


# Parallel and Distributed Pattern Mining

Ishak H.A Meddah, Labri Laboratory, École Supérieure en Informatique, Sidi Bel Abbès, Algeria

 <https://orcid.org/0000-0003-4164-7487>

Nour El Houda REMIL, Mascara University “Mustapha STAMBOULI”, Mascara, Algeria

## ABSTRACT

The treatment of large data is difficult and it looks like the arrival of the framework MapReduce is a solution of this problem. This framework can be used to analyze and process vast amounts of data. This happens by distributing the computational work across a cluster of virtual servers running in a cloud or a large set of machines. Process mining provides an important bridge between data mining and business process analysis. Its techniques allow for extracting information from event logs. Generally, there are two steps in process mining, correlation definition or discovery and the inference or composition. First of all, their work mines small patterns from log traces. Those patterns are the representation of the traces execution from a log file of a business process. In this step, the authors use existing techniques. The patterns are represented by finite state automaton or their regular expression; and the final model is the combination of only two types of different patterns whom are represented by the regular expressions  $(ab)^*$  and  $(ab^*c)^*$ . Second, they compute these patterns in parallel, and then combine those small patterns using the Hadoop framework. They have two steps; the first is the Map Step through which they mine patterns from execution traces, and the second one is the combination of these small patterns as a reduce step. The results show that their approach is scalable, general and precise. It minimizes the execution time by the use of the Hadoop framework.

## KEYWORDS

Hadoop, MapReduce, Process, Trace

## 1. INTRODUCTION

Many techniques have been proposed that mine such patterns from execution traces. However, the most existing techniques mine only simple patterns, or a single complex pattern that is restricted to a particular set of manually selected events.

Recent work has recognized that patterns can be specified as regular languages (Ammons, Bodík, & Larus, 2002). This allows the compact representation of patterns as regular expressions or finite state automata, and it allows the characterization of the pattern mining as a language learning problem.

Current approaches are fundamentally similar; each takes as input a static program or a dynamic traces or profile and produces one or more compact regular languages that specify the pattern representation or the workflow. However, the individual solutions differ in key ways.

In this paper, we present a new general approach to patterns mining that addresses several of the limitations of current techniques. Our insight is twofold. First, we recognize that instances of smaller patterns which can be composed in parallel into larger patterns. Second, we also observe that the composition of small pattern can be in parallel.

DOI: 10.4018/IJRSDA.2019070101

This article published as an Open Access Article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

Then, we leverage this insight to divide our work into two parts; The first one, we use a technique on how we can mine two types of small patterns and we compose them by using standard algorithms for finite state automaton manipulation, and some special rules used by M. Gabel and Z. Su (2008), The mining is also performed by symbolic mining algorithm (Gabel & Su, 2008; Zhang, Zhu, Chen, & Yang, 2015).

For the second one, we use the framework MapReduce in mining and composing micro-patterns. Those patterns have been shown as regular expressions or their finite state automata. In this part we mine small patterns using the same symbolic mining algorithm but in parallel as Map step, and we compute this small pattern into a larger pattern in parallel as a reduce step.

Our work is an amelioration to the existing techniques in mining patterns and parallel distributed process.

Our approach has been implemented in the java programming language with two log files of two applications namely; SKYPE and VIBER. The size of the first log file is 10 GB, and the second is 18 GB, which are generated by log file generator.

We have tested our approach in three clusters in a cloud, the first regroups five machines, the second regroups ten machines, and the third regroups 20 machines. The traces in our applications are the call, the answer, and the messages, etc.

## **2. RELATED WORK**

Many techniques are suggested in the domain of process mining; we quote:

M. Gabel et al. (2008) present a new general technique for mining temporal specification, they realized their work in two steps, firstly they discovered the simple patterns using existing techniques, then combine these patterns using the composition and some rules like Branching and Sequencing rules.

Temporal specification expresses formal correctness requirement of an application's ordering of specific actions and events during execution, they discovered patterns from traces of execution or program source code; The simple patterns are represented using regular expression  $(ab)^*$  or  $(ab^*c)^*$  and their representation using finite state automaton, after they combine simple patterns to construct a temporal specification using a finite state automaton.

G. Greco et al. (2006) discovered several clusters by using a clustering technique, and then they calculate the pattern from each cluster, they combine these patterns to construct a final model, they discovered a workflow scheme from, and then they mine a workflow using a mine workflow algorithm, after they define many clusters from a log traces by using clustering technique and process discover algorithm and some rules cluster.

Then they use a find features algorithm to find patterns of each cluster, finally they combine these patterns to construct a completely hierarchical workflow model.

In their clustering algorithm, clusters reflect only structural similarities among traces; they say that in future works extending their techniques to take care of the environment so that clusters may reflect not only structural similarities among traces, but also information about, e.g., users and data values.

H.R. Motahari-Nezhed et al. (2008) use a service conversation log; first they split a log into several partitions, second they discovered a model from each partition, and third, they annotate the discover protocol model with various metadata to construct a protocol model from real-word service conversation logs.

The protocol is the specification of all possible conversations that a service can have with its partners and the conversation consists of a sequence of messages exchanged between two or more services.

During the split they discovered a simple precise protocol models by analyzing messages sequences in the log, they eliminate conversations considered noisy or not presented in the log; they

discovered a protocol with various metadata including state and transition supports to get a final protocol model of the log a most generalized model based splitting.

