on input function takes array of arrays with rules, array of serialized into JSON maps, array of datablock sizes, minimum support, minimum confidence and total size of original dataset. Firstly, it deserializes all maps and combines their values together. That is why we need sizes of blocks to restore original number of itemset in datablock and then sum them all up. Then each rule is evaluated using this common hash map and false ones are pruned and right ones are sent back on output.

4.5 Grouped by attributes dataset prototype implementation

The second prototype that was implemented is "Grouped by attributes" which was described in Chapter 3. Basically it groups data similar attributes group e.g. age or sex. The workflow of this prototype was shown in Figure 8.

To implement this workflow in Apollo Framework five things are required: preprocessed data, AFCL model, Apollo implementation script in Python, Python script that groups data by similar attributes group and Python script that combines rules and data received from serverless functions. Preprocessing of data was described in Chapter 4.1 and Apollo Python script was described in Chapter 4.2. For the implementation AFCL model and 2 more Python scripts are required.

Model was written using special IDE provided by Apollo[8]. The result is shown in Figure 15. According to this Figure to implement Grouped by attributes Apriori, it is required to send to the node transactions list, minimum support metric, minimum confidence metric, total size of dataset and all fields that can be present in itemset, it is sent as JSON representation of map where each key is a field and its value is array with possible values for this field.

First step in this model is to group data. For this goal separate function is created. This function name is "GroupData". On input it takes all dataset transactions and all possible attributes. Then it groups dataset by some field or fields. On output it gives Array where each item is datablock that contains approximately same amount of data and array with attributes where each element is corresponding to the element in array with groups, value in this case shows by which attribute this block of data was grouped. This function is the most complicated one in this approach because you should attribute in such a way to make blocks of data approximately equal by size. In this work was chosen to group data by age.

Second step in this model is to mine rules using Apriori. Here is used the same Apriori function that was written in Chapter 4.2. On the input it takes earlier created block of data, minimum support value, minimum confidence value, size of original dataset and group name which is value by which this block was grouped. On output after computation it gives array that consists locally mined rules and group name. Apollo Framework combines all these results and add them into common arrays. So after all nodes are finished their work, all their results are located in array. On output we get array of arrays with rules and array of group names.

Final step is to collocate all these results together. In this function it is not more than

deserialization of rules and adding group name to the left hand side rule to get final rule and then adding these all rules to the common array. On output it gives back all rules that were mined from each data block.

```
Input:
  transactions: Array = JSON(transactions)
  min_support: Number = JSON(min_support)
  min_confidence: Number = JSON(min_confidence)
  total_size: Number = JSON(dataset_length)

Body:
  DivideData():
    type: DivideData
    Input:
      transactions = DividedApriori.transactions
    Output:
      divided_data: Array

  forEachDataBlock = forEach:
    Input:
      iterator = DivideData.divided_data
    Body:
      DividedRulesMining():
        type: DividedRulesMining
        Input:
          transactions = forEachDataBlock.iterator
          min_support = DividedApriori.min_support
          min_confidence = DividedApriori.min_confidence
        Output:
          rules: Array
          support_map: String
          datablock_size: Number

    Output:
      rules_array = DividedRulesMining.rules
      maps_array = DividedRulesMining.support_map
      sizes_array = DividedRulesMining.datablock_size

  CollocateDividedResult():
    type: CollocateDividedResult
    Input:
      rules_array = forEachDataBlock.rules_array
      maps_array = forEachDataBlock.maps_array
      sizes_array = forEachDataBlock.sizes_array
      min_support = DividedApriori.min_support
      min_confidence = DividedApriori.min_confidence
      total_size = DividedApriori.total_size
    Output:
      rules: Array
```

Figure 14. *Horizontally divided dataset AFCL model.*

```
Input:
  transactions: Array = JSON(transactions)
  fields: String = JSON(fields)
  min_support: Number = JSON(min_support)
  min_confidence: Number = JSON(min_confidence)
  dataset_length: Number = JSON(dataset_length)

Body:
  GroupData():
    type: GroupData
    Input:
      transactions = GroupedApriori.transactions
      fields = GroupedApriori.fields
    Output:
      grouped_data: Array
      groups: Array

  forEachDataGroup = forEach:
    Input:
      iterator_data = GroupData.grouped_data
      iterator_name = GroupData.groups
    Body:
      GroupedRulesMining():
        type: GroupedRulesMining
        Input:
          transactions = forEachDataGroup.iterator_data
          group_name = forEachDataGroup.iterator_name
          min_support = GroupedApriori.min_support
          min_confidence = GroupedApriori.min_confidence
          size = GroupedApriori.dataset_length
        Output:
          rules_by_group: Array
          group_name: String

    Output:
      rules_array = GroupedRulesMining.rules_by_group
      group_names = GroupedRulesMining.group_name

  CollocateGroupedResult():
    type: CollocateGroupedResult
    Input:
      rules_array = forEachDataGroup.rules_array
      group_names = forEachDataGroup.group_names
    Output:
      rules: Array
```

