Applying MapReduce to Conformance Checking

I.S. Shugurov <shugurov94@gmail.com>
A.A. Mitsyuk <amitsyuk@hse.ru>
National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems, 20 Myasnitskaya St., Moscow, 101000, Russia

Abstract. Process mining is a relatively new research field, offering methods of business processes analysis and improvement, which are based on studying their execution history (event logs). Conformance checking is one of the main sub-fields of process mining. Conformance checking algorithms are aimed to assess how well a given process model, typically represented by a Petri net, and a corresponding event log fit each other. Alignment-based conformance checking is the most advanced and frequently used type of such algorithms. This paper deals with the problem of high computational complexity of the alignment-based conformance checking algorithm. Currently, alignment-based conformance checking is quite inefficient in terms of memory consumption and time required for computations. Solving this particular problem is of high importance for checking conformance between real-life business process models and event logs, which might be quite problematic using existing approaches. MapReduce is a popular model of parallel computing which allows for simple implementation of efficient and scalable distributed calculations. In this paper, a MapReduce version of the alignment-based conformance checking algorithm is described and evaluated. We show that conformance checking can be distributed using MapReduce and can benefit from it. Moreover, it is demonstrated that computation time scales linearly with the growth of event log size.

Key words: process mining; conformance checking; MapReduce; Hadoop; big data.

DOI: 10.15514/ISPRAS-2016-28(3)-7

For citation: Shugurov I.S., Mitsyuk A.A. Applying MapReduce to Conformance Checking. *Trudy ISP RAN / Proc. ISP RAS]*, vol.28, issue 2, 2016. pp. 103-122. DOI: 10.15514/ISPRAS-2016-28(3)-7

1. Introduction

Ever-increasing size and complexity of modern information systems force both researchers and practitioners to find novel approaches of formal specification, modeling, and verification. This process is essential for ensuring their robustness and for possible optimization and improvements of existing business processes.

Process mining is a research field, which offers such approaches [1]. *Process mining* is a discipline, which combines techniques from data analysis, data mining, and conventional process modeling. Typically, three main sub-fields of process mining are distinguished in the literature: (1) process discovery; (2) conformance checking and (3) enhancement [1].

The aim of *process discovery* is to build a process model based solely on the execution history of a particular process. Event logs are the most common and natural way of persisting and representing execution history. By an event log, we understand a set of traces where each trace corresponds exactly to one process execution. A typical process discovery algorithm takes an even log as an input parameter and constructs a process model which adequately describes the behavior observed in the event log.

The task of *conformance checking* is to measure how well a given process model and an event log fit each other. Furthermore, showing only the coefficient of conformance is usually insufficient for real-life application since analysts often need to see where and how often deviations happen in order to draw any conclusions. Therefore, it is often the case when conformance checking algorithms include computation of additional metrics as well as visualization of deviations.

Process enhancement deals with improvements of processes as well as corresponding process models.

One of the challenges of process mining, when applied in real life, is the size of data to be processed and analyzed [2], [3]. Since process discovery has drawn significant attention of researchers, there are a number of solutions which allow for fast process discovery from large event logs [4]. These solutions vary from using distributed systems and parallel computing [5] to applying more efficient algorithms, which require less data scans and manipulations [6], [7]. In contrast, conformance checking remains problematic to be made fast due to its theoretical and algorithmic difficulties. At the same time, efficient, easy-to-use and robust conformance checking is the key to better process improvement since enhancement approaches often rely heavily on measuring conformance (for example, see model repair approaches [8], [9]).

This paper concentrates on implementation details of distributed conformance checking rather than on its theoretical aspects. It describes a possible way of speeding up conformance checking. It implies improving one of the existing conformance checking algorithms so that it can be executed in a distributed manner by means of using MapReduce [10]. One of the very first papers discussing distributed conformance checking [11] was dedicated solely to theoretical foundations of process models and event logs decomposition. The author takes a look at the algorithmic side of distributed conformance checking and totally skips problems of its software implementation. In this paper, we consider practical aspects of distributed conformance checking. Furthermore, we prove viability of the proposed approach by demonstrating that it really allows measuring conformance of bigger event logs better than currently existing approaches.

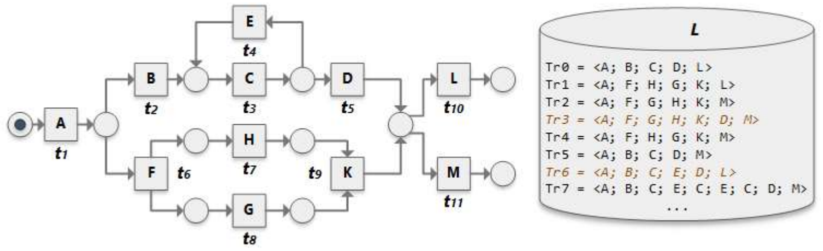
This paper is structured as follows. Section 2 introduces foundational concepts we use in the paper. In section 3, the reader can find the main contribution. Section 4 proposes several improvements of the approach proposed in section 3. An implementation of the presented approach is described in section 5. Related work is reviewed in section 6. Finally, section 7 concludes the paper.

2. Preliminaries

In this paper, we consider process models in the Petri net (simple P/T-nets) notation. A *Petri net* is a bipartite graph, which consists of nodes of two types. In process mining, transitions, denoted by rectangles, are considered as process activities, whereas places, denoted by circles, designate the constraints imposed on the control-flow. String labels may be associated with transitions in order to show the correspondence between activities and transitions. Transitions without labels are called *silent*. It implies that silent transitions model behavior and constrains of an activity in a process, executions of which are not recorded into event logs. Each place denotes a causal dependence between two or more transitions. Places may contain so-called *tokens*. A transition may fire if there are tokens in all places connected to it via incoming arcs. When fired, it consumes one token from each input place and produces one token to each output place. *Marking* is a distribution of tokens over all places of a Petri net, thus a marking denotes the current state of a process.

An *event log* is a recorded history of process runs. Usually the execution of a process in some information system is recorded for documenting, administrative, security, and other purposes. The main goal of process mining is to explore and use these data for the diagnosis and improvement of actual processes.