Li Li, Xiangfeng Luo, and Haiyan Chen (2015) Proposed two clustering-based student grouping methods which automatically group students for all kinds of organizational learning, they say that they are flexible, comprehensive, and timely compared with manual group methods;

Their general grouping method consist to divide students into groups by their teacher manually, which is not timely or accurate. They say that overcome the shortcomings of manual methods, their paper proposes an automatic grouping method based on clustering technologies. In first time, “the student profile is built to model the student’s knowledge level, which can be updated based on the results of examinations automatically. Then, to meet the different teaching goals, two student clustering methods are proposed: similarity student clustering and complementation student clustering” (Li, Luo, & Chen, 2015). Last, their proposed methods are evaluated by comparing the students of clustered groups with those of the manual groups in the learning effectiveness. Their results show that their methods are flexible, comprehensive, and timely compared with manual group methods. Knowledge level, which can be updated based on the results of examinations automatically. Then, to meet the different teaching goals, two student clustering methods are proposed: similarity student clustering and complementation student clustering. At last the proposed methods are evaluated by comparing students of clustered groups with those of the manual groups in the learning effectiveness (Meddah & Belkadi, 2018)

“Mining frequent patterns and finding associations among them require handling large and distributed databases. As FP-tree considered being the best compact data structure to hold the data patterns in memory there has been efforts to make it parallel and distributed to handle large databases” (Itkar & Kulkarni, 2013). However, it incurs lot of communication over head during the mining. Parallel and distributed frequent pattern mining algorithm using Hadoop Map Reduce framework is proposed, which shows best performance results for large databases. Proposed algorithm partitions the database in such a way that, it works independently at each local node and locally generates the frequent patterns by sharing the global frequent pattern header table. These local frequent patterns are merged at final stage (Itkar & Kulkarni, 2013).

This reduces the complete communication overhead during structure construction as well as during pattern mining. The item set count is also taken into consideration reducing processor idle time. Hadoop Map Reduce framework is used effectively in all the steps of the algorithm. Their experiments are carried out on a PC cluster with 5 computing nodes which shows execution time efficiency as compared to other algorithms. Their experimental result shows that proposed algorithm efficiently handles the scalability for very large data bases (Itkar & Kulkarni, 2013; Deng & Lv, 2015; Meddah & Belkadi, 2018).

Ranjan and Bhatnagar (2010) present the advantages of the application of data mining techniques in the management of client relationship in the financial sectors like banking, forecasting stock market, currency exchange rate and bank bankruptcies, say they; there are a big concurrence in the society financial sectors, and they find a difficult to sustain the ever-changing behavior of the customer.

“Data mining techniques is helping the firms to achieve profitable and efficient CRM by providing them with advance techniques to analyze the already existing data in the databases of the firms using the complex modeling algorithms” (Ranjan & Bhatnagar, 2010); their work demonstrated that data mining is able to automate the process of searching to mountain of customer’s related data to find patterns that are good predictors of the behaviors of the clients.

Authors propose an idea of how data mining provide the increased customer, and minimize the risk involved in the financial sectors to obtain the advantages and conclude by providing the limitations of opportunities in the field.

Processing and analysis of big data can be problematic, it refers to data is so large that it exceeds the processing capabilities of traditional systems, it can be awkward work and the storage (Hsu, Slagter, & Chung, 2015).

*MapReduce is a recent programming model that can handle big data. MapReduce achieves this by distributing the storage and processing of data amongst a large number of computers (nodes). However, this means the time required to process a MapReduce job is dependent on whichever node is last to complete a task. Heterogeneous environments exacerbate this problem (Hsu, Slagter, & Chung, 2015).*

Ching-Hsien Hsu et al. propose a method to improve MapReduce execution in heterogeneous environments.

Their work is done by dynamically partitioning data before the Map phase and by using virtual machine mapping in the Reduce phase in order to maximize resource utilization. Their Simulation and experimental results show an improvement in MapReduce performance, including data locality and total completion time with different optimization approaches.

“Big Data refers to the massive amounts of structured and unstructured data being produced every day from a wide range of sources. Big Data is difficult to work with and needs a large number of machines to process it, as well as software capable of running in a distributed environment” (Slagter, Hsu, & Chung, 2015). “MapReduce is a recent programming model that simplifies writing distributed programs on distributed systems. For MapReduce to work it needs to divide work amongst computers in a network. Consequently, the performance of MapReduce is dependent on how evenly it distributes the workload” (Hsu, Slagter, & Chung, 2015; Slagter, Hsu, & Chung, 2015). Authors (Slagter, Hsu, & Chung, 2015) propose an adaptive sampling mechanism for total order partitioning that can reduce memory consumption whilst partitioning with a trie-based sampling mechanism (ATrie). The performance of the proposed algorithm (Slagter, Hsu, & Chung, 2015) is compared to a state-of-the-art trie-based partitioning system (ETrie). Their experiments show the proposed mechanism is more adaptive and more memory efficient than previous implementations. Since ATrie is adaptive, its performance depended on the type of data used. A performance evaluation of a 2-level ATrie shows it uses 2.43 times less memory for case insensitive email addresses, and uses 1,024 times less memory for birthdates compared to that of a 2-level ETrie. Their results show the potential of the proposed method.

*MapReduce is an effective tool for processing large amounts of data in parallel using a cluster of processors or computers. One common data processing task is the join operation, which combines two or more datasets based on values common to each (Slagter, Hsu, Chung, & Yi, 2014).*

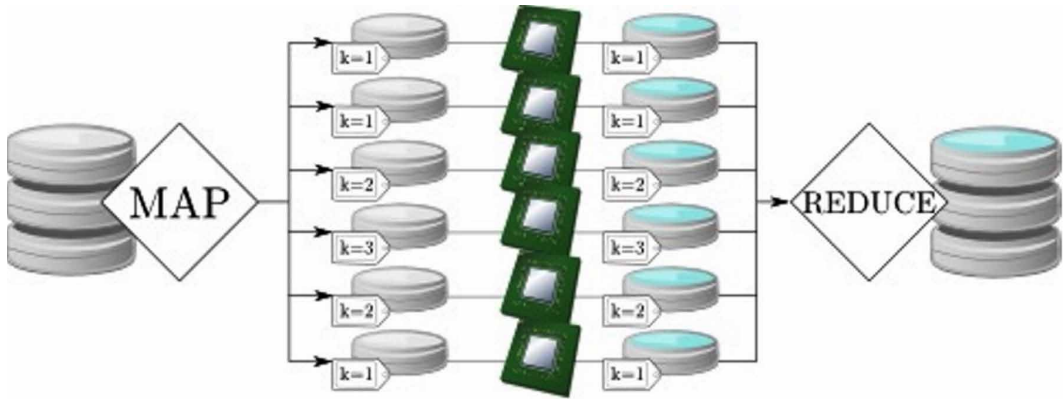
Kenn Slagter et al. present a network aware multi-way join for MapReduce (SmartJoin) that improve performance and considers network traffic when redistributing workload amongst reducers. SmartJoin achieves this by dynamically redistributing tuples directly between reducers with an intelligent network aware algorithm. They show that their technique has significant potential to minimize the time required to join multiple datasets. Their evaluation, they show that SmartJoin has up to 39% improvement compared to the non-redistribution method, a 26.8% improvement over random redistribution and 27.6% improvement over worst join redistribution.