Figure 15. *Grouped dataset AFCL model.*

5. Results and Comparison

The algorithms, which have been explained in detail in Chapter 4, were implemented and executed with the chosen dataset[3]. This step was essential to evaluate their effectiveness in achieving the research objectives. By running the algorithms on a local computer, a comprehensive comparison and analysis of their performance could be conducted. As target column was "Lung Cancer" column. Metrics that were used are 10% support and 50% confidence because otherwise nothing could be mined from this dataset. Concluding this fact was made a decision that chose dataset was auto generated using random generator. In general, it does not affect the result as rules from dataset are not primary goal of this work.

5.1 Setup Description

To execute the Apriori algorithm locally, a specific configuration was employed, which is summarized in Table 12.

To execute Apriori using Apollo Framework AWS Lambda was chosen as very popular serverless platform. AWS Lambda exhibits limited configurability, primarily allowing for adjustments to the allocated RAM memory size as the sole means of configuring computational power. Within the scope of this study, a RAM memory size of 2048 MB was selected, as AWS imposed restrictions on increasing it beyond this value. It is worth noting that the maximum allowable RAM size set by AWS is limited to 10240 MB.[9].

Table 12. *Local system configuration.*

CPU	**RAM**	**Operating system**	**Python**
2.6 GHz 6-Core Intel Core i7	16GB	macOS 12.4	3.8

5.2 Analysis of performance

Important note is that for distributed algorithm execution only first 40000 transactions of chosen dataset[3] were chosen because AWS has limitation for the size of request that can be sent and it is 6MB that is why not entire dataset could be sent[10]. One more fact is that AWS Gateway timeout is 30 seconds[11] for each endpoint and it is not configurable so it was important for Single Apriori to choose right size of dataset when its execution time would fit under 30 seconds. For that 20000 transactions were used.

Results are shown in Figure 16. For each algorithm 5 independent executions were performed and analyzed. Average of performed executions are shown in this Figure. Each step of algorithm was measured separately in addition to analyze their execution times.