We consider event logs of standardized nature as they are used in process mining. Formally, an event log is multiset of traces where each trace is a sequence of events. Each trace corresponds to exactly one process run. An event contains the name of associated activity, timestamp, performer name and may contain other additional properties. In this paper we consider simple event logs, in which events contains only names of activities. An example model and the corresponding event log are shown in fig. 1.



```
Tr0 = <A; B; C; D; L>
Tr1 = <A; F; H; G; K; L>
Tr2 = <A; F; G; H; K; M>
Tr3 = <A; F; G; H; K; D; M>
Tr4 = <A; F; H; G; K; M>
Tr5 = <A; B; C; D; M>
Tr6 = <A; B; C; E; D; L>
Tr7 = <A; B; C; E; C; E; C; D; M>
...
```

Fig. 1. Petri net and event log

2.1 Conformance checking

The conformance checking and its place in process mining are defined in [1]. Usually four dimensions of conformance are considered: fitness, precision, generalization, and simplicity. However, this paper focuses exclusively on fitness. By the term *fitness,* we understand the extent to which a model can reproduce traces from an event log. In other words, fitness shows how well the model reflects the reality. The fitness dimension is typically regarded as being the most frequently used and best-defined [1] among the dimensions.

Nowadays, the most advanced and refined conformance checking approach is the one using alignments [12]. The term *alignment* is used to denote the set of pairs where each pair consists of an event from an event log and a corresponding transition of a model. Such pairs are constructed sequentially for each event in a trace. A simple alignment for the trace $Tr3$ (see fig. 1) is depicted in fig. 2. However, it is allowed to pair an event with no transitions (a special ”no move“ symbol ≫). This means that the event is present in a log but cannot be replayed by any transition in the model. It is also possible to map a transition to no events (this is denoted by the same symbol ≫). In that case, the transition is fired but there is no evidence of this fact in the event log. Thus, there are two main types of steps composing any alignment: a synchronous move (a transition fired with the same label as an event name from the event log) and a non-synchronous move (a transition label and an event name are the different ones or a move is skipped either in the model or in log).

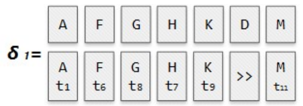


Fig. 2. Alignment

Alignments help to measure the difference between a trace from an event log and behavior specified by a model. In order to quantify the difference one has to calculate the number of non-synchronous moves and assess their significance. This assessment is accomplished by introducing a *cost function*, which is used for calculating *cost of an alignment*. By cost, we understand a number which somehow designates the significance. The general idea is that some deviations are more severe than others, thus these deviations have more impact on the overall conformance. Using cost function one can assign cost for each type of deviation for each transition and event. Thus, cost function maps a pair of an event and a transition to a number, which signifies a penalty for having such a pair in a trace. The more the cost is, the more significant this deviation is. Assuming that all costs are set to 1, the alignment shown in fig. 2 has the cost 1, because there is only one nonsynchronous move in it (event D in the trace has to be skipped during model run). Accumulating costs for all alignments of a particular event log, it is possible to derive the cost for the entire log.

It is possible that a particular run through the model and a particular trace have several possible alignments. In order to choose between them a cost function is used to evaluate the cost of each alignment. An alignment with the lowest cost is selected as the optimal alignment. According to [12], it makes sense to use only optimal alignments when calculating fitness. Alignment-based fitness can be measured using the metric defined in [13]:

$$f(L,N) = 1 - \frac{\sum_{tr\in L}\sum_{e\in tr} cost_{fn}^{\delta_{\text{opt}}}(e,N)}{\sum_{tr\,\in L} cost_{ai}}$$

where L is an event log, N is a model, $cost_{fn}^{\delta_{opt}}$ (e,N) is a cost of a pair $(e,(t_i,t_i^l))$ (e is an event, t_i is a transition from model run, t_i^l is its label) in the particular optimal alignment δ_{opt}, which depends on used cost function cf, $cost_{ai}$ is a total cost of the trace tr if all moves in it are considered as non-synchronous. Thus, fitness is a normalized ratio of the accumulated costs calculated for the optimal alignments to the accumulated costs for the worst possible alignments for a particular event log.

It is shown in [12] that construction of alignments and selection of optimal among them for each trace can be converted to solving the shortest path problem. Formally, a trace from the event log is represented as an event net, which is a special Petri net having the form of the sequence of transitions connected through places. Then the product of the model and this event net is constructed. It is shown in [12] that the problem of optimal alignment calculation can be viewed as a problem of finding a firing sequence in this product, which can be achieved by using a state-space exploration approach.

The proposed approach has a low computational performance when dealing with large models, large event logs or in case of low fitness because of the necessity to solve the shortest path problem, especially for model of certain types [12]. The author himself states in [12] that ”from a computational point of view, computing alignments is extremely expensive“. Moreover, its existing implementation keeps the processed models, event logs, event nets, and computed alignments in computer’s main memory. This approach allows for flexible configuration of visualization settings, and, in some cases, faster completion. However, this feature makes usage of existing implementation rather hard and inconvenient because the algorithm typically consumes several gigabytes of main memory even for processing relatively small models and small event logs (dozens of megabytes). Thus, it is not suitable for real-life usage.

This paper proposes a way of checking conformance between process models and big event logs of gigabyte sizes using MapReduce.

2.2 MapReduce

MapReduce is a computational model proposed and popularized in [10], although the idea dates back to the origins of functional programming. MapReduce is a

popular technology among practitioners and a research area among scientists. It has a good tool support; all major cloud platform vendors provide the possibility to execute MapReduce jobs on their cloud clusters.