### 3. MAPREDUCE

Since its introduction just a few years ago, the MapReduce framework (Dean & Ghemawat, 2008; Itkar & Kulkarni, 2013; Tsourakakis, 2010) has become extremely popular for analyzing large datasets in cluster environments. The success of MapReduce stems from hiding the details of parallelization, fault tolerance, and load balancing in a simple programming framework.

The MapReduce Framework is increasingly being used to analyze large volume of data. Having several small processes, meaning these small processes or patterns can be computed in parallel.

Figure 1. Principal of MapReduce (Dean & Ghemawat, 2008)

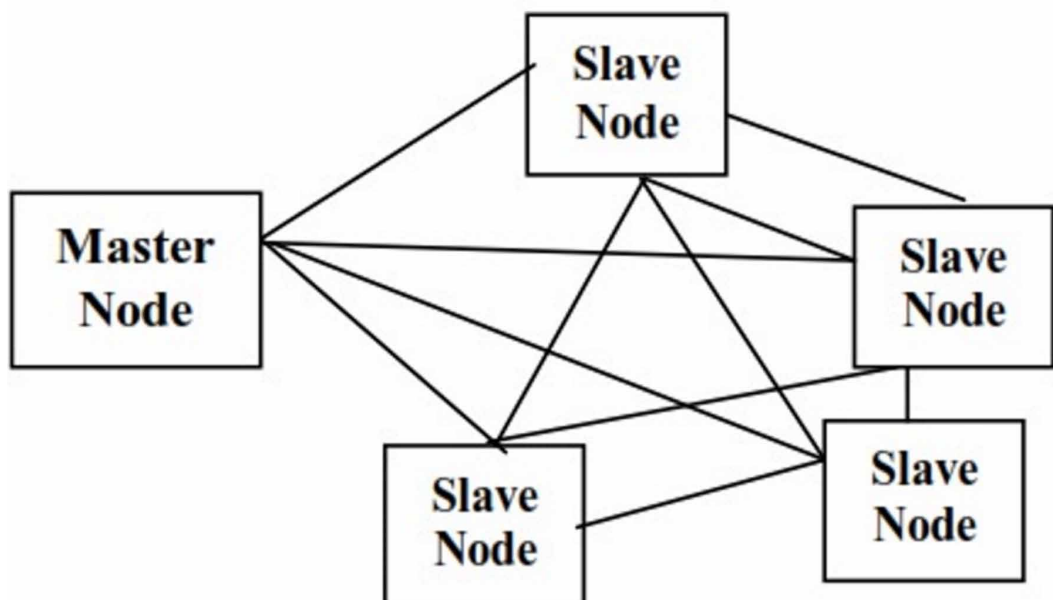


The MapReduce framework is implemented in the C programmer language, there are an open sources implementation called Hadoop, it uses a distributed processing architecture in which a task is mapped to a set of servers in then reduced down a single output set, one node, designed as the master node, controls the distribution of tasks. The following diagram shows a Hadoop cluster with the master node directing a group of slave nodes, which process the data.

MapReduce (Dean & Ghemawat, 2008; Itkar & Kulkarni, 2013; Bhuiyan & Al Hasan, 2015; Tsourakakis, 2010; Lan, Wang, Fong, Liu, Wong, & Dey, 2018) was originally proposed and used by Google engineers to process the large amount of data they must analyze on a daily basis.

The input data for MapReduce consists of a list of key/value pairs. Mappers accept the incoming pairs and map them into intermediate key/value pairs. Each group of intermediate data with the same key is then passed to a specific set of reducers, each of which performs computations on the data

Figure 2. Master node directing a group of slave nodes (Shang, Jiang, Adams, & Hassan, 2009)

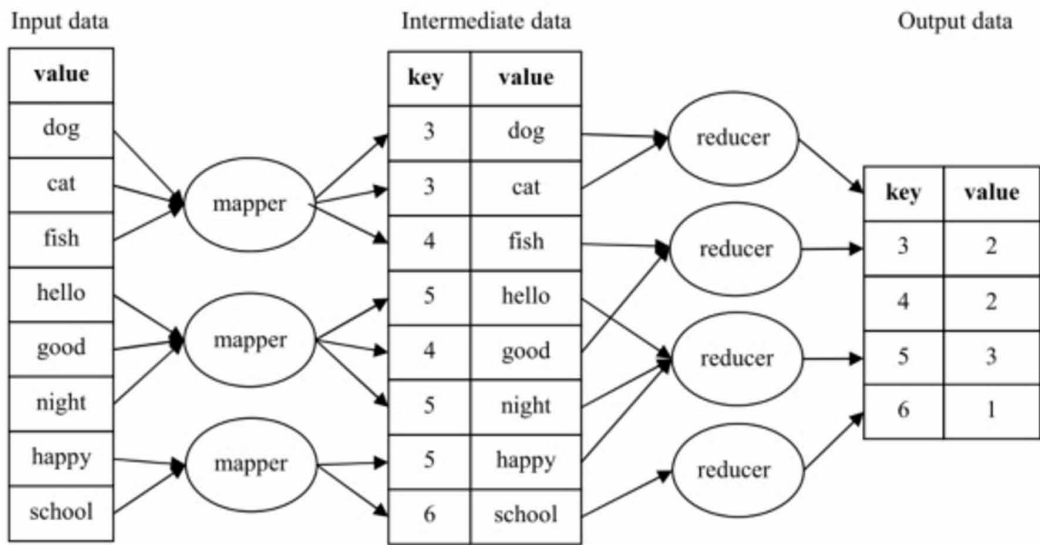


and reduce it to one single key/values pair. The sorted output of the reducers is the final result of the MapReduce process.

