Assessing Process Discovery Scalability in Data Intensive Environments

Sergio Hernández, Joaquín Ezpeleta
Department of Computer Science and Systems Engineering
University of Zaragoza
Zaragoza, Spain
Email: {shernandez, ezpeleta}@unizar.es

S.J. van Zelst, Wil M.P. van der Aalst
Department of Mathematics and Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
Email: {s.j.v.zelst, w.m.p.v.d.aalst}@tue.nl

***Abstract*—Tremendous developments in Information Technology (IT) have enabled us to store and process huge amounts of data at unprecedented rates. This impacts business processes in various ways. Throughput times of processes decrease while the data stored associated to the execution of these processes increases. The field of process discovery, originating from the area of process mining, is concerned with automatically discovering process models from event data related to the execution of business processes. In this paper, we assess the scalability of applying process discovery techniques in data intensive environments. We propose ways to compute parts of the internal data abstractions used by the discovery techniques within the MapReduce framework. The combination of MapReduce and process discovery enables us to tackle much bigger event logs in less time. Our generic approach scales linearly in terms of the size of the data and the number of computational resources used, and thus, shows great potential for the adoption of process discovery in a Big Data context.**

***Keywords*-Process mining; MapReduce; Hadoop; Big Data; Automated process discovery; Scalability; ProM**

I. Introduction

The amount of data being recorded has increased tremendously over the last few years. Nowadays, the total amount of data generated within a few minutes equals the total amount of data generated between the Stone Age and 2003. Moreover, the rate of data generation keeps increasing [1]. The main challenge related to this data explosion phenomenon is not the storage, but the extraction of valuable information from these huge collections of data [2].

The latter challenge also applies to the analysis of business processes. Multiple *events* related to the execution of business processes are recorded everyday including the activities performed, the resource executing the activity, the associated cost, etc. A collection of such events is referred to as an *event log*. A thorough analysis of an event log can lead to improvements in the operational process under study, e.g., causing increases in revenue, system reliability, process quality, client satisfaction, etc. Enabling the aforementioned improvements is the main goal of *process mining* [3].

Process mining includes three types of analysis. *Process discovery*, the most prominent process mining task, is concerned with automatically extracting a process model from an event log. *Conformance checking* measures how well the behaviour recorded in an event log fits a given process model and vice versa. *Process enhancement* is concerned with improving an existing model using information about the actual process recorded in an event log.

The spectacular growth of event data poses new challenges for process mining techniques. The majority of process mining techniques are not designed to scale, i.e., they assume all data to be in local memory and do not apply distributed computation schemes across several computers [4]. These assumptions are reflected in the poor handling of large amounts of data by tools used for process mining, e.g., the ProM framework[1], Disco[2], etc. Hence, current process mining techniques fail when being applied to large amounts of data.

Some approaches to tackle the scalability issues in process mining have been proposed, e.g., the application of *divide and conquer* schemes for the purpose of problem decomposition [5]. Additionally a first implementation of process discovery algorithms using the MapReduce [6] framework was proposed [7]. However, an exhaustive study of the possible benefits of the use of distributed computing techniques in the context of process mining is still missing.

MapReduce is of special interest as it concerns a scalable distributed computing paradigm suitable for data-intensive applications [8]. Apache Hadoop[3] is the most popular open source implementation supporting MapReduce. Additionally, Hadoop provides a distributed file system (HDFS) suitable for storing data at a large-scale. Thus, Hadoop provides a bridge between large amounts of event data, typically stored in a company's data warehouse, and the MapReduce paradigm.

The main goal of this paper is to assess the scalability and viability of distributed process discovery approaches. For this purpose, we have designed a distributed approach for large-scale process discovery based on MapReduce. The approach starts with partitioning the input event log and computing different data abstractions suitable to be used by different process discovery techniques. Then, the process

[1] http://www.promtools.org
[2] http://fluxicon.com/disco/
[3] http://hadoop.apache.org/

model calculated from these abstraction is discovered and visualized within the ProM framework, the most widely used process mining tool, taking advantage of the algorithms and methods already available in the tool. Thus, the integration of these technologies enables researchers and practitioners in the process mining community to use and develop distributed process mining techniques and provides the opportunity of improving the performance and scalability of existing techniques.

A comprehensive assessment of the scalability of the developed approach is carried out by performing several experiments varying different dimensions like event log size, computational resources used, and event log characteristics. The results obtained from the conducted experiments prove the scalability of the solution and the importance of the specific log in the final execution time. The results demonstrate the viability of using this type of techniques in the process mining field. Thus, the analysis of vast amounts of data is possible in a reasonable time.

The remainder of this paper is organized as follows. Section II provides an overview of the process mining field, MapReduce and Hadoop. In Section III the MapReduce-based approach for large-scale distributed process discovery is explained. Section IV presents the evaluation of the approach using several datasets and discusses the results. In Section V related work is discussed. Finally, Section VI summarizes and concludes the paper.

II. Preliminaries

A. Event Logs and Process Models

We present a brief overview of the key concepts within the field of process mining. For a more detailed overview of the field and associated concepts we refer to [3].

The typical input for process mining is an *event log*. An *event log* corresponds to a collection of *traces*, referring to a specific execution of the business process, i.e. a *process instance* or *case*. Each trace is an ordered sequence of events. Each *event* represents an execution of an *activity* that is part of the process in the context of a case. In addition, other information about events can also be recorded and used for further analysis. For instance, the *resource* that is executing the activity or the *timestamp* of the event are attributes of interest that can be recorded with an event. As an example of an event log consider Table I, adopted from [3].

The event log describes two cases of a (fictional) process related to handling compensation requests. The two cases depicted in the table are identified by the *Case-id* column, i.e. *355410* and *355411*. Various activities are performed within the process, i.e. *register request*, *check ticket*, etc., and various resources are active within the business process, i.e. *Alfred*, *Lucy*, etc.