The model simplifies parallel and distributed computing by allowing software developers to define only two quite primitive functions: *map* and *reduce*. At each invocation of a map function (also called *mapper*), it takes a key-value pair and produces an arbitrary number of key-value pairs. The aim of reduce functions (also called *reducers*) is to aggregate values with the same key and perform necessary computations over them. Thus, a reduce function takes a key-list pair as input parameters. Usage of such rather trivial functions makes their distribution straightforward. Last but not least, comes another important function allowed by MapReduce which is called *combine*. Its main purpose is to perform reduce-like computation between mappers and reducers. Combine functions (also known as *combiners*) are invoked on the same very computers as mappers. Combiners allow for further parallelizing computations and decreasing amount of data transferred to reducers and processed by them. It was pointed out even in the original article [10] that combiners may dramatically decrease computation time.

One of the most crucial advantages of MapReduce is that algorithms expressed in such a model are inherently deadlock-free and parallel. Another important advantage is the tendency to perform computations where required data resides. Generally, computation of map tasks take place where the required data is stored since its location is known beforehand. Such an approach ensures that data transfer between computers and latency, inflicted by it, are minimized. Ideally, data is transferred between computers where map tasks are executed and computers where reduce tasks are executed. Unfortunately, it is rarely achievable since all files are separated into smaller parts, called blocks, and distributed (and also replicated) over a cluster, thus data needed for execution of a single map task may reside in different data chunks — there will be a need to move a portion of data from one computer to another.

3. Fitness measurement using MapReduce

This section describes the approach we propose for checking conformance.

The few adjustments of the existing conformance checking algorithm with alignments need to be done in order to implement the proposed schema. It is expected that the algorithm will benefit if distribution is applied to traces. It means that traces are distributed over a cluster so that their alignments can be computed in parallel. Another possible option was to distribute computation of each alignment since efficient distributed graph algorithms for solving the shortest path problem are known. However, use of them seems excessive because they are aimed at solving problems on graphs consisting of thousands and millions of nodes, which is not the case for business process models. A process model consisting of more than a hundred nodes seems unrealistic.

The general schema is depicted in fig. 3. Map function takes traces one by one and computes their alignments. This process can easily be carried out in parallel since, by its definition, an alignment is computed individually for each trace. It is enough to use a single reduce function, which aggregates fitnesses of all traces and calculating fitness of the overall event log. Single reducer implies that key-value pairs emitted by all mappers have the same key. Single reducer can be considered as a bottleneck due to the reason that before it can start processing it waits for completion of all maps and transition of all costs to a single computer. To diminish the negative effect of a single reducer, a combiner function comes in handy. The problem is that calculating average is not an associative operation, thus it is impossible to use the basic reduce function instead of the combine function. We implemented it in a manner resembling the one described in [14]. The general idea is that calculating average can be easily decomposed into calculating a sum of all entries of some metric and counting a number of entries, where both of them are associative operations. It implies changing the structure of values used in key-value pairs. The modified version of values contains not only statistics (fitness and so on) but also a counter which shows how many traces describes a particular value. Given that, combiners only have to sum the values they receive and increment the counter.

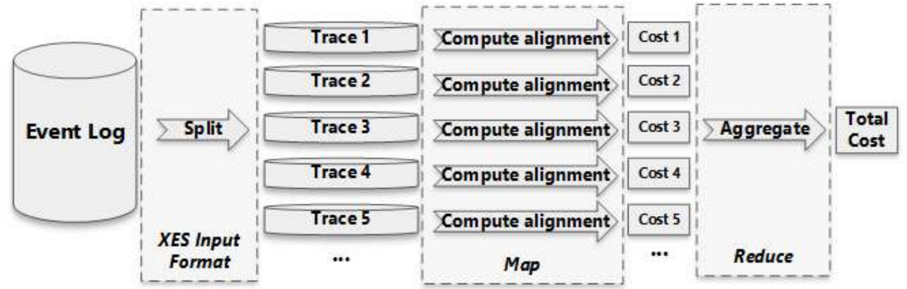


Fig. 3. Conformance checking with MapReduce

4. Potential improvements

One of the possible improvements of the algorithm is to enhance it by adding trace deduplication. When large event logs are considered, the possibility of the equivalent traces occurring several times is very high. Hence, it might be desired to find only unique traces, number of their occurrences and compute alignments only for them. It will allow for lessening the number of computed alignments. However, efficient MapReduce algorithm for deduplication of event sequences is far from trivial. Moreover, it is not guaranteed that time needed for deduplication and subsequent conformance checking will be shorter than in case of using the standard approach. This question can only be answered by conducting relevant experiments.

Even though process models is not prone to be large, a lot of time is still required for checking conformance. Another possible improvement, which aims at reducing model size, is to employ the ”divide and conquer“ principle. The way in which the

principle can be applied to cope with high computational complexity of conformance checking was proposed in [15] and [16]. The general idea is to divide a process model into smaller sub-parts. Next step is event log projection. This means that for each fragment of a model all events from the event log that correspond (names of events are equal to labels of activities) to a particular fragment are selected. As a result, we get as many projected event logs as decomposed Petri net fragment.

Once it is done, alignments and costs of each fragment can be computed. Then it is possible to sum costs of parts following specific rules to get a lower bound of the cost of the entire log. Having these costs, an upper bound of fitness can be computed. Performance gain is the most crucial motivation of this approach. Since time needed for computing alignments depends on trace size, usage of smaller parts of the model ensures faster computation. A wide range of model decomposition strategies have been proposed in [17], [15], [18], which leaves the user with the necessity to empirically choose between them. Last but not least, decomposition also incurs time overhead and projected event logs takes up disk space, so usage of the algorithm is not beneficial (or even feasible) in all the possible cases. Furthermore, there is no research done to establish when usage of which approach makes more sense.

It is possible to employ a similar approach in the MapReduce environment. There are two possible options: (1) computation of the overall event log fitness and (2) computation of fitness of each separate model part. In all the cases fitness is computed in a three-stage process as it is shown in fig. 4. The zero stage again is the splitting of the log by traces, which is followed by trace decomposition. Traces are decomposed using the maximal decomposition described in [15]. However, incorporation of other decomposition techniques [15], [17], [16] is also possible. At the second stage, alignments of sub-traces are computed and then aggregated. The final stage differs depending on the selected computation option. At this stage either fitness of the overall event log is computed at a single reducer or fitnesses of individual parts are computed at different reducers (the number of reducers can be up to the number of model parts). If fitness of individual process parts is calculated, after the second map unique identification of a model part is used as a key for emitted key-value pairs. When decomposition is applied, log deduplication’s importance and potential benefit grow even more.