To illustrate MapReduce, we consider an example MapReduce process, which counts the frequency of word lengths in a book. The example process is shown in Figure 3. Mappers accept every single word from the book and make keys for them. Because we want to count the frequency of all words with different length, a typical approach would be to use the length of the word as key. So, for the word “hello”, a mapper will generate a key/value pair of “5/hello”. Afterwards, the key/value pairs with the same key are grouped and sent to reducers. A reducer, which receives a list of values with the same key, can simply count the size of this list, and keep the key in its output. If a reducer receives a list with key “5”, for example, it will count the size of the list of all the words that have as length “5”, if the size is “n”, it outputs an output pair “5/n” which means there are “n” words with length “5” in the book.

The power and challenge of the MapReduce (Shang, Jiang, Adams, & Hassan, 2009; Tsourakakis, 2010; White, Yeh, Lin & Davis, 2010; Chaki, Dey, Shi, & Sherratt, 2019) model reside in its ability to support different mapping and reducing strategies. For example, an alternative mapper implementation could map each input value (i.e, word) based on its first letter and its length. Then the reducers would process those words starting with one or a small number of different letters (keys) and perform the counting. This MapReduce strategy permits an increasing number of reducers that can work in parallel on the problem. However; the final output needs additional post-processing in the comparison to the first strategy. In short, both strategies can solve the problem, but each strategy has different performance and implementation benefits and challenges.

Figure 3. Example MapReduce process for counting the frequency of word lengths in a book (Shang, Jiang, Adams, & Hassan, 2009)



The open source implementation of MapReduce” Hadoop” (Stewart, Trinder, & Loidl, 2011; Itkar & Kulkarni, 2013; Bhuiyan & Al Hasan, 2015) is supported by yahoo and used by Amazon, AOL, Baidu and a number of other companies for their distributed solutions. Hadoop can run on various operating systems such as Linux, Windows, FreeBSD, Mac OSX and OpenSolaris. It doesn’t only implement the MapReduce model, but also provides a distributed file system, called the Hadoop

Distributed File System (HDFS). Hadoop supplies java interfaces to simplify the MapReduce model and to control the HDF programmatically. Another advantage for users is that Hadoop by default comes with some basic and widely used mapping and reducing methods, for example to split files into lines, or to split a directory into files. With these methods, users occasionally do not have to write new code to use MapReduce. We used Hadoop as our MapReduce implementation for the following four reasons:

### **3.1. Hadoop is Easy to Use**

Researchers do not want to spend considerable time on modifying their mining program to make it distributed. The simple MapReduce java interface simplifies the process of implementing the mappers and reducers (Tsourakakis, 2010; Koundinya, Sharma, Kumar, & Shanbag, 2012).

### **3.2. Hadoop Runs on Different Operating Systems**

Academic research labs tend to have a heterogeneous network of machines with different hardware configurations and varying operating systems. Hadoop can run on most current operating systems and hence to exploit as much of the available computing power as possible (Koundinya, Sharma, Kumar, & Shanbag, 2012; Meddah & Khaled, 2016).

### **3.3. Hadoop Runs on Commodity Machines**

The largest computation resources in research labs and software development companies are desktop computers and laptops. This characteristic of Hadoop permits these computers to join and leave the computing cluster in a dynamic and transparent fashion without user intervention (Shang, Jiang, Adams, & Hassan, 2009; Seki, K., Jinno & Uehara, 2013; Koundinya, Sharma, Kumar, & Shanbag, 2012)

### **3.4. Hadoop is Mature and an Open Source Systems**

Hadoop has been successfully used in many commercial projects. It is actively developed with new features and enhancements continuously being added. Since Hadoop is free to download and redistribute, it can be installed on multiple machines without worrying about costs and per seat licensing.

Based on these points, we consider Hadoop as the most suitable MapReduce implementation for our research.

## **4. PROCESS MINING**

Recently, process mining has become a vivid research area (Van der Aalst & Weijters, 2004; Van der Aalst, Weijters, & Maruster, 2004). The basic idea of process mining is to diagnose business processes by mining event logs for knowledge. Process mining techniques and tools provide the means for discovering process, control, data, organizational, and social structures from event logs (Van der Aalst, 2004; Meddah & Belkadi, 2018).

It provides an important bridge between data mining and business process analysis (Weske, 2012; Gao, 2013; De Weerd, Schupp, Vanderloock, & Baesens, 2013; Karaa & Dey, 2017), and even allow for extracting information from event logs.

The idea is that process mining is:

Process Discovery: What processes are executed in our company, supported by enterprise information systems (ERP, BPM, total ad-hoc, e-mail).

Conformance checking: Business processes are executed according to the rules defined, or human variants exist.

Performance analysis: Where are the bottlenecks?

Process prediction: When will the process end?

Process improvement: How to redesign a process?

For example, the audit trails of a workflow management system or the transaction logs of an enterprise resource planning system can be used to discover models describing processes, organizations, and products. Moreover, it is possible to use process mining to monitor deviations (e.g., comparing the observed events with predefined models or business rules in the context of SOX).” (Versteeg & Bouwman, 2006; Zhang, Zhu, Chen, & Yang, 2015).

In addition, Process mining is interesting (Versteeg & Bouwman, 2006):

In enterprise architecture, when analysts and people who work in your company lost time going fishing for processes that exist, in order to establish process and system architecture. Process Mining plays an indispensable role in the discovery of true enterprise architecture.

Process Conformity, how many times people discovered that the processes are not performed according to the rules (our human nature love finding new ways to execute). This does not mean that the process should be executed according to the rules, because sometimes the rules were not correctly set up.