Techniques for process discovery return process models learned from event data. A process model describes the possible *flow of execution* of a business process. An example of a business process model, using BPMN [9] as process modelling language, able to describe the two traces presented in Table I is depicted in Figure 1.

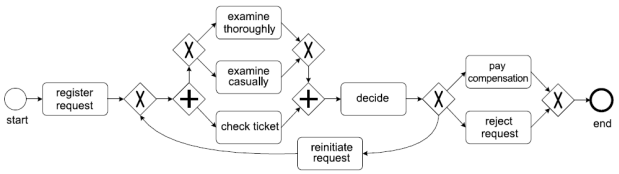


Figure 1: BPMN model adopted from [3], describing the two traces presented in Table I.

The process model allows for different traces of events as well. In fact, due to the loop structure (i.e. executing *reinitiate request*), the process model describes an infinite number of traces.

B. Process Discovery Techniques

The main goal of process discovery is to construct a process model, based on event data stored in an event log. An abundance of process discovery techniques exist. We refer to [10] for an overview. The solution proposed in this paper is tailored towards three specific process discovery techniques which we describe in more detail, i.e., the *Alpha Miner* [11], the *Inductive Miner* [12] and the *Flexible Heuristics Miner* [13]. These three techniques are representative for a wider range of process discovery techniques. The conceptual computational structures of the algorithms indicating the data relations are depicted in Figure 2.

Alpha Miner: The Alpha Miner consists of three stages as shown in Figure 2a. First, direct succession information of the activities in the event log is recorded in a *directly-follows graph* (`stage 1`). Direct succession of activity a

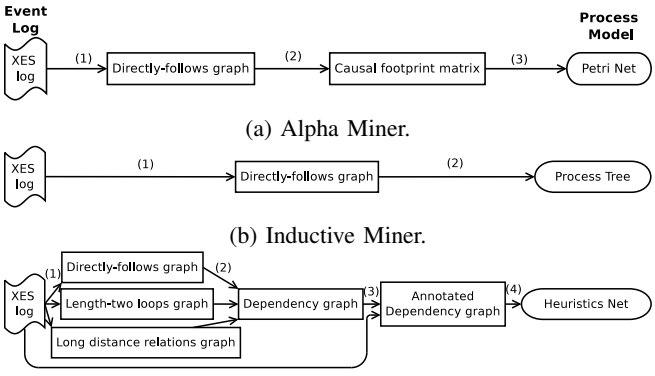
Case-id	Activity	Resource	Time-stamp
⋮	⋮	⋮	⋮
355410	*register request*	*Alfred*	*2015-07-08T08:45:37+00:00*
355410	*examine thoroughly*	*Lucy*	*2015-07-09T09:13:37+00:00*
355410	*check ticket*	*Alfred*	*2015-07-15T09:14:25+00:00*
355410	*decide*	*Ralph*	*2015-07-18T10:11:15+00:00*
355410	*reject request*	*Ralph*	*2015-07-19T10:28:18+00:00*
355411	*register request*	*Lucy*	*2015-07-08T10:33:37+00:00*
355411	*examine casually*	*Rob*	*2015-07-10T10:43:56+00:00*
355411	*check ticket*	*Ralph*	*2015-07-12T11:00:28+00:00*
355411	*decide*	*Rob*	*2015-07-15T11:14:04+00:00*
355411	*reinitiate request*	*Rob*	*2015-07-15T11:35:17+00:00*
355411	*examine thoroughly*	*Lucy*	*2015-07-18T11:55:38+00:00*
355411	*check ticket*	*Alfred*	*2015-07-22T11:57:46+00:00*
355411	*decide*	*Alfred*	*2015-07-22T12:20:32+00:00*
355411	*pay compensation*	*Lucy*	*2015-07-25T12:26:56+00:00*
⋮	⋮	⋮	⋮

Table I: A fragment of a fictional event log adopted from [3]. Each line corresponds to an event.

by some activity b means that in some trace, activity a is immediately followed by activity b. For example in Table I, *register request* is directly followed by *examine thoroughly*. The directly-follows graph used by the Alpha Miner is unweighted, i.e., it only records whether a direct succession occurs within the event log or not. Next, a *causal footprint matrix* is calculated using the directly-follows graph as input `(stage 2)`. The matrix indicates for any two activities if they are executed in parallel, sequentially or whether they never follow each other directly. Finally, a process model representing is constructed by the α-algorithm [11] based on the casual footprint matrix `(stage 3)`. The Alpha Miner uses *Petri Nets* as a representation.

Inductive Miner: An overview of the stages carried out by the Inductive Miner is presented in Figure 2b. The Inductive Miner approach also converts the event log to a directly-follows graph `(stage 1)`. The technique is however able to apply a filtering scheme on a weighted-directly follows graph. The weighted variant explicitly captures the frequency of a direct succession. From the weighted directly-follows graph, a process tree is discovered by considering the frequencies of each relation `(stage 2)` [12]. The process tree can be converted later to a Petri net or a BPMN model.

Flexible Heuristics Miner: The structure of the Flexible Heuristics Miner is shown in Figure 2c. In the first stage, three graphs are calculated `(stage 1)`. The first graph contains information about direct succession relations represented by a *directly-follows graph*, including *length-one loops*. The second graph represents *length-two loop* relations within the event log. Finally, the third graph represents *long distance relations*, i.e, the number of times each activity is indirectly followed by another activity. Using these three graphs, the algorithm builds a *dependency graph* that for each activity indicates the possible preceding and succeeding activities `(stage 2)`. Then, the dependency graph is annotated by detecting the splits and joins of each activity



(a) Alpha Miner.

(b) Inductive Miner.

(c) Flexible Heuristics Miner.

Figure 2: Structure of the different stages carried out in the process discovery techniques under study.