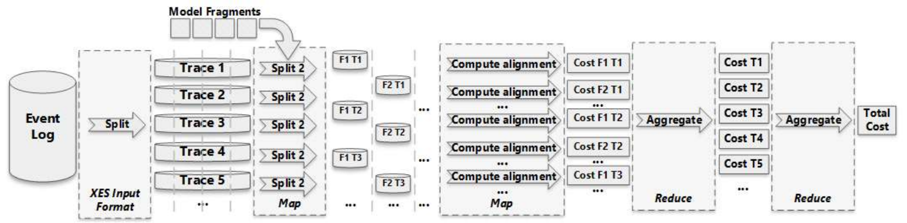


Fig. 4. Possible approach with vertical decomposition

5. Implementation and testing

This section describes the actual implementation[1] of the proposed approach and its experimental testing. Hadoop [19] was used for implementation and testing of the approach because it is a common and widely supported open source tool.

5.1 Implementation

The original algorithm was implemented as a ProM Framework plugin. The ProM Framework [20], [21] is a well-known tool for implementation of process mining algorithms. The ProM Framework consists of two main components:

- ProM core libraries which are responsible for the main functionality used by all users and extensions,
- extensions (typically called plugins) which are created by researchers and are responsible for import/export operations, visualization, and actual data processing.

The platform is written in such a way that it allows plugins to use data produced by other plugins. Furthermore, ProM encourages programmers to separate concerns: export plugins are only used for exporting data, visualization plugins are used for visualizing objects. As a result, a common usage scenario always consist of a chain of invocations of different plugins. Among main advantages of ProM are configurability, extensibility, and simplicity of usage. Last but not least, the platform allows researchers to easily create and share plugins with others thus extending the tool and contributing to the overall field of process mining. Despite all these positive sides, usage of ProM can be inconvenient and tedious, if the desired goal is unusual in any way.

XES [22] is often considered as a de facto standard for persisting event logs in the area of process mining. Technically, it is an XML-based standard, which means that it is tool-independent, extensible, and easy to use. Moreover, ProM fully supports this standard and has all required plugins for working with it.

Our approach involves usage of raw event logs stored in the format of XES only at the zero step of the algorithm. Before separate traces are available for the required computations, it is necessary to sequentially read XES files dividing them into separate traces. It is accomplished by using the *XMLInputFormat* from the *Mahout* project [23]. XMLInputFormat provides the capability of extracting file parts located between two specified tags. Moreover, the class is responsible for ensuring that the entire requested part (in our case — trace) is read, no matter in which blocks and on which data nodes it resides.

The fact that the initial algorithm was implemented for ProM inflicts several inconveniences for its distribution. First of all, it is assumed that the plugin is invoked by ProM via a special context. Essentially, it implies several things:

- the entire ProM distribution has to be sent to each computational node,

[1] The tool is available at https://sourceforge.net/p/distributedconformance/

- at each computational node, it is required to start up ProM (it may take up to couple of minutes on an average computer).

As a result, it may significantly increase latency and incur higher time needed for termination of computations. To avoid this, it was decided to alter implementation in such a way that a number of libraries the algorithm depends on in as minimal as it is possible to achieve. In other words, on the one hand it was desired to separate the implementation of the algorithm from ProM. On the other hand, usage of ProM could be useful for initial settings and visualization of final results. As a result, we achieved such a level of decoupling, that it is possible to launch the algorithm completely autonomously without the need of installation of the ProM Framework or any ProM plugins.

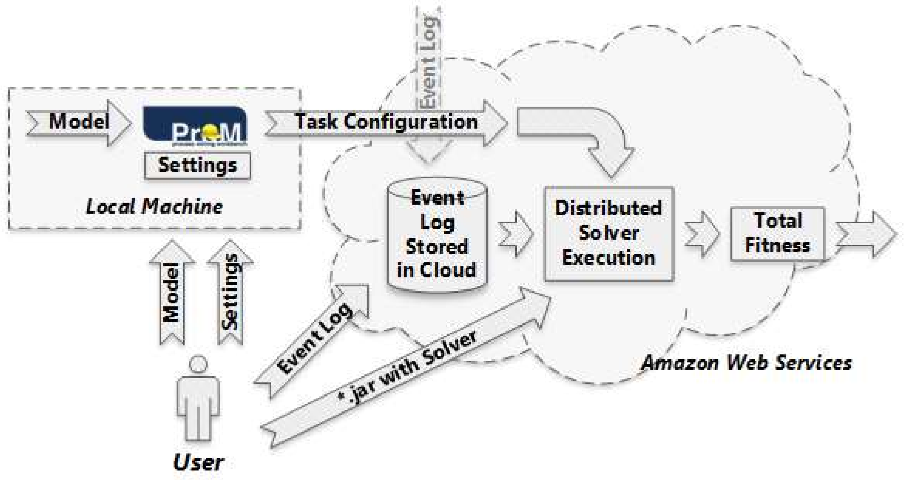


Fig. 5. Implementation of the approach

The resulting architecture is illustrated in fig. 5. Conformance measurement is done in two-step approach. At first step, the user loads a model, represented by a Petri net into a special ProM plugin, which serves for setting the options of the alignments-based conformance algorithm (mapping between transitions and events in event logs, costs of insertion and skipping in alignments). We use standard ProM classes for representing Petri nets because they allow for easier compatibility with other ProM plugins. Loading a model to a main memory should not be a problem because it is highly unlikely for such models to contain even hundreds of nodes, thus the size of process models is typically relatively small. Another possible option was to specify settings exclusively via XML files, though we found it less intuitive and convenient than visual settings. Once the algorithm is configured, settings are written to a file which later will be uploaded to a cluster. Last but not least, it is important to state that this ProM plugin depends neither on Hadoop nor on a chosen cloud cluster nor on any other auxiliary Hadoop libraries.