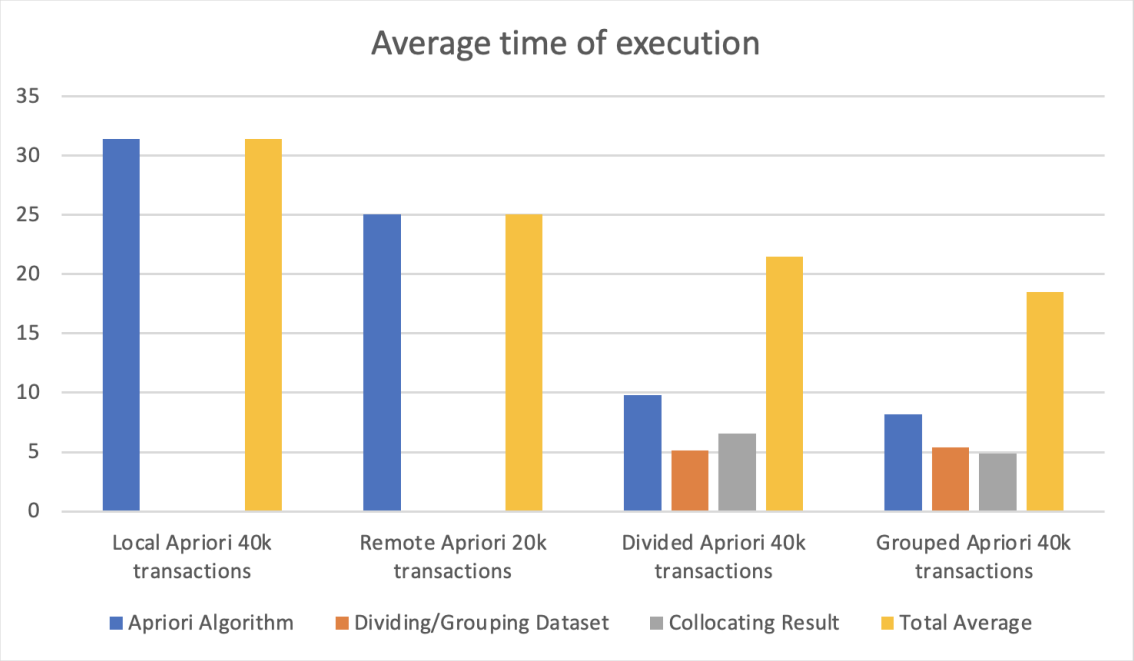

Figure 16. *Average time of algorithms execution.*

Note: In the Divided Apriori approach, the mining process involved the utilization of 8 nodes, with each node assigned an equal portion of the dataset for rule extraction. On the other hand, the Grouped Apriori approach also employed 8 nodes, but the data was grouped based on 8 distinct age gaps, ensuring that the groups were approximately equal in size (with a deviation of approximately ±10%).

5.2.1 Detailed comparison of Local Apriori with distributed ones

One crucial aspect to consider is the local execution time of the Apriori algorithm, which amounts to 31.49 seconds for 40000 transactions. This metric serves as a fundamental benchmark for comparison, as the primary objective of this thesis is to enhance the speed of Apriori.

From Figure 16, it is evident that the average time required for rule mining from 40000 transactions using the Divided approach is approximately 21.50 seconds. On the other hand, the average time for rule mining from the same dataset using the Grouping approach is 18.51 seconds. Both approaches outperform the local execution of Apriori, demonstrating that the distribution of workload contributes to improved performance. This signifies a notable 30% increase in speed.

These findings support the investigation of whether the proposed distribution methods effectively expedite the mining process. The comparison of execution times substantiates the hypothesis that the Divided and Grouping approaches facilitate faster rule mining in comparison to the traditional local execution of Apriori.

By presenting these comparative results, it is apparent that the proposed approaches provide tangible benefits in terms of computational efficiency. The observed improvements in speed not only validate the effectiveness of the distribution methods but also underline the potential for optimizing the Apriori algorithm in practice.

5.2.2 Detailed comparison of Serverless approaches between each other

In this section, three approaches, namely Single Apriori, Divided Apriori, and Grouped Apriori, are compared. Although their overall results show a similar completion time of around 20 seconds, a closer examination reveals notable differences between them.

One significant observation is that the Single Apriori approach running on AWS Lambda required 25 seconds to complete the mining process for 20000 transactions. On the other hand, the two distributed approaches achieved the same task 5 seconds faster. This clear discrepancy substantiates the assertion that distributed approaches are significantly faster in comparison.

When comparing the two distributed approaches, the Grouping approach proves to be 16% faster overall than the Divided approach. This difference warrants further investigation and analysis.

One crucial aspect to consider in the comparison of the Divided and Grouped Apriori approaches is the Apriori step itself. It is observed that the Grouped approach outperforms the Divided approach by 20% in terms of time, indicating that the Grouped approach mines association rules at a faster rate. This performance difference can be attributed to the variance in dataset dimensionality and the reduction in the number of attributes, such as Age, utilized in the Grouped approach.

When it comes to the dataset division and grouping steps, both approaches exhibit similar time requirements. The Divided approach is slightly faster, with a 5% advantage, as it solely divides the dataset and performs a single iteration over the transactions. Conversely, the Grouped approach involves both the grouping of data based on age groups and two iterations—one over the transactions and the other over the age groups. While the speed difference in this step is negligible, it is worth noting that both approaches efficiently distribute the workload across the nodes.

However, when it comes to the collocation of results, the Grouped approach proves to be faster. This step involves combining all the mined rules together. In the Grouped approach, this process is straightforward, while in the Divided approach, it requires deserializing rules and hash maps and checking each rule again to ensure they meet the confidence and support measures. The Grouped approach completes the collocation step 34% faster, showcasing its advantage in this aspect.

Considering the cumulative performance across these steps, the Grouped approach demonstrates a 16% overall improvement in terms of speed. This performance gain makes the Grouped approach a viable alternative and a promising solution for mining association rules in a distributed manner.

By carefully analyzing these performance differences and considering the specific requirements of the research objectives, it is possible to make an informed decision on which approach to adopt for efficient and timely rule mining.