`(stage 3)`. XOR/AND/OR-splits/joins can be discovered by considering subsets of predecessor and successor activities. To do so, the event log is revisited considering the previously calculated dependency graph and detecting all the activators and reactors of each activity in every trace. This step is computationally expensive and the execution time highly depends on the event log characteristics [13]. Finally, a *heuristics net* model that visualizes the annotated dependency graph is built `(stage 4)`.

C. MapReduce and Apache Hadoop

MapReduce [6] is an easy-to-use programming model introduced by Google inc. to process large datasets on clusters of servers in an scalable manner. It is based on the use of two functions, inspired from the functional programming paradigm: `map` and `reduce`. The user needs to define these two functions using *key-value* pairs to express both the input and the output of the computation. The input of the *map* phase is a list of key-value pairs, where the input key represents the position of the data in the input file and the input value is the actual data. Its output is a list of *intermediate* key-value pairs. The output is grouped by key and all the intermediate values associated with the same key are provided together to the *reduce* function. Then, a set of reducers merges the values, according to a user-defined reduce function. The reduce phase outputs a list of key-value pairs that represent the final output of the MapReduce computation. Additionally, the MapReduce paradigm is supported by the use of a distributed storage system that allows partitioning the input files in several smaller files across the cluster, increasing the scalability of the approach. Within this paper the open-source implementation of the MapReduce framework, Apache Hadoop, is used. It has two main features. First a MapReduce programming model, as explained above, and a resource manager (*Yet Another Resource Manager*, YARN) to schedule computational jobs along a cluster providing high-scalability and fault-tolerance capabilities. Second, a distributed file system (*Hadoop Distributed File System*, HDFS) able to split input data in several blocks that are distributed and replicated along the cluster to provide high-throughput and reliable access to them.

III. Computing Intermediate Event Log Abstractions Using MapReduce

A. Conceptual Approach

The MapReduce-based process discovery approach proposed in this paper is based on calculating the intermediate log abstractions (directly-follows graph, dependency graph, etc.) required by the process discovery techniques. Computation of these intermediary results is done using one or several MapReduce jobs. The final process model is calculated in the ProM framework. The design decision is based on the observation that computing the intermediate abstractions is the most computational expensive task since the whole event

log must be analysed and different calculations must be performed. Additionally, by calculating the final process model in ProM, we can benefit from the use of the algorithms and abstractions already included in the ProM framework.

First, the input event log is split into several independent sublogs that contain a subset of traces. During the *map* phase, the specific computations needed for the discovery algorithm are performed calculating *local* abstractions corresponding to each sublog. Next, the *reduce* phase aggregates all intermediate data to get a final abstraction of the whole event log and additionally performs, if necessary, some basic transformations. Finally, when the last abstraction is calculated using one or several MapReduce jobs, the final process model is discovered inside the ProM framework. Note that this is possible since the size of the intermediate abstractions is typically small (only depending on the number of unique activities and not on the number of events).

The distributed computing approach takes advantage of the possibility of calculating the intermediate abstractions in parallel, at trace level, and aggregating the results later. This type of computation is highly suitable to be modelled within MapReduce since the computation of the intermediate phase can be performed during the *map* phase and the aggregation of the results can be carried out during the *reduce* phase.

An important aspect of the approach is the input format used to represent the event logs. The *eXtensible Event Stream* (XES) format[4] [14] is used since it is a standard language to represent event logs tailored towards process mining, data mining, text mining and statistical analysis. To use the MapReduce based approach, the event log must be stored in the HDFS. Thus, the event log is split in blocks of some predefined size, i.e, every block can be considered as an independent sublog. Then, a *mapper* calculates the intermediate data corresponding to each sublog. Although the blocks do not respect the division of the log in term of traces, Hadoop automatically manages the traces that are split in two blocks providing complete traces to mappers.

B. Intermediate Abstractions

Both the Alpha Miner and the Inductive Miner use the directly-follows graph as an intermediate abstraction of the event log (`stage 1` of Figures 2a and 2b). Hence we design the computation of the directly-follows graph as a MapReduce job. Figure 3 shows schematically the calculation of the *directly-follows* graph illustrating the MapReduce approach proposed in this paper. First, the log must be placed in the HDFS. As a consequence, the input log is automatically partitioned in several sublogs which serve as input to the mappers. Next, a directly-follows graph from each sublog is calculated during the *map* phase. For instance, the local graph corresponding to the second sublog in the example indicates that a is the start activity two times, a is followed by b twice, b is followed by d on two occasions and d is the end activity two times. Finally, all the intermediate graphs calculated by the mappers are aggregated during the *reduce* phase by using a single reducer. Thus, a directly-follows graph representing an abstraction of the whole log is calculated as output of the MapReduce job and used to discover a process model by applying the Alpha or the Inductive discovery algorithm.

[4]http://www.xes-standard.org/

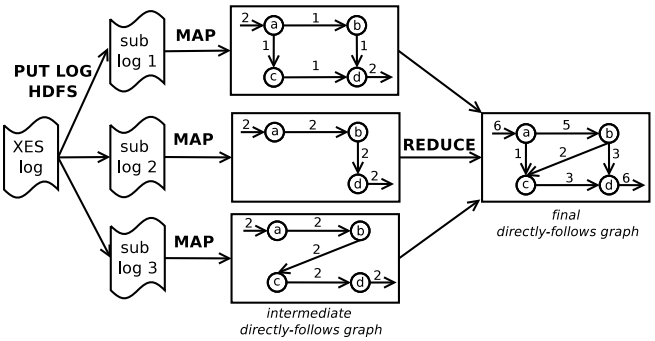


Figure 3: Example showing the MapReduce computation of a directly-follows graph in Hadoop.