When Hadoop job is initiated, the user is asked to specify directories where event logs are placed, a path to a Petri net, and a path to conformance settings. A model and settings are then automatically added into the Hadoop distributed cache — the files are replicated to each data node, so they are available for fast access by any mapper. At a startup of each model, the files are loaded into main memory because they will be used for all the alignment computations.

After completion of conformance measurement, the results are written to a single file, which afterwards can be downloaded and viewed in ProM. Another sub-task is to find in which cases deduplication is worthwhile and how exactly it affects computational time.

5.2 Experimental results

The proposed algorithm was tested and evaluated using Amazon Web Services [24]. In our cluster, we used five m3.xlarge instances (one as a master node, four as data nodes). A local computer used for conducting experiments with the original algorithm had the following configuration: Intel Core i7-3630QM, 2.40 GHz, 8 GB of main memory, Windows 7 64 bit.

For testing purposes, we created a process model comprising some of the main workflow patterns: sequence, parallel split, synchronize, exclusive choice, and simple merge [25]. Afterwards, several models derived from the original were created — they all differ in fitness. Artificial event logs were generated using the approach proposed in [26]. Logs were generated only for the original model. All resulting logs were of different sizes.

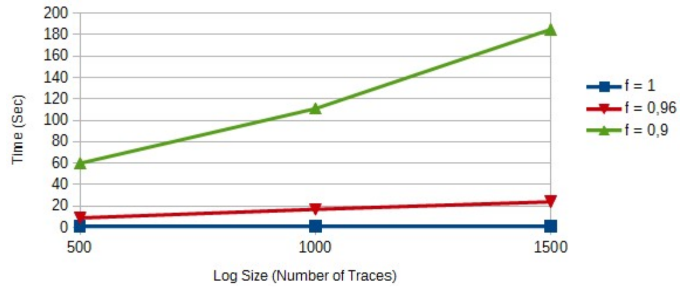


Fig. 6. Computation time of the standard approach

Fig. 6 illustrates how computation time depends on a number on traces and fitness. It is clear from the plots that computational complexity scales linearly with the growth of a number of traces. Moreover, it is seen that computation time highly depends on fitness. The lower the fitness, the slower the computations will be. It seems that computation time does not scale linearly with the decrease of fitness if the same quantity of logs is used. The clear indicators are the margins between lines representing fitness 1 and 0.96, and 0.96 and 0.9. Furthermore, we can conclude that

the lower fitness, the faster computation time increases with the rise of the number of traces.

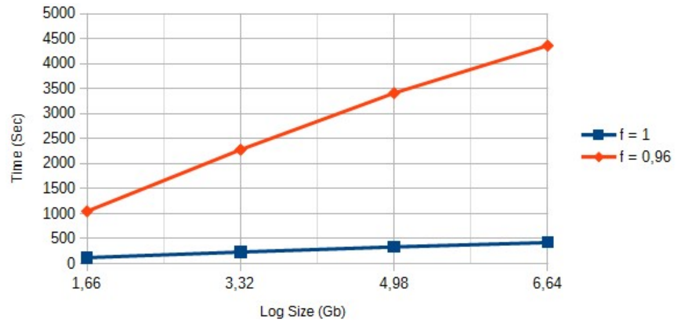


Fig. 7. Computation time with MapReduce

Fig. 7 provides an overview of how the algorithm scales when it is distributed using MapReduce. It is worth mentioning that 1.66 Gb of logs contain 500 thousand traces. As in the case of the not distributed algorithm, the graph shows that the algorithm scales linearly with the increase of a number of traces. Furthermore, similarly to the not distributed case, for non-fitting models computations take considerably longer than for perfectly fitting ones, and that computation time grows faster for non-fitting models.

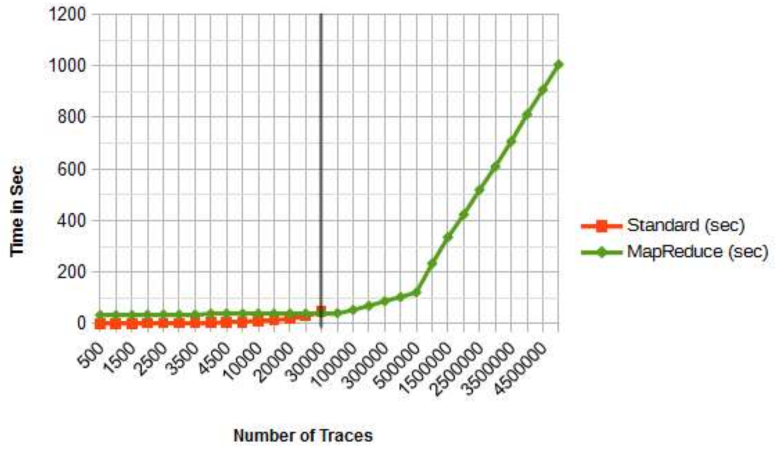


Fig. 8. Comparison of the standard and the distributed approaches

In fig. 8, a comparison of distributed and not distributed approaches is provided. Unfortunately, it is impossible to establish exactly when the distributed implementation beats the original in terms of performance since the original one cannot handle event logs of considerable size. In addition, the original algorithm was not able of handling more than a hundred of Mbytes. On data of such small sizes, MapReduce and Hadoop fail to work efficiently because they are designed for processing much bigger files. In fact, Hadoop does not parallelize processing of files which are smaller than a single file block. It is clear from Figure 8 that for relatively small event logs the distributed version works more slowly. It is clear from the graph that our solutions can handle event logs of several dozens of GBs even on a small cluster used for conducting these experiments.

6. Related work

Although applicability of MapReduce or distributed systems for the tasks of process mining has not drawn significant attention yet, there are a few papers, which consider this subject.

In [27] the authors focus exclusively on finding process and events correlation in large event logs. According to them, MapReduce solution for such a computationally and data intensive task as events correlation discovery performs well and can be scaled to large datasets.