5.3 Analysis of algorithms accuracy

The comparison between the Divided approach and the Grouped Apriori approach becomes more nuanced when considering the theoretical basis and the accuracy of the results. The Divided approach, which achieves 100% accuracy, ensures that all mined rules are identical to those obtained from the local Apriori execution.

On the other hand, the rules mined in Grouped Apriori approach are based on a distinct dataset. When comparing the Grouped approach with the local Apriori, it is found that 60% of the rules mined by the Grouped Apriori are identical to those obtained through local execution. However, the remaining 40% of rules are more detailed and provide a deeper understanding of the dataset.

Therefore, the Grouped Apriori approach not only mines a greater number of rules but also offers a more comprehensive and detailed view of the dataset compared to the local Apriori execution. This aspect highlights the advantage of the Grouped approach in terms of efficiency and the level of insight it provides.

6. Summary

The primary objective of this study was to enhance the speed of association rule mining through the implementation of a distributed approach based on the Apriori algorithm. To accomplish this goal, two distinct algorithm prototypes were developed. Python was chosen as the main programming language for implementation, and the Apollo Framework was utilized as the runtime environment. Through rigorous testing and comprehensive analysis of the results, it can be confidently asserted that the primary objective has been achieved. Both prototypes exhibit a 30% improvement in performance compared to the local execution of the Apriori algorithm.

The algorithms presented in this study have demonstrated their efficacy and applicability for association rule mining in real-world scenario. Each approach possesses its own set of advantages and limitations. The Divided approach provides a simpler implementation, removing the need for intricate data grouping and collocation steps. On the other hand, the Grouped approach necessitates careful consideration of attribute selection for data grouping, aiming to achieve approximately equal-sized data blocks. It should be noted that the Grouped approach is constrained by the requirement to exclude the target columns from the grouping process.

Furthermore, there are opportunities for further advancements in this area. Potential avenues for future research include the exploration of distributed hash tables for improved data management, the utilization of alternative serverless platforms, the investigation of alternative data grouping techniques, and the development of algorithms to facilitate efficient pruning of data on remote machines through periodic data synchronization.

In conclusion, this work has successfully achieved its main objective of enhancing the speed of association rule mining through a distributed approach. The implemented algorithms have proven their effectiveness and can be applied in practical scenarios. While both approaches have their merits, the choice of implementation depends on specific requirements and considerations. Future research should focus on refining and expanding upon the existing methodologies to unlock further advancements in the field of distributed association rule mining.

References

[1] Yang-Jun GAO Huan-Bin WANG. *Research on parallelization of Apriori algorithm in association rule mining*. [Accessed: 20-03-2023]. URL: https://www.sciencedirect.com/science/article/pii/S1877050921005858.

[2] Thomas Fahringer Fedor Smirnov Behnaz Pourmohseni. *Apollo: Modular and Distributed Runtime System for Serverless Function Compositions on Cloud, Edge, and IoT Resources*. [Accessed: 20-02-2023]. URL: https://dl.acm.org/doi/abs/10.1145/3452413.3464793.

[3] *Lung cancer dataset*. [Accessed: 10-03-2023]. URL: https://www.kaggle.com/datasets/h13380436001/h-lung-cancer?select=survey+lung+cancer.csv.

[4] *Association rule*. [Accessed: 12-04-2023]. URL: http://ethen8181.github.io/machine-learning/association_rule/apriori.html.

[5] *Association Analysis: Basic Concepts and Algorithms*. [Accessed: 12-04-2023]. URL: https://www-users.cse.umn.edu/~kumar001/dmbook/ch6.pdf.

[6] *Data Preprocessing: 6 Techniques to Clean Data*. [Accessed: 15-05-2023]. URL: https://www.scalablepath.com/data-science/data-preprocessing-phase.

[7] *AFCL 1.1*. [Accessed: 15-05-2023]. URL: https://apollowf.github.io/learn.html.

[8] *AFCL Editor*. [Accessed: 15-05-2023]. URL: https://github.com/Apollo-AFCL/AFCLEditor/blob/main/docs/tutorial.md.

[9] *Memory and computing power*. [Accessed: 21-05-2023]. URL: https://docs.aws.amazon.com/lambda/latest/operatorguide/computing-power.html.

[10] *Lambda quotas*. [Accessed: 21-05-2023]. URL: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html.

[11] *Amazon API Gateway quotas and important notes*. [Accessed: 21-05-2023]. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Kirill Timofejev

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Distribution of Apriori Algorithm for Lung Cancer Data Set Using Apollo Framework", supervised by Mahtab Shahin
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

22.05.2023

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.