For the *Flexible Heuristics Miner*, two MapReduce jobs are executed sequentially. The first one calculates the dependency graph and the second one annotates it by computing the splits and joins. The *map* phase of the first MapReduce job calculates the directly-follows, length-two loops and long distance relations graphs of each sublog (stage 1 of Figure 2c). Next, a *reducer* aggregates all the graphs and computes the dependency graph that is written in the HDFS as result of the job (stage 2 of Figure 2c). The second MapReduce job computes the annotated dependency graph (stage 3 of Figure 2c). Before starting the job, the dependency graph generated in the first MapReduce job is written in the Hadoop *distributed cache* so it can be used by all the mappers of the second job. These *mappers* calculate the splits and joins of each sublog using the dependency graph. Finally, they emit the split and joins that are received by the *reducer* to compose the annotated dependency graph that is written as output.

C. Final Process Model

Once the event log abstractions have been calculated using MapReduce, the final process model is discovered using the ProM framework. Thereby, these abstractions are downloaded from the remote Hadoop cluster and the selected discovery algorithm is applied to get the final process model. This way, we benefit from using the algorithms and techniques previously developed within the tool.

The integration of the MapReduce-based approach and ProM provides plugins with the ability to establish a connection with a Hadoop cluster, to import XES logs stored in a remote HDFS and to execute the implemented Hadoop-

based process discovery techniques. The plug-ins automatically manage the communication with the Hadoop cluster, the execution of the MapReduce jobs in the remote cluster, the transfer of the intermediate results and the computation and visualization of the final process model. Additionally, generic methods to enable the development of new process mining techniques are included. Thus, the developer only needs to implement the new MapReduce jobs and the final computation of the process model, meanwhile ProM manages the interaction with the remote cluster. For more details on the integration of Apache Hadoop in the ProM framework we refer to [15].

IV. Experimental Evaluation

We have evaluated the proposed approach to discover process models from large events logs using Apache MapReduce. In the experiments, we measured the execution time of applying different process discovery techniques and evaluated the scalability of the solution. The quality of the generated models is not evaluated as the focus of this work is on scalability of the existing techniques when they face huge event logs.

A. Experimental Setup

For the experiments, we have used a Hadoop cluster composed of 5 computing nodes. One of them is configured as the master node and the remaining four as worker nodes. The master node consists of eight Intel Xeon CPU E5430 at 2.66 GHz processors, 32 GB of RAM memory and five 300 GB hard disks. Each worker node consists of eight Intel Xeon CPU E5-2407 v2 at 2.40 GHz processors, 64 GB of RAM memory and eight 1 TB hard disks.

Regarding Apache Hadoop, version 2.6.0 was deployed in the cluster. The main configuration parameters establish that up to 16 Hadoop tasks per worker node can be carried out in parallel using up to 56 GB of memory. The HDFS is configured to store files in blocks of 256 MB making 2 replicas of each block. This configuration is in line with the recommended guidelines by different MapReduce distributions[5][6] and was adopted after different performance tests.

Finally, the following parameters for the process discovery techniques were used in all the experiments. For the Alpha Miner, there are no configuration parameters and hence, the default implementation is used. For the Inductive Miner, the *Inductive Miner - infrequent* variant with the *noise* threshold set to 0.2 is used. For the Flexible Heuristics Miner, the *all tasks connected* and *long distance dependency* heuristics are selected whereas the *dependency*, *length-one-loop*, *length-two-loops* and *long-distance* thresholds are set to 90.0 and the *relative-to-best threshold* is set to 5.0.

[5]http://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-common/ClusterSetup.html

[6]http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html

All results presented below are based on an average of five executions.

B. Varying the Size of the Event Log

The aim of this experiment is to evaluate the scalability of the presented approach according to the total size of the event log, i.e., the total number of events in the log.

Setup: The dataset used in the experiments consists of several event logs of different size that have been created randomly from a process tree consisting of 40 activities. The size of the events logs varies from 2 GB to 218 GB, from almost 1 million to 100 million traces and from around 32 million to 3.500 million events (Table II). On average, the traces are composed of 35 events.

Results: Figure 4 summarizes the obtained results. Specifically, Figure 4a shows the execution time of applying the Alpha Miner and the Inductive Miner whereas Figure 4b shows the results concerning the Flexible Heuristics Miner.

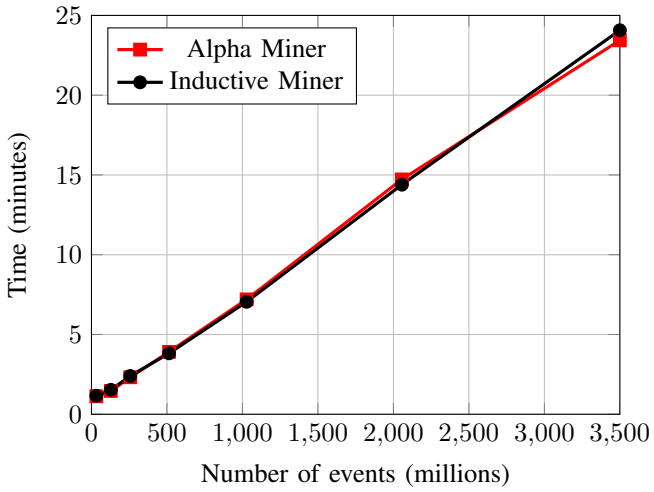
The results of Alpha and Inductive Miner are comparable. Both algorithms perform the same computation in the Hadoop cluster (calculating the *directly-follows* graph) and, then, use this information as an input for the mining algorithms which take only a few seconds.

For the Flexible Heuristics Miner, Figure 4b shows the execution time of the first MapReduce phase (*Computing DGraph* dotted line), the execution time of the second MapReduce phase (*Annotating DGraph* dotted line) and the total execution time resulting of considering both phases (*Flexible Heuristics Miner* line). There is a big difference between both MapReduce phases. The execution time of the first phase is slightly higher than the execution time of Alpha Miner and Inductive Miner. The second phase is much more computationally expensive and accounts for around 90% of the total execution time of the discovery algorithm.

