# Distributed Compliance Monitoring of Business Processes over MapReduce Architectures

Daniela Loreti
Università di Bologna, Italy
daniela.loreti@unibo.it

Federico Chesani
Università di Bologna, Italy
federico.chesani@unibo.it

Anna Ciampolini
Università di Bologna, Italy
anna.ciampolini@unibo.it

Paola Mello
Università di Bologna, Italy
paola.mello@unibo.it

## ABSTRACT

In the era of IoT, large volumes of event data from different sources are collected in the form of streams. As these logs need to be online processed to extract further knowledge about the underlying business process, it is becoming more and more important to give support to run-time monitoring. In particular, increasing attention has been turned to conformance checking as a way to identify when a sequence of events deviates from the expected behavior. Albeit rather straightforward on a small log file, conformance verification techniques may show poor performance when dealing with big data, making increasingly attractive the possibility to improve scalability through distributed computation. In this paper, we adopt a previously implemented framework for compliance verification (which provides a high-level logic-based notation for the monitoring specification) and we show how it can be efficiently distributed on a set of computing nodes to support scalable run-time monitoring when dealing with large volumes of event logs.

## Keywords

Business Process Management; Conformance checking; Distributed monitoring; MapReduce

## 1. INTRODUCTION

Digital event data is the new crucial raw material for business. Over the last decade, the management of process execution quality has gained increasing interest, giving birth to a novel research area called Business Process Management (BPM) and, in particular, to a set of log analysis techniques commonly addressed with the name of *process mining*. As stated in the Process Mining manifesto [27], this field spans process discovery, conformance checking, predictive analytics, process optimization and many other techniques that start from the observation of a collection of occurred events (e.g., logged events) to extract further critical knowledge about the evolution of the business process. In particular, conformance checking (i.e., detecting when a sequence of events deviates from the expected behavior) is a crucial activity for business because it can reveal the need for a better control of the process (in case of unwanted deviations) or a refinement/update of the model (to include desirable circumstances not yet foreseen).

In the era of Internet of Things (IoT) and big data, log events are collected from different sources for online stream processing. Indeed, each log event may appear irrelevant when considered by itself but can provide a larger comprehension of the system evolution when combined with other events in the stream. This makes the development of tools that support run-time monitoring of big data more and more important.

Business processes typically involve dynamic and complex interconnected environments demanding a highly expressive notation to represent the behavioral model. For this reason, over the last years, several approaches for process model definition have been proposed. Some of these involve procedural techniques (providing a detailed and complete description of any possible behaviour of the overall system at design time)[26, 28, 29, 2], while some others are declarative approaches (focusing on the elicitation of the set of desired behavioural properties that the overall system should exhibit during its execution, expressed by rules, logical assertions or constraints) [24, 32]. Additionally, some hybrid solutions have been proposed [3] that combine procedural and declarative notations in order to take advantage of both the approaches. The highly expressive power of these notations, increases the complexity of the overall conformance checking system and exacerbate the need for high performance tools.

We can therefore identify two crucial and contrasting features that a conformance checking system for business process must provide: (i) high expressiveness of the notation, to provide a rich representation of the behavioral model; (ii) high performance and scalability, in order to support run-time monitoring of event streams from the business environment.

Aiming to provide (i), in this work, we adopt the SCIFF framework [4], a logic-based proof procedure that has been previously applied to the monitoring of various systems and environments [12, 11]. Focusing on this tool brings us two main advantages. First, thanks to its highly expressive notation, SCIFF is able to operate with both procedural as well as declarative formalisms to express the behavioural model.



*ICPE '17 Companion, April 22-26, 2017, L'Aquila, Italy*



DOI: http://dx.doi.org/10.1145/3053600.3053616

Second, being initially developed for the purpose of run-time checking the compliance of agents to interaction protocols, SCIFF is particularly suitable for the basis of a monitoring system that analyse log data as they are collected.