Process optimization: People that has the experience to perform process analysis by looking at process flows usually indicate easily where are the bottleneck, duplication, repetition, but nowadays (!) in the world of knowledge management where flows do not dictate the manner of execution is necessary to sit side by side with the people who perform work to understand what are the obstacles (in a large company this is a daunting task). But it is also true that normally escapes analysis teams some of the problem sources, or because there are based opinions, or simply ... bad reasoning (Meddah, Belkadi, & Boudia, 2016).

Business Intelligence helps to understand how we do things but does have predictive capabilities needed to understand how work could be performed.

## 4.1 Business Process

Business process is a set of activities occurring within a company that lead to a specific end. Most often, it focuses on meeting the needs of the customer and delivering a good or service that will fulfill that need. This process is actually often a collection of interrelated processes that function in a logical sequence to achieve the ultimate goal (Meddah & Belkadi, 2018; De Weerd, Schupp, Vanderloock, & Baesens, 2013; Zhang, Zhu, Chen, & Yang, 2015).

The log files correspond to the actions or traces of the business process. The following example, which is very simple, represents a sequence of characters:

1x Case1 A B C D E G

1x Case2 A C B D G

1x Case3 A E D

Simple Log File

A= Send e-mail, B= check credit, C= calculate capacity, D= check system, E= accept, F= reject, G=send e-mail.

There are many techniques to discover micro patterns from event log or process traces, the next section describes how we mine small patterns from business process.

## 5. IMPLEMENTATION AND RESULTS

### 5.1 Implementation and Experimental Setup

We implemented our approach in the java programming language. The first phase, which consist of mining micro patterns, is performed by an existing symbolic specification-mining algorithm (Gabel & Su, 2008; Itkar & Kulkarni, 2013; Tsourakakis, 2010; Meddah & Belkadi, 2017; Van der Aalst, 2015). This algorithm leverages Binary Decision Diagrams (Bryant, 1986; Seki, Jinno, & Uehara,



2013; Ranjan & Bhatnagar, 2010) to maintain a compact state throughout its execution, despite simultaneously tracking up to billions of potential micro patterns. This algorithm is currently the most scalable pattern-based approach, and it is the only algorithm capable of scalably mining micro patterns with alphabets of size three (Meddah & Belkadi, 2018; Wang, 2014; Wang, & Wiebe, 2014).

Our Hadoop installation is divided into three steps; first is deployed on five computers in a local network. The five computers have an Intel Core I5 3210M @ 2.5 GHz CPU with 8 GB Ram memory. The second phase is deployed on ten computers. The third step is deployed on twenty computers. In the second and third steps, the computers have the same characteristics of the five computers.

Our small patterns were represented by using regular expression or their finite state automaton, to compose these small patterns, we use the standard algorithm for finite state automaton, using the following rules:

**Definition (Projection).** The projection  $\pi$  of a string  $s$  over an alphabet  $\Sigma$ ,  $\pi_{\Sigma}(s)$ , is defined as  $s$  with all letters not in  $\Sigma$  deleted.

The projection of a language  $L$  over  $\Sigma$  is defined as  $\pi_{\Sigma}(L) = \{\pi_{\Sigma}(s) \mid s \in L\}$ .

**Definition (Specification Pattern).** A specification pattern is a finite state automaton  $A = (Q, \Sigma, \delta, q_0, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is a set of input symbols,  $\delta: Q \times \Sigma \rightarrow Q$  is the transition function,  $q_0$  is the single starting state, and  $F$  is a set of final states. A pattern is satisfied over a trace  $T$  with alphabet  $\Sigma' \supseteq \Sigma$  if  $\pi_{\Sigma}(T) \in L(A)$ .

**Definition (Expansion)** (Gabel and Su, 2008, §2.6). Assume a regular language defined by finite state automaton  $A = (Q, \Sigma, \delta, q_0, F)$ . The expansion of  $(L(A))$  over an arbitrary alphabet  $\Sigma'$ , written  $E_{\Sigma'}(L(A))$ , is the maximal language over  $\Sigma \cup \Sigma'$  whose projection over  $\Sigma$  is  $L(A)$ .

An automaton accepting  $E_{\Sigma'}(L(A))$  can be constructed by first duplicating  $A$  and then adding a looping transition  $\delta(q, a) = q$  to each state  $q$  for each letter  $a \in \Sigma' \setminus \Sigma$ .

Expansion can be thought of maximal inverse of projection. For example, an expression corresponding to  $E_{\{a,b,c\}}((a b)^*)$  is  $c^*(a c^* b c^*)^*$ .

Note that projecting this new language over  $\{a, b\}$  yields the original language,  $(a b)^*$ . The composition of two patterns is defined as follows:

**Definition (Composition).** The composition of two patterns  $A_1$  and  $A_2$  is the intersection of the expansion of each pattern over their combined alphabets,  $E_{\Sigma_1}(A_1) \cap E_{\Sigma_2}(A_2)$ .

Intuitively, the composition of two patterns defines a language of traces in which both patterns hold.

We could use this general definition to arbitrary compose patterns by using standard algorithms for finite state automaton manipulation. However, in general, performing these pairwise compositions directly are undesirable. Given a reasonably large set of patterns, the finite state expansion, intersection, and minimization operations become more expensive as the automata grow.

There is some difficulty of treatment with the use of those rules, however, we use some other rules used by Gabel and Su (2008) in their framework Javert.

They recognize special cases of composition in which the result of composition is compact and intuitive. Then they formulate these cases as inference rules, which leads to straightforward implementation in which composition is a constant time operation.