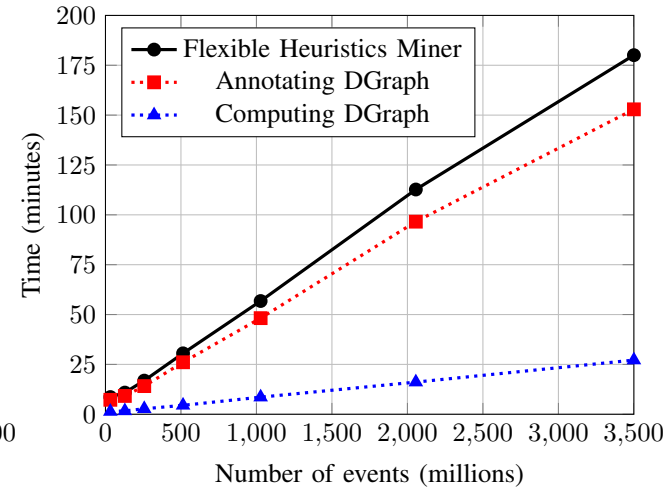
Discussion: The experiments show that the proposed approach is scalable in terms of event log size. We identify a linear increase in execution time w.r.t. the event log size for all techniques. An interesting result is the small difference between the execution time of the Alpha Miner, the Inductive Miner and the execution time of the first part of the Flexible Heuristics Miner. In the two first algorithms, this time corresponds to the computation of the directly-follows graph, whereas in the second case, it corresponds

Log size (GB)	Number of traces	Number of events
2	926.006	32.407.324
8	3.671.032	128.481.890
16	7.341.929	256.963.820
32	14.683.619	513.927.754
64	29.367.284	1.027.855.444
128	58.734.750	2.055.710.794
218	100.000.000	3.499.987.460

Table II: Summary of log size, number of traces and events corresponding to the event logs forming the *first* dataset used in the experiments.



(a) Alpha Miner and Inductive Miner.



(b) Flexible Heuristics Miner.

Figure 4: Scalability results when varying the size of the event log. The plots show the execution time of the studied process discovery techniques when the total number of events in the log is increased.

to calculation of the directly-follows graph including the length-two loops graph and the long distance relations graph. Thus, the additional time used to calculate more information in the first part of the Flexible Heuristics Miner does not have a great impact on the final execution time. Furthermore, it is explained because Hadoop is best suited to support large and computing-intensive tasks instead of real-time ones.

C. Varying the Number of Computational Resources

In this experiment we analyse the speed-up achieved when increasing the number of computing resources used.

Setup: In this experiment, the previous experiment was repeated with different numbers of available computational resources. The experiment is repeated using 1, 2 and 3 worker nodes and is compared with the previous results, i.e., where the whole cluster (4 worker nodes) was used.

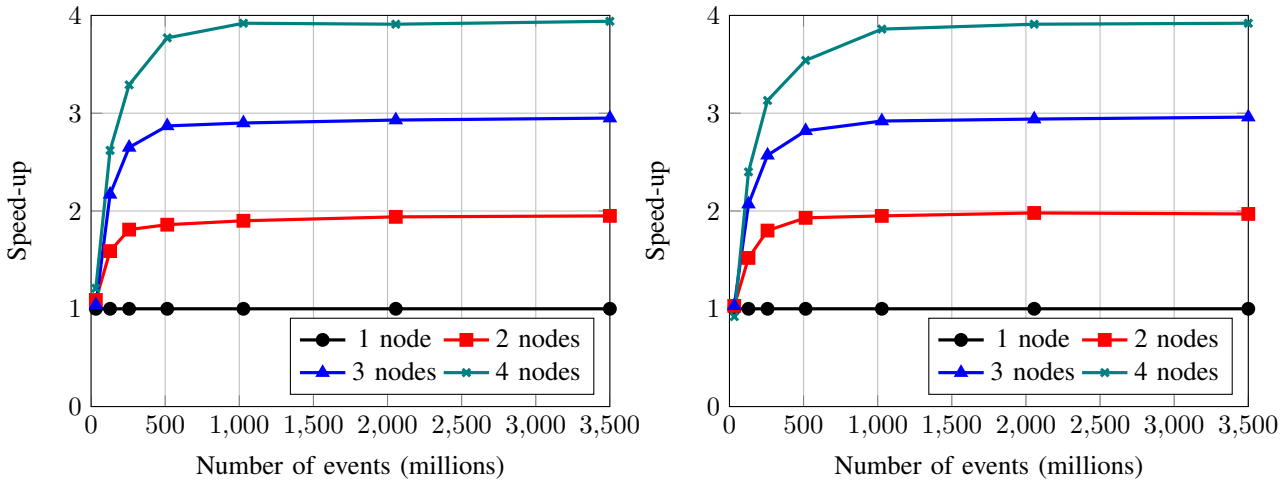
Results: Figure 5 shows the results of this second experiment. The figure shows the speed-up of the 2, 3 and 4 workers nodes deployments compared with the single worker configuration of the Hadoop cluster. Figure 5a shows the speed-up of computing the directly-follows graph in Hadoop, corresponding to computation performed in both the Alpha Miner and the Inductive Miner. Figure 5b shows the results corresponding to the execution of the Flexible Heuristics Miner. In both cases, one can see the same trend and the algorithms get an almost linear speed-up when the input log is large enough.