Other works where the authors study applicability of MapReduce to process mining are [28], [29]. In these articles, a thorough description of several popular discovery algorithms is provided (the alpha algorithm [30], and the flexible heuristics miner [31]). Every one of them consists of several consequent MapReduce jobs. First MapReduce job is responsible for reading event logs from the disc, splitting them into traces, and ordering event in each trace. The general idea of the second MapReduce all the implementations is that first step of process discovery typically requires extracting trivial dependencies between events called *log-based ordering relations*. Examples of those are:

- a > b — event a is directly followed by event b,
- a >> b — a loop of length two,
- a >>> b — event a is followed by event b somewhere in the log.

These relations can be found individually for each trace. Therefore, their computations are trivially parallelized using Mappers. Further MapReduce jobs vary but they somehow use mined primitive log-ordering relations to build a process model. The main potential problem of implementations is that these further MapReduce jobs typically compute relations for the overall event log. To achieve this, it is often the case when it is necessary for mappers to produce identical keys for all emitted pairs so that they all end up on the same computer and processed by the same reducer. Moreover, the proposed implementations extensively use identity mappers. It is a standard term for mappers, which emit exactly the same key-value pairs as they receive without performing any additional computations — all useful computations performed by combiners or reducers. They are used only because MapReduce paradigm requires presence of mappers. Despite these concerns, it is shown that performance and scalability provided by MapReduce are good enough for the task of process discovery from large volumes of data. Our solution, in

contrast to the described above, uses a more suitable file format. It allows measuring conformance without extra steps needed for preliminary log transformations.

In [32] the authors describe their framework for simplified execution of process mining algorithms on Hadoop clusters. The primarily focus of this work is to show how process mining algorithm can be submitted to a Hadoop cluster via the ProM user interface. In order to demonstrate viability of their approach, the authors claim that they implemented and tested the Alpha miner, the flexible heuristics miner, and the inductive miner [33]. We opted for not using the presented framework in order to simplify the usage of our ProM plugin and not to force the user to download all the codebase required by Hadoop and its ecosystem.

To sum up, these papers clearly demonstrate not only that process mining can benefit from using distributed systems and MapReduce, but also that such distributed process mining algorithms are needed and desired for usage in the real-life environment. Moreover, from these papers it is clear that some common approaches and techniques of process mining suit the MapReduce model well. Last but not least, analysis of the related work reveal that there are only theoretical considerations of parallel or distributed conformance checking and its usefulness.

7. Conclusions

This paper presents one of the possible ways of speeding up large-scale conformance checking. The paper provides a helicopter-view of distributed conformance checking and suggests ways for possible extensions and improvements. One of the proposed algorithms was implemented and evaluated on event logs, which were different in terms of size and fitness.

As a possible extension, it is worth considering implementing the algorithm using the Spark framework rather than Hadoop because as it is often claimed *Spark* might provide better performance due to its in-memory nature. Furthermore, the *XES* standard which defines how event logs should be structured for convenient process mining, but it seems that the XES standard is not the best option for using with Hadoop. Thus, it is possible to consider other storage formats such as Hadoop sequence files or the *Avro* format.

Acknowledgment

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 15- 37-21103.

References

[1]. Wil M. P. van der Aalst, Process mining: discovery, conformance and enhancement of business processes. Springer, 2011. C. Lattner. LLVM: An Infrastructure for Multi-Stage Optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL.

[2]. S. A. Shershakov and V. A. Rubin, “System runs analysis with process mining,” Modeling and Analysis of Information Systems, vol. 22, no. 6, pp. 818–833, December 2015.
[3]. S. A. Shershakov, “VTMine framework as applied to process mining modeling,” International Journal of Computer and Communication Engineering, vol. 4, no. 3, pp. 166–179, May 2015.
[4]. W. M. van der Aalst, “Process Mining in the Large: A Tutorial,” in Business Intelligence. Springer, 2014, pp. 33–76.
[5]. C. Bratosin, N. Sidorova, and W. van der Aalst, “Distributed Genetic Process Mining,” in Evolutionary Computation (CEC), 2010 IEEE Congress on, 2010, pp. 1–8.
[6]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour,” in Business Process Management Workshops, ser. Lecture Notes in Business Information Processing, N. Lohmann, M. Song, and P. Wohed, Eds. Springer International Publishing, 2014, vol. 171, pp. 66–78.
[7]. A. A. Kalenkova, I. A. Lomazova, and W. M. P. van der Aalst, “Process Model Discovery: A Method Based on Transition System Decomposition,” in Petri Nets, ser. Lecture Notes in Computer Science, vol. 8489. Springer, 2014, pp. 71–90.
[8]. D. Fahland and W. M. P. van der Aalst, “Model Repair - Aligning Process Models to Reality,” Inf. Syst., vol. 47, pp. 220–243, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.is.2013.12.007
[9]. I. S. Shugurov and A. A. Mitsyuk, “Iskra: A Tool for Process Model Repair,” Proceedings of the Institute for System Programming, vol. 27, no. 3, pp. 237–254, 2015.
[10]. J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492
[11]. W. M. P. van der Aalst, “Distributed Process Discovery and Conformance Checking,” in Fundamental Approaches to Software Engineering, ser. Lecture Notes in Computer Science, J. de Lara and A. Zisman, Eds. Springer Berlin Heidelberg, 2012, vol. 7212, pp. 1–25.
[12]. A. Adriansyah, “Aligning Observed and Modeled Behavior,” PhD Thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2014.
[13]. A. Adriansyah, B. van Dongen, and W. M. van der Aalst, “Conformance Checking using Cost-Based Fitness Analysis,” in IEEE International Enterprise Computing Conference (EDOC 2011), C. Chi and P. Johnson, Eds. IEEE Computer Society, 2011, pp. 55–64.
[14]. D. Miner and A. Shook, MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems, 1st ed. O’Reilly Media, Inc., 2012.
[15]. W. M. P. van der Aalst, “Decomposing Petri Nets for Process Mining: A Generic Approach,” Distributed and Parallel Databases, vol. 31, no. 4, pp. 471–507, 2013.
[16]. J. Munoz-Gama, “Conformance checking and diagnosis in process mining,” PhD Thesis, Universitat Politecnica de Catalunya, 2014.
[17]. W. M. P. van der Aalst, “Decomposing Process Mining Problems Using Passages,” in Application and Theory of Petri Nets, ser. Lecture Notes in Computer Science, S. Haddad and L. Pomello, Eds. Springer Berlin Heidelberg, 2012, vol. 7347, pp. 72–91.
[18]. J. Munoz-Gama, J. Carmona, and W. M. van der Aalst, “Single-Entry Single-Exit Decomposed Conformance Checking,” Information Systems, vol. 46, pp. 102–122, 2014.
[19]. “Apache hadoop,” http://hadoop.apache.org/, accessed: 2016-04-01.