They suggest two rules: the branching rule and the sequencing rule:

**Branching Rule** describes the composition of two patterns with identical “endpoint”, i.e., the first and the last letters of a single iteration of the pattern. Defining  $\Sigma'$  as  $\{a, b\} \cup \Sigma_{L_1} \cup \Sigma_{L_2}$ , the correctness of the branching rule follows:

$$E_{\Sigma'}(aL_1^*b)^* \cap E_{\Sigma'}(aL_2^*b)^* = (a(L_1|L_2)^*b)^* \quad (\text{Gabel and Su, 2008, §3.5})$$

This rule performs the composition, of two patterns that describe legal operations at the same logical state. For example, from the patterns:  $[\text{Call answer}^* \text{Close}]^*$ , and  $[\text{Call not answer}^* \text{Close}]^*$

We can infer a third pattern  $[\text{Call (answer | not answer}^* \text{Close)}]^*$

**Sequencing Rule** describes the sequencing of two patterns with compatible endpoints. as with the previous rule,  $L_1$  and  $L_2$  must have disjoint alphabets, which must in turn be disjoint from  $\{a, b, c\}$ . Redefining  $\Sigma'$  as  $\{a, b, c\} \cup \sum_{L_1} \cup \sum_{L_2}$ , the correctness of the sequencing rule follows from the following fact:

$$E_{\Sigma'}(a L_1 b)^* \cap E_{\Sigma'}(b L_2 c)^* \cap E_{\Sigma'}(a c)^* = (a L_1 b L_2 c)^* \text{ (Gabel and Su, 2008, §3.6)}$$

Continuing the earlier example, from the patterns:

[Call (answer | not answer)\* Close]\*, [Connect Call]\*, and [Connect close]\*

We can infer a fourth pattern [Connect Call (answer | not answer)\* Close]\*

Both of these rules are general; they apply to both micro patterns and any intermediate assembly thereof.

For the second party, we computed and composed the small patterns in parallel with the use of MapReduce framework.

In the MapReduce implementation, we have two steps: the Map step and the Reduce step; we consider the mining Patterns as the Map step, and compose patterns as the Reduce step.

As an input we have log file of two applications, in this log file we have all applications traces, from those traces we mine in parallel small patterns, having  $\langle P, \text{Value} \rangle$ , Patterns as the key and their values. In the Map step we mine all possible patterns, there are two types of patterns  $(ab)^*$  and  $(ab^*c)^*$ ;  $P$  is a small pattern and value is the number of patterns, in this step we have many cases of  $\langle P, \text{Value} \rangle$ . In the reduce step we have only the result of patterns  $\langle P1, \text{Value} \rangle$  and  $\langle P2, \text{Value} \rangle$ , after we compose them in parallel in order to get the final pattern that represents the whole model of the process, this model generates a lot of cases, Unfortunately there are two cases can't be generated by our approach.

Our algorithm is an amelioration of existing techniques by using MapReduce:

Repeat the process for all application

```
Input
Process Traces or the log files of Applications
Mapper
Pattern 1, value
..... /*Pattern 1= (ab)* */
Pattern 2, value
..... /*Pattern 2= (ab*c)* */
Reducer
Pattern 1, Values
Pattern 2, Values
Composition
Compose the patterns using standard Algorithms of finite state
automaton, and the sequencing, branching rules
Output
```

The workflow how represents the actions of users for three web applications.

Figure 4 represents our MapReduce approach; P1 and P2 are the small mining patterns.

The next section presents our result in all steps.

## 5.2 Results

In first time we used a log file of size 10 GB; Table 1 presents the result of our approach without using MapReduce; we used a log file of size 10 GB of two Applications Skype and Viber. The number of

Figure 4. The approach architecture using MapReduce

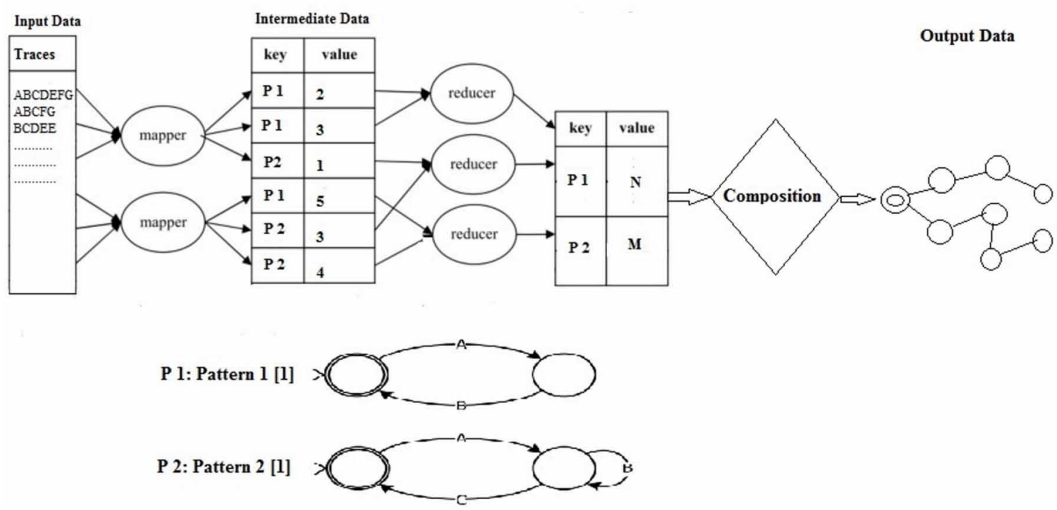


Table 1. Trace data and analysis times

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	825,458,970	5432,8s	462,0s
Viber	8	332,697,321	3951,0s	589,4s

traces is the actions effected, and total trace events are the number of all actions contained in the log file, we know that we have two steps; the pattern mining and the composition of patterns whom are executed in limited times.

Table 2 presents the result of our approach using MapReduce distributed in five (05) computers with the same applications and the log file size.

Table 2. Trace data and analysis times with using MapReduce in five machines

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	825,458,970	2092,5s	298,0s
Viber	8	332,697,321	1901,0s	222,1s

Table 3 presents the result of our approach using MapReduce distributed in ten (10) computers with the same applications and the log file size.

**Table 3. Trace data and analysis times with using MapReduce in ten machines**

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	825,458,970	1431,3s	153,0s
Viber	8	332,697,321	1052,8s	143,7s

Table 4 presents the result of our approach using MapReduce distributed in twenty (20) computers with the same applications and the log file size.