Discussion: The approach presents a great scalability with the number of worker nodes. The results show an almost linear speed-up compared to the 1 worker node deployment when the log is large enough. The execution time for the 2 GB log is independent of the number of computing resources because the whole computation can be completed using only 1 worker node and as a consequence, there is no speed-up when several nodes are added. In the 2 nodes deployments, an almost linear speed-up is achieved when the log size is at least 16 GB. When 3 worker nodes are used, the linear speed-up is obtained from the 32 GB log. Finally, for the whole cluster deployment, the size varies in the Alpha and Inductive Miner and in the Flexible Heuristics Miner. For the first two miners, an almost linear speed is achieved with the 32 GB logs. However, for the Flexible Heuristics Miner we at least need the 64 GB event log. The reason for the requirement of increasing the event log size to achieve linear speed-ups is twofold. First, more mappers are needed to fill all the worker nodes. Second, communication and network-related delays increase. Obviously, these values will vary in other clusters since it depends on the number of parallel tasks executed in every node and the HDFS block size.

D. Varying the Number of Activities

The main reason for analysing the impact of the number of activities with regard to performance is its great influence on the computational cost of process discovery techniques. Some techniques analyse complex properties and explore several possibilities whereby the number of events per trace and the number of activities forming the process can greatly influence the execution time, e.g., in the case of the Flexible Heuristics Miner.

Setup: In this experiment, three different datasets representing three different business processes are used. The first dataset corresponds to the event logs used in the previous experiments, i.e. as shown in Table II. The second dataset corresponds to event logs generated from the same underlying business process with an additional loop present



(a) Directly-follows graph.

(b) Flexible Heuristics Miner.

Figure 5: Scalability results varying the number of worker nodes used in the Hadoop cluster. The figures show the speed-up of distributed configurations compared to a single worker node deployment.

Log size (GB)	Number of traces	Number of events
3,6	926.006	64.814.648
14,2	3.671.032	256.963.780
28,5	7.341.929	513.927.640
56,9	14.683.619	1.027.855.444
113,9	29.367.284	2.055.710.888
227,8	58.734.750	4.111.421.588
387,9	100.000.000	6.999.974.920

Table III: Summary of log size, number of traces and events corresponding to the event logs in the *second* and *third* datasets used in the last scalability experiment.

at the end of the process model. The loop is configured to execute one additional iteration of the process. The number of *events* per trace doubles compared to event logs in the first dataset whereas the number of *activities* remains the same. The third dataset is based on the process model used for the second one. However, the activities in the second iteration of the loop have been renamed. Hence, the event logs in the third dataset has both a double number of events and a double number of activities per trace, compared to the logs in the first dataset. The characteristics of the two additional datasets are depicted in Table III. Summarizing, the logs in the first dataset include 40 possible activities and 35 events per trace, the ones in the second dataset include 40 possible activities and 70 events per trace whereas the logs in the third dataset include 80 possible activities and 70 events per trace.

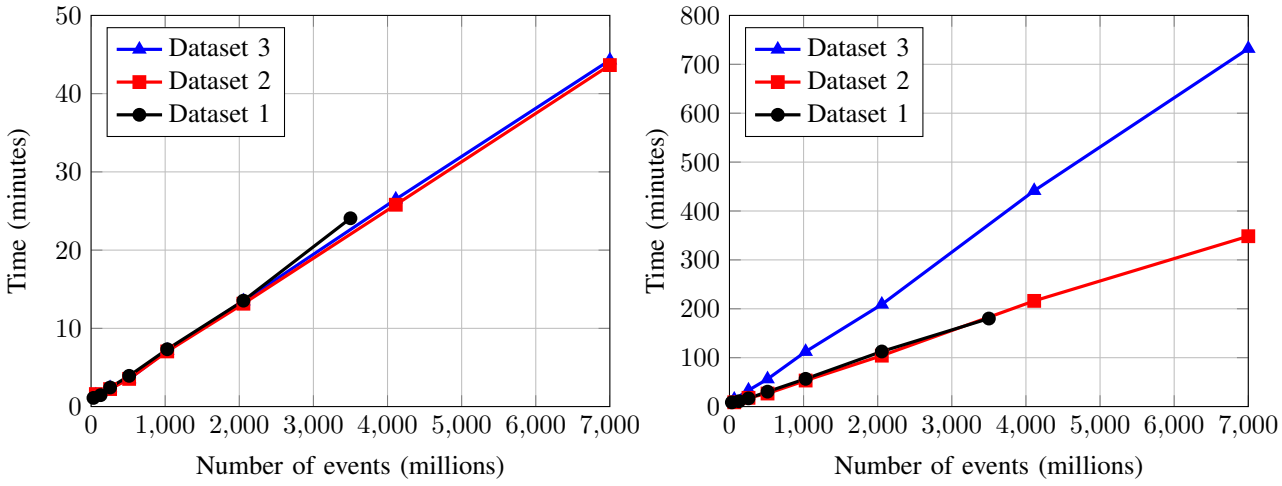
Results: Figure 6 shows the results of the experiment corresponding to the three different datasets. Note the first dataset contains logs with half events than the logs in the other two datasets. The results of Inductive Miner are provided in Figure 6a. The execution time is similar for the three datasets. The results of the Flexible Heuristics Miner are depicted in Figure 6b. In this case, there is a big difference between the execution time of the logs in the third dataset compared to the executing time of the ones in the first two datasets. This is due to the computational cost of the Flexible Heuristics Miner varies with the number of activities in the log. The results related to the Alpha Miner results are not shown since they are again roughly equal to the results of the Inductive Miner.

Discussion: On the one hand, the results of the Inductive Miner, and consequently the Alpha Miner, show that the number of events per trace and the number of activities have no influence on the techniques. The execution time is linear in the size of the event logs since the computation only considers the current and the next event in a trace. On the other hand, the characteristics of the event log greatly influence the execution time of the Flexible Heuristics Miner. The first two datasets show the same trend and have a similar execution time when the same number of events is considered. However, the computation of the annotated dependency graph corresponding to the logs in the third dataset is much slower. This difference is mainly due to the number of activities heavily impacts the computation of the splits and joins used to annotate the dependency graph [13].