[20]. W. M. P. van der Aalst and. B. van Dongen, C. Gunther, A. Rozinat, ¨ E. Verbeek, and T. Weijters, “ProM: The Process Mining Toolkit,” in Business Process Management Demonstration Track (BPMDemos 2009), ser. CEUR Workshop Proceedings, A. Medeiros and B. Weber, Eds., vol. 489. CEUR-WS.org, 2009, pp. 1–4.
[21]. “Prom framework,” http://www.promtools.org/doku.php, accessed: 2016-04-01.
[22]. IEEE Task Force on Process Mining, “XES Standard Definition,” www.xes-standard.org, 2013.
[23]. “Apache mahout,” http://mahout.apache.org/, accessed: 2016-04-01.
[24]. “Amazon EMR,” https://aws.amazon.com/ru/elasticmapreduce/, accessed: 2016-04-01.
[25]. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, “Workflow Patterns,” Distrib. Parallel Databases, vol. 14, no. 1, pp. 5–51, Jul. 2003. [Online]. Available: http: //dx.doi.org/10.1023/A:1022883727209
[26]. I. S. Shugurov and A. A. Mitsyuk, “Generation of a Set of Event Logs with Noise,” in Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88–95.
[27]. 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.
[28]. J. Evermann, “Scalable Process Discovery using Map-Reduce,” IEEE Transactions on Services Computing, vol. PP, no. 99, pp. 1–1, 2014.
[29]. J. Evermann and G. Assadipour, “Big Data meets Process Mining: Implementing the Alpha Algorithm with Map-Reduce,” in Proceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, 2014, pp. 1414–1416.
[30]. W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster, “Workflow Mining: Discovering Process Models from Event Logs,” IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 9, pp. 1128–1142, 2004.
[31]. A. J. M. M. Weijters and J. T. S. Ribeiro, “Flexible Heuristics Miner (FHM),” in Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on, April 2011, pp. 310–317.
[32]. S. Hernandez, S. Zelst, J. Ezpeleta, and W. M. P. van der Aalst, “Handling big (ger) logs: Connecting ProM 6 to Apache Hadoop,” in Proceedings of the BPM2015 Demo Session, ser. CEUR Workshop Proceedings, vol. 1418, 2015, pp. 80–84.
[33]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering Block-Structured Process Models from Incomplete Event Logs,” in Application and Theory of Petri Nets and Concurrency, ser. Lecture Notes in Computer Science, G. Ciardo and E. Kindler, Eds. Springer.

Применение MapReduce для проверки соответствия моделей процессов и логов событий

И.С. Шугуров <shugurov94@gmail.com>
А.А. Мицюк < amitsyuk@hse.ru >
Национальный Исследовательский Университет «Высшая Школа Экономики», Лаборатория процессно-ориентированных информационных систем, ул. Мясницкая, д. 20, 101000, г. Москва, Россия.

Аннотация. Process mining – это относительно новая область исследований, в рамках которой разрабатываются методы исследования и улучшения бизнес-процессов. Спецификой методов process mining является то, что они основываются на анализе истории выполнения процессов, которая представляется в виде логов событий. Проверка соответствия моделей процессов и логов событий является одним из ключевых направлений в области process mining. Алгоритмы проверки соответствия используются для того, чтобы оценить, насколько хорошо данная модель бизнес-процесса, представленная, например, в виде сети Петри, описывает поведение, записанное в логе событий. Проверка соответствия, базирующаяся на использовании так называемых "выравниваний", на данный момент является самым передовым и часто используемым алгоритмом проверки соответствия. В данной работе рассматривается проблема большой вычислительной сложности данного алгоритма. В настоящее время проверка соответствия на основе выравниваний является не слишком эффективной с точки зрения потребления памяти и времени, необходимого для вычислений. Решение этой проблемы имеет большое значение для успешного применения проверки соответствия между реальными моделями бизнес-процессов и логами событий, что весьма проблематично с использованием существующих подходов. MapReduce является популярной моделью параллельных вычислений, которая упрощает реализацию эффективных и масштабируемых распределенных вычислений. В данной работе представлена модифицированная версия алгоритма проверки соответствия на основе выравниваний с применением MapReduce. Так же в работе показано, что проверка соответствия может быть распределена с помощью MapReduce, и что такое распределение может привести к уменьшению времени, требуемого для вычислений. Показано, что алгоритм проверки соответствия модели процесса и лога событий может быть реализован в распределенном виде с помощью MapReduce. Показано, что время вычисления растет линейно с ростом размера логов событий.

Ключевые слова: process mining; conformance checking; MapReduce; Hadoop; big data.

DOI: 10.15514/ISPRAS-2016-28(3)-7

Для цитирования: Шугуров И.С., Мицюк А.А. Применение MapReduce для проверки соответствия моделей процессов и логов событий. Труды ИСП РАН, том 1, вып. 3, 2016 г. стр. 103-122 (на английском). DOI: 10.15514/ISPRAS-2016-28(3)-7.