**Table 4. Trace data and analysis times with using MapReduce in twenty machines**

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	825,458,970	802,9s	99,8s
Viber	8	332,697,321	709,0s	82,3s

In second time we used the same clusters with log file of size 18 GB;

Table 5 presents the result of our approach without using MapReduce; we used a log file of size 18 GB of two Applications Skype and Viber. The number of traces is the actions effected, and total trace events are the number of all actions contained in the log file, we know that we have two steps; the pattern mining and the composition of patterns whom are executed in limited times.

**Table 5. Trace data and analysis times**

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	935,478,970	7442,8s	982,0s
Viber	8	424,707,321	5951,0s	789,4s

Table 6 presents the result of our approach using MapReduce distributed in five (05) computers with the same applications and the log file of size 18GB.

**Table 6. Trace data and analysis times with using MapReduce in five machines**

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	935,478,970	4201,7s	541,8s
Viber	8	424,707,321	3151,8s	498,7s

Table 7 presents the result of our approach using MapReduce distributed in ten (10) computers with the same applications and the log file of size 18GB.

**Table 7. Trace data and analysis times with using MapReduce in ten machines**

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	935,478,970	2092,5s	368,0s
Viber	8	424,707,321	1701,0s	302,1s

Table 8 presents the result of our approach using MapReduce distributed in twenty (20) computers with the same applications and the log file of size 18GB.

**Table 8. Trace data and analysis times with using MapReduce in twenty machines**

Application	Num. of Traces	Total Trace Events	Execution Time	
			Pattern Mining	Composition
Skype	11	935,478,970	999,9s	128,7s
Viber	8	424,707,321	896,9s	131,1s

## 6. CONCLUSION

A scalable pattern mining solution should be efficient, scalable. In this paper, we propose to use MapReduce as a general framework to mining micro patterns from process traces. To validate our approach, we presented our experience of mining small patterns and compose them. Our experiments demonstrate that our solution minimizes the execution time, and we conclude that the parallel compute have an inverse relationship with the execution time, with the grow of the machines, the time execution decreases. In addition to the existence of a proportional relationship between the grow of computers and the efficiency of treatment.

There are a number of directions for future work, including and evaluating our method in a big number of computers in cloud, also for big log file size of different applications.

## **ACKNOWLEDGMENT**

We would like to thank the anonymous reviewers for their insightful comments and helpful suggestions.

## REFERENCES

- Gabel, M., & Su, Z. (2008, November). Javert: fully automatic mining of general temporal properties from dynamic traces. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (pp. 339-349). ACM. doi:10.1145/1453101.1453150
- Greco, G., Guzzo, A., Pontieri, L., & Sacca, D. (2006). Discovering expressive process models by clustering log traces. *Knowledge and Data Engineering. IEEE Transactions on*, 18(8), 1010–1027.
- Motahari-Nezhad, H. R., Saint-Paul, R., Benatallah, B., & Casati, F. (2008). Deriving protocol models from imperfect service conversation logs. *IEEE Transactions on Knowledge and Data Engineering*, 20(12), 1683–1698.
- Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107–113. doi:10.1145/1327452.1327492
- Gabel, M., & Su, Z. (2008, May). Symbolic mining of temporal specifications. In *Proceedings of the 30th international conference on Software engineering* (pp. 51-60). ACM.
- Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *Computers. IEEE Transactions on*, 100(8), 677–691.
- van der Aalst, W. (2004). Discovering coordination patterns using process mining. In *Proceedings of the Workshop on Petri Nets and Coordination* (pp. 49-64). Academic Press.
- Van der Aalst, W. M., & Weijters, A. J. M. M. (2004). Process mining: A research agenda. *Computers in Industry*, 53(3), 231–244. doi:10.1016/j.compind.2003.10.001
- Van der Aalst, W., Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9), 1128–1142.
- Ammons, G., Bodik, R., & Larus, J. R. (2002). Mining specifications. *ACM SIGPLAN Notices*, 37(1), 4–16. doi:10.1145/565816.503275
- Shang, W., Jiang, Z. M., Adams, B., & Hassan, A. E. (2009). *Mapreduce as a general framework to support research in mining software repositories*. MSR.
- Stewart, R. J., Trinder, P. W., & Loidl, H. W. (2011). Comparing high level mapreduce query languages. In *Advanced Parallel Processing Technologies* (pp. 58–72). Springer Berlin Heidelberg. doi:10.1007/978-3-642-24151-2\_5
- Weske, M. (2012). *Business process management: concepts, languages, architectures*. Springer Science & Business Media.
- Gao, X. (2013). Towards the next generation intelligent BPM—in the era of big data. In *Business Process Management* (pp. 4–9). Springer Berlin Heidelberg.
- Versteeg, G., & Bouwman, H. (2006). Business architecture: A new paradigm to relate business strategy to ICT. *Information Systems Frontiers*, 8(2), 91–102. doi:10.1007/s10796-006-7973-z
- Meddah, I. H., Belkadi, K., & Boudia, M. A. (2016). Parallel Mining Small Patterns from Business Process Traces. *International Journal of Software Science and Computational Intelligence*, 8(1), 32–45. doi:10.4018/IJSSCI.2016010103
- Meddah, I. H., & Belkadi, K. (2018). Efficient Implementation of Hadoop MapReduce-Based Dataflow. In *Handbook of Research on Biomimicry in Information Retrieval and Knowledge Management* (pp. 372–385). Hershey, PA: IGI Global. doi:10.4018/978-1-5225-3004-6.ch020
- De Weerd, J., Schupp, A., Vanderloock, A., & Baesens, B. (2013). Process Mining for the multi-faceted analysis of business processes—A case study in a financial services organization. *Computers in Industry*, 64(1), 57–67. doi:10.1016/j.compind.2012.09.010
- Itkar, S. A., & Kulkarni, U. V. (2013). Distributed Algorithm for Frequent Pattern Mining using HadoopMap Reduce Framework. In *Proc. of Int. Conf. on Advances in Computer Science, AETACS*. Academic Press.