E. A Note on Storing Event Logs in the HDFS

The event logs have to be written to the HDFS, prior to using them. We analysed the time required to perform this operation for the three datasets considered in the evaluation.

Results: Figure 7 shows the execution time required to store the event logs in the datasets according to their size. The time required to store the input log in the HDFS is



(a) Inductive Miner.

(b) Flexible Heuristics Miner.

Figure 6: Results of the influence of the number of events and activities experiment. The plots show the execution time of the studied process discovery techniques when they are applied to event logs representing similar processes but with different characteristics.

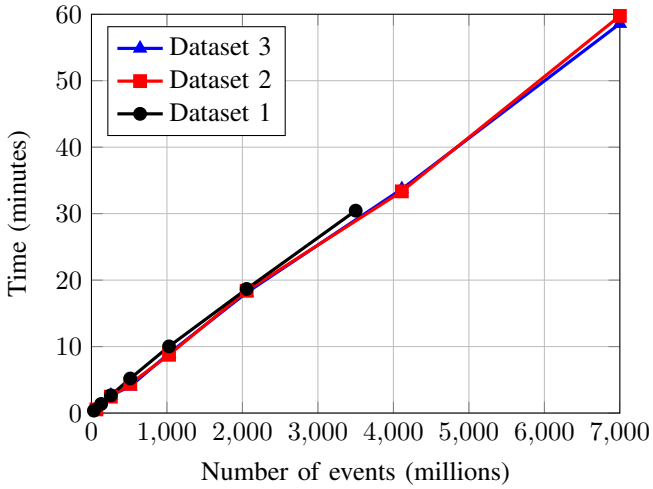


Figure 7: Time required to store the event logs of the three considered datasets in the Hadoop Distributed File System.

linear with the log size. The average I/O recorded is around 113 MB/s.

Discussion: Storing files in the HDFS has a linear performance since the event log must be read and written sequentially. Note that the contents of the input file are not relevant for this operation since Apache Hadoop splits the file in blocks of exactly 256 MB, as indicated in the configuration, without considering its inner content.

F. General Discussion

The execution time of the Alpha Miner and the Inductive Miner are roughly equal since both algorithms share the same intermediate abstraction. Moreover, discovering the process model is much faster than computing the intermediate directly-follows graph. The execution time of computing the dependency graph (first MapReduce job of the Flexible Heuristics Miner) takes a slightly longer time than discovering the directly-follows graph. If we consider the fact that this MapReduce job calculates more abstractions than just the directly-follows graph, we can conclude that the approach is more beneficial to address computing-intensive discovery algorithms.

One of the requirements of the framework used for evaluation is the use of the XML-based XES format to represent the event log. However, event logs can be represented using different formats, e.g. `CSV` (*Comma Separated Values*). Although the XES format is the standard format to represent event logs, it could be interesting to explore the use of different formats as well. Furthermore, the input format heavily impacts the size of the log and, consequently, the I/O performance. XES allows to perform specific computations trace by trace, as required for the process discovery techniques. The read performance is however rather low. If we consider other common formats, like `CSV`, the read performance should be higher because events are read line by line. In the case of `CSV` we however need an additional, initial, MapReduce phase to transform the data in order to be able to analyze the event log trace by trace. Another possibility could be using a format specifically designed to work in MapReduce environments like Apache Avro[7], that is designed to handle XML files in Hadoop. Most likely, this will enable an improvement in I/O performance and a

[7]http://avro.apache.org/

reduction in log size.

V. Related Work

An overview of different options for distributed process mining is presented in [5]. This work proposes to compute several process models by partitioning the log in several sub-logs and merging the results to obtain a final model. The first option is to divide the event log in sub-logs that contains the whole process but fewer cases (*vertical partitioning*). The second option is to split the log in sub-logs that contain specific parts of the process, i.e., by projecting on a subset of activities (*horizontal partitioning*).

Closely linked to the previous approach is the use of *divide-and-conquer* approaches [4], [16]–[18]. In general, the aim is to partition the event log, ensuring the resulting models can be merged getting a valid result. Specifically, the use of *passages*, special pairs of events that allows to decompose the event log and aggregate the final results with correctness guarantees, is explored in [16]; decomposing the event log using overlapping sets of events is proposed in [17] and partitioning the process model in several *single-entry single-exit* subprocesses for conformance checking in the large is presented in [18]. However, the previous techniques are limited to Petri nets. Finally, in [4] the divide and conquer approach is generalized beyond Petri nets. Compared to the previous approaches, our technique is based on a vertical partition of the event log since this model is more suitable for parallel MapReduce computations.

MapReduce is rarely applied in a process mining context. In [19], MapReduce has been used to support event correlation discovery, i.e., to identify the events that are part of the same case. The approach uses two MapReduce jobs to compute simple correlations conditions in the first job and composite ones in the second job. In our case, we assume that the input event log has information about the process instance corresponding to each event. In the context of process discovery, MapReduce was previously used to implement the Alpha Miner and the Flexible Heuristics Miner using the Amazon Elastic Map-Reduce service [7]. The implementation uses two MapReduce jobs for the Alpha Miner and five jobs for the Flexible Heuristics Miner.

Although the work presented in [7] is related to the approach presented in this paper, the implementation is significantly different. Our approach overcomes some of the problems not addressed in [7]. The architecture proposed in [7] does not fit properly with a MapReduce approach since it uses the map phase to collect data and several reducers to perform different computations in parallel. This design decision has two main drawbacks. First, multiple "identity" map phases are used, i.e., map phases that do not manipulate any of the input key-value pairs. This increases the amount of data transferred over the network and results in useless computations with the sole goal of grouping sparse data by key. Second, the number of key-value pairs emitted in some reduce phases is higher than the input key-value pairs. In such cases, much more data must be written to the distributed storage system which increases the execution time. Likewise, much more data must be read as input of the following map phase. In this situation, using an "identity" reducer is preferable.