Список литературы

[1]. Wil M. P. van der Aalst, Process mining: discovery, conformance and enhancement of business processes. Springer, 2011. C. Lattner. LLVM: An Infrastructure for Multi-Stage Optimization. Master’s thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL.
[2]. S. A. Shershakov and V. A. Rubin, “System runs analysis with process mining,” Modeling and Analysis of Information Systems, vol. 22, no. 6, pp. 818–833, December 2015.
[3]. S. A. Shershakov, “VTMine framework as applied to process mining modeling,” International Journal of Computer and Communication Engineering, vol. 4, no. 3, pp. 166–179, May 2015.
[4]. W. M. van der Aalst, “Process Mining in the Large: A Tutorial,” in Business Intelligence. Springer, 2014, pp. 33–76.
[5]. C. Bratosin, N. Sidorova, and W. van der Aalst, “Distributed Genetic Process Mining,” in Evolutionary Computation (CEC), 2010 IEEE Congress on, 2010, pp. 1–8.
[6]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour,” in Business Process Management Workshops, ser. Lecture Notes in Business Information Processing, N. Lohmann, M. Song, and P. Wohed, Eds. Springer International Publishing, 2014, vol. 171, pp. 66–78.
[7]. A. A. Kalenkova, I. A. Lomazova, and W. M. P. van der Aalst, “Process Model Discovery: A Method Based on Transition System Decomposition,” in Petri Nets, ser. Lecture Notes in Computer Science, vol. 8489. Springer, 2014, pp. 71–90.
[8]. D. Fahland and W. M. P. van der Aalst, “Model Repair - Aligning Process Models to Reality,” Inf. Syst., vol. 47, pp. 220–243, 2015. [Online]. Доступно по ссылке: http://dx.doi.org/10.1016/j.is.2013.12.007.
[9]. I. S. Shugurov and A. A. Mitsyuk, “Iskra: A Tool for Process Model Repair,” Proceedings of the Institute for System Programming, vol. 27, no. 3, pp. 237–254, 2015.
[10]. J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008. [Online]. Доступно по ссылке: http://doi.acm.org/10.1145/1327452.1327492.
[11]. W. M. P. van der Aalst, “Distributed Process Discovery and Conformance Checking,” in Fundamental Approaches to Software Engineering, ser. Lecture Notes in Computer Science, J. de Lara and A. Zisman, Eds. Springer Berlin Heidelberg, 2012, vol. 7212, pp. 1–25.
[12]. A. Adriansyah, “Aligning Observed and Modeled Behavior,” PhD Thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2014.
[13]. A. Adriansyah, B. van Dongen, and W. M. van der Aalst, “Conformance Checking using Cost-Based Fitness Analysis,” in IEEE International Enterprise Computing Conference (EDOC 2011), C. Chi and P. Johnson, Eds. IEEE Computer Society, 2011, pp. 55–64.
[14]. D. Miner and A. Shook, MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems, 1st ed. O’Reilly Media, Inc., 2012.
[15]. W. M. P. van der Aalst, “Decomposing Petri Nets for Process Mining: A Generic Approach,” Distributed and Parallel Databases, vol. 31, no. 4, pp. 471–507, 2013.
[16]. J. Munoz-Gama, “Conformance checking and diagnosis in process mining,” PhD Thesis, Universitat Politecnica de Catalunya, 2014.

[17]. W. M. P. van der Aalst, “Decomposing Process Mining Problems Using Passages,” in Application and Theory of Petri Nets, ser. Lecture Notes in Computer Science, S. Haddad and L. Pomello, Eds. Springer Berlin Heidelberg, 2012, vol. 7347, pp. 72–91.

[18]. J. Munoz-Gama, J. Carmona, and W. M. van der Aalst, “Single-Entry Single-Exit Decomposed Conformance Checking,” Information Systems, vol. 46, pp. 102–122, 2014.

[19]. “Apache hadoop,” доступно по ссылке: http://hadoop.apache.org/, 2016-04-01.

[20]. W. M. P. van der Aalst and. B. van Dongen, C. Gunther, A. Rozinat, ¨ E. Verbeek, and T. Weijters, “ProM: The Process Mining Toolkit,” in Business Process Management Demonstration Track (BPMDemos 2009), ser. CEUR Workshop Proceedings, A. Medeiros and B. Weber, Eds., vol. 489. CEUR-WS.org, 2009, pp. 1–4.

[21]. “Prom framework,” доступно по ссылке: http://www.promtools.org/doku.php, 2016-04-01.

[22]. IEEE Task Force on Process Mining, “XES Standard Definition,” www.xes-standard.org, 2013.

[23]. “Apache mahout,” доступно по ссылке: http://mahout.apache.org/,: 2016-04-01.

[24]. “Amazon EMR,” доступно по ссылке: https://aws.amazon.com/ru/elasticmapreduce/, accessed: 2016-04-01.

[25]. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, “Workflow Patterns,” Distrib. Parallel Databases, vol. 14, no. 1, pp. 5–51, Jul. 2003. [Online]. Доступно по ссылке: http: //dx.doi.org/10.1023/A:1022883727209

[26]. I. S. Shugurov and A. A. Mitsyuk, “Generation of a Set of Event Logs with Noise,” in Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88–95.

[27]. 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.

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

[29]. J. Evermann and G. Assadipour, “Big Data meets Process Mining: Implementing the Alpha Algorithm with Map-Reduce,” in Proceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, 2014, pp. 1414–1416.

[30]. W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster, “Workflow Mining: Discovering Process Models from Event Logs,” IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 9, pp. 1128–1142, 2004.

[31]. A. J. M. M. Weijters and J. T. S. Ribeiro, “Flexible Heuristics Miner (FHM),” in Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on, April 2011, pp. 310–317.

[32]. S. Hernandez, S. Zelst, J. Ezpeleta, and W. M. P. van der Aalst, “Handling big (ger) logs: Connecting ProM 6 to Apache Hadoop,” in Proceedings of the BPM2015 Demo Session, ser. CEUR Workshop Proceedings, vol. 1418, 2015, pp. 80–84.

[33]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering Block-Structured Process Models from Incomplete Event Logs,” in Application and Theory of Petri Nets and Concurrency, ser. Lecture Notes in Computer Science, G. Ciardo and E. Kindler, Eds. Springer.