- Bhuiyan, M. A., & Al Hasan, M. (2015). An iterative MapReduce based frequent subgraph mining algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 27(3), 608–620.
- Tsourakakis, C. E. (2010). *Data Mining with MAPREDUCE: Graph and Tensor Algorithms with Applications* [Doctoral dissertation]. Carnegie Mellon University.
- Seki, K., Jinno, R., & Uehara, K. (2013). Parallel distributed trajectory pattern mining using hierarchical grid with MapReduce. *International Journal of Grid and High Performance Computing*, 5(4), 79–96. doi:10.4018/ijghpc.2013100106
- Koundinya, A. K., Sharma, K. A. K., Kumar, K., & Shanbag, K. U. (2012). Map/Reduce Design and Implementation of Apriori Algorithm for handling voluminous data-sets.
- Braun, P., Cameron, J. J., Cuzzocrea, A., Jiang, F., & Leung, C. K. (2014). Effectively and efficiently mining frequent patterns from dense graph streams on disk. *Procedia Computer Science*, 35, 338–347. doi:10.1016/j.procs.2014.08.114
- Deng, Z. H., & Lv, S. L. (2015). PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning. *Expert Systems with Applications*, 42(13), 5424–5432. doi:10.1016/j.eswa.2015.03.004
- Meddah, I., & Khaled, B. (2016). Discovering Patterns using Process Mining. *International Journal of Rough Sets and Data Analysis*, 3(4), 21–31. doi:10.4018/IJRSDA.2016100102
- White, B., Yeh, T., Lin, J., & Davis, L. (2010, July). Web-scale computer vision using mapreduce for multimedia data mining. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining* (p. 9). ACM. doi:10.1145/1814245.1814254
- Meddah, I. H., & Belkadi, K. (2017). Parallel distributed patterns mining using hadoop mapreduce framework. *International Journal of Grid and High Performance Computing*, 9(2), 70–85. doi:10.4018/IJGHPC.2017040105
- van der Aalst, W. M. (2015). Extracting event data from databases to unleash process mining. In *BPM-Driving innovation in a digital world* (pp. 105–128). Springer International Publishing. doi:10.1007/978-3-319-14430-6\_8
- Li, L., Luo, X., & Chen, H. (2015). Clustering Students for Group-Based Learning in Foreign Language Learning. *International Journal of Cognitive Informatics and Natural Intelligence*, 9(2), 55–72. doi:10.4018/IJCINI.2015040104
- Ranjan, J., & Bhatnagar, V. (2010). Application of data mining techniques in the financial sector for profitable customer relationship management. *International Journal of Information and Communication Technology*, 2(4), 342–354. doi:10.1504/IJICT.2010.034976
- Zhang, S., Zhu, G., Chen, H., & Yang, D. (2015). Mining Conflict Semantic from Drug Dataset for Detecting Drug Conflict. *International Journal of Cognitive Informatics and Natural Intelligence*, 9(3), 87–104. doi:10.4018/IJCINI.2015070105
- Meddah, I. H., & Belkadi, K. (2018). Mining Patterns Using Business Process Management. In *Handbook of Research on Biomimicry in Information Retrieval and Knowledge Management* (pp. 78–89). Hershey, PA: IGI Global. doi:10.4018/978-1-5225-3004-6.ch005
- Meddah, I. H., & Belkadi, K. (2018). Efficient Implementation of Hadoop MapReduce-Based Dataflow. In *Handbook of Research on Biomimicry in Information Retrieval and Knowledge Management* (pp. 372–385). Hershey, PA: IGI Global. doi:10.4018/978-1-5225-3004-6.ch020
- Wang, Y. (2014). Fuzzy causal patterns of humor and jokes for cognitive and affective computing. *International Journal of Cognitive Informatics and Natural Intelligence*, 8(2), 34–46. doi:10.4018/IJCINI.2014040103
- Wang, Y., & Wiebe, V. J. (2014). Big Data Analytics on the Characteristic Equilibrium of Collective Opinions in Social Networks. *International Journal of Cognitive Informatics and Natural Intelligence*, 8(3), 29–44. doi:10.4018/IJCINI.2014070103
- Hsu, C. H., Slagter, K. D., & Chung, Y. C. (2015). Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications. *Future Generation Computer Systems*, 53, 43–54. doi:10.1016/j.future.2015.04.006



- Slagter, K., Hsu, C. H., & Chung, Y. C. (2015). An adaptive and memory efficient sampling mechanism for partitioning in MapReduce. *International Journal of Parallel Programming*, 43(3), 489–507. doi:10.1007/s10766-013-0288-z
- Slagter, K., Hsu, C. H., Chung, Y. C., & Yi, G. (2014). SmartJoin: A network-aware multiway join for MapReduce. *Cluster Computing*, 17(3), 629–641. doi:10.1007/s10586-014-0348-1
- Lan, K., Wang, D. T., Fong, S., Liu, L. S., Wong, K. K., & Dey, N. (2018). A survey of data mining and deep learning in bioinformatics. *Journal of Medical Systems*, 42(8), 139. doi:10.1007/s10916-018-1003-9 PMID:29956014
- Chaki, J., Dey, N., Shi, F., & Sherratt, R. S. (2019). Pattern mining approaches used in sensor-based biometric recognition: A review. *IEEE Sensors Journal*, 19(10), 3569–3580. doi:10.1109/JSEN.2019.2894972
- Karaa, W. B. A., & Dey, N. (2017). *Mining multimedia documents*. Chapman and Hall/CRC.

*Ishak H.A Meddah is an associate professor at ESI-SBA (École Supérieure en Informatique, SBA-Algeria). Ishak was received his master's degree from USTO-MB in 2011 and he received a PhD of computer science from the same university. Ishak has published more than 10 papers in national and international conference also international journals from 2011 to 2018. His research activities concern knowledge extraction, data mining, parallel and distributed systems, fog and cloud computing, and information retrieval.*