We can not directly compare the approach of [7] to the approach proposed here as different computing resources and event logs are used. Moreover, the developed techniques are not available so the work is hardly reproducible. In any case, the execution times presented in the paper show poor scalability of the approach. When moving from a cluster of 10 general-purpose processors (that uses up to 20 parallel threads) to a cluster of 320 high-performance processors (that uses up to 480 parallel threads), the speed-up is 10.5 for the Alpha Miner and 6.27 for the Flexible Heuristics Miner. Hence, we adopted a new scheme to better allow for scalable distributed process discovery.

Finally, there is a big difference in the size of the logs used in both works. In [7] the discovery techniques are applied to a log consisting of 5 million traces and around 184,5 million events. In this work, we apply them to logs consisting of up to 100 million traces and almost 7 billion events getting scalable results. Thus, our approach overcomes the approach in [7] since it is applied to bigger logs and results prove the scalability of the approach both regarding the log size and the number of computational resources used.

VI. Conclusion and future work

In this paper, we have evaluated the scalability and validity of using a MapReduce-based approach for large-scale process discovery. The approach used for evaluation is based on computing intermediate abstractions, such as the directly-follows graph, used by the process discovery algorithms as MapReduce jobs. Our evaluation demonstrated that the approach is able to handle large event logs in an efficient and scalable manner. Additionally, the solution has been implemented within the ProM framework enabling researches and practitioners of the process mining community to use the developed algorithms and to implement new MapReduce-based techniques. This overcomes one of the main limitations of the tool since it was unable to analyse huge event logs.

As future work, we plan to develop new process discovery techniques using this approach and to extend the approach to conformance checking and process enhancement. Specifically, we consider that computing *alignments* should be beneficial due to its high computational cost [20]. Another interesting topic is to support the use of different input formats, such as Comma Separated Values (CSV) or Apache Avro, to represent the input event logs and to explore the performance and scalability of the algorithms when different input formats are used. Also, we would like to explore the possibility of partitioning the log horizontally, by subsets of

activities, since super-linear speed-ups could be obtained for process mining techniques whose computational complexity is exponential in the number of activities [4]. Finally, we intend to explore different technologies and approaches to support in process mining in the large (Big Data). In this aspect, we will explore other technologies that are built on top of Hadoop like Apache Spark[8], other distributed computing paradigms like grid and cloud computing [21] and the integration of several computing infrastructures that allow for more parallelism level in discovery algorithms, thus further reducing the execution time [22].

ACKNOWLEDGMENT

This work has been supported by the research projects TIN2014-56633-C3-2-R, granted by the Spanish Ministerio de Economía y Competitividad, FPU12/04775, granted by the Spanish Ministerio de Educación, Cultura y Deporte, 287230/2, granted by Gobierno de Aragón, and the DELI-BIDA (Desire Lines in Big Data) research program supported by NWO.

REFERENCES

[1] R. Smolan and J. Erwitt, *The Human Face of Big Data*. Against All Odds Productions, 2012.

[2] W. M. P. van der Aalst, "Data scientist: The engineer of the future," in *Enterprise Interoperability VI*. Springer, 2014, pp. 13–26.

[3] ——, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st ed. Springer Publishing Company, Incorporated, 2011.

[4] ——, "A general divide and conquer approach for process mining," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*. IEEE, 2013, pp. 1–10.

[5] ——, "Distributed process discovery and conformance checking," in *Fundamental Approaches to Software Engineering*. Springer, 2012, pp. 1–25.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] J. Evermann, "Scalable Process Discovery using Map-Reduce," *Services Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[8] A. Nandi, C. Yu, P. Bohannon, and R. Ramakrishnan, "Data cube materialization and mining over mapreduce," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 10, pp. 1747–1759, 2012.

[9] Object Management Group (OMG), "Business process model and notation (BPMN) version 2.0," Tech. Rep., 2011.

[8]http://spark.apache.org/

[10] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs," *Inf. Syst.*, vol. 37, no. 7, pp. 654–676, 2012.

[11] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 9, pp. 1128–1142, 2004.

[12] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs - A constructive approach," in *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013*, 2013, pp. 311–329.

[13] A. Weijters and J. Ribeiro, "Flexible Heuristics Miner (FHM)," in *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*. IEEE, 2011, pp. 310–317.

[14] H. Verbeek, J. C. Buijs, B. F. Van Dongen, and W. M. P. van der Aalst, "XES, XESame, and ProM 6," in *Information Systems Evolution*. Springer, 2011, pp. 60–75.

[15] S. Hernández, S. J. v. Zelst, J. Ezpeleta, and W. M. P. van der Aalst, "Handling Big(ger) logs: Connecting ProM 6 to Apache Hadoop," in *To appear in Proceedings of the BPM Demo Sessions 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015)*, 2015.

[16] W. M. P. van der Aalst, "Decomposing process mining problems using passages," in *Application and Theory of Petri Nets*. Springer, 2012, pp. 72–91.

[17] W. M. van der Aalst, "Decomposing Petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 471–507, 2013.

[18] J. Munoz-Gama, J. Carmona, and W. M. P. van der Aalst, "Conformance checking in the large: Partitioning and topology," in *Business Process Management*. Springer, 2013, pp. 130–145.

[19] H. Reguieg, F. Toumani, H. R. Motahari-Nezhad, and B. Benatallah, "Using MapReduce to scale events correlation discovery for business processes mining," in *Business Process Management*. Springer, 2012, pp. 279–284.

[20] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*. IEEE, 2011, pp. 55–64.

[21] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*. IEEE, 2008, pp. 1–10.

[22] J. Fabra, S. Hernández, J. Ezpeleta, and P. Álvarez, "Solving the interoperability problem by means of a bus," *Journal of grid computing*, vol. 12, no. 1, pp. 41–65, 2014.