As regards feature (ii), "analysis techniques dealing with big data need to resort to distributed computing" [30]. Indeed, the only way to handle computationally intensive applications over large volumes of data (while providing an answer within acceptable time) is to distribute the analysing software over a network of computing nodes (e.g., trough the adoption of multicore systems, grid computing or virtualization in cloud environment). According to Van der Aalst [30], there are two different ways to distribute a conformance checking task over a network of nodes: *vertical distribution* and *horizontal distribution*.

In this work, we aim to leverage these distribution techniques to parallelize SCIFF execution over a collection of computing nodes, thus providing a first sketch of conformance checking system that shows both expressiveness and scalability features.

The remain of this paper is structured as follows. Section 2 details the architecture of the proposed monitoring system. Sections 3 and 4 provide further insight of the implemented vertical and horizontal partitioning respectively. Related work and conclusions follows.

## 2. THE MONITORING ARCHITECTURE

In order to provide high expressiveness to business process definition, we employ the SCIFF language. Indeed, it has been shown that various expressive declarative [24, 12] as well as procedural [18, 11] approaches to business model definition can be translated into the SCIFF formalization (i.e., into a collection of declarative constraints that must be satisfied by every event log in the input). In general, we focus on constraint-based languages and we assume (as in the work of Ly et al. [21]) that it is possible to define the model in terms of a collection of rules and constraints (which must be fulfilled by the correspondent event log data). In particular, we employ the SCIFF language to specify these constraints.

The desirable scalability feature is realized through the adoption of a distributed architecture and the application of vertical/horizontal parallelization methods. In the following, we show that SCIFF is particularly suitable for the decomposition of the model requested by these parallelization approaches.

Fig. 1 illustrates the basic components of our monitoring architecture. The event streams and the behavioural model are distributed over the network of nodes according to the prescriptions of V/H Partitioner. This component relays on a MapReduce [13] distributed platform that coordinates the nodes and distributes the data according to the vertical or horizontal partitioning algorithm. MapReduce is a well know and widespread programming model for distributed computation, that constitutes the basis of several engines for big data and stream processing. Finally, Fig. 1 shows that the SCIFF framework is executed on each node for conformance checking purpose.

### 2.1 The SCIFF Monitoring Framework

The SCIFF constraint abductive logic programming framework [4] is an extension of Fung and Kowalski's IFF proof-procedure for abductive logic programming [16]. In addition

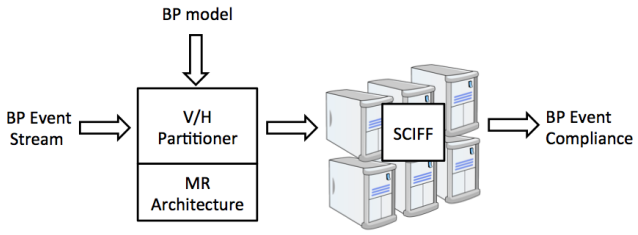


Figure 1: General architecture of the monitoring framework.

to the general notion of abducible, the SCIFF framework also provides the concepts of *happened event*, *expectation*, and *compliance* of an observed sequence of events with a set of expectations. These notions make the SCIFF particularly suitable for dealing with conformance checking of event logs.

In SCIFF formalization, the events are represented as **H** atoms, whereas expectations are modeled by **E** atoms. The following form

$$\mathbf{H}(Ev, T) \tag{1}$$

is a ground atom signifying that an event $Ev$ **H**appens (i.e., occurs) at time $T$. Differently, $\mathbf{E}(Ev, T)$ denotes that an event unifying with $Ev$ is **E**xpected to occur at some time in the range indicated by $T$.

A SCIFF program is composed of a knowledge base $\mathcal{KB}$, a set of integrity constraints $\mathcal{IC}$ and a goal. $\mathcal{KB}$ contains a collection of backward rules in the form: $head \leftarrow body$, while $\mathcal{IC}$ is made of forward implications like: $body \rightarrow head$. The integrity constraints are considered as reactive rules i.e., when the body of an implication becomes true (because of the occurrence of the involved events), then the rule fires and the expectations in the head are generated with an abductive process. Consider the example $\mathbf{H}(a, T) \rightarrow \mathbf{EN}(b, T')$. This defines a constraint between events $a$ and $b$: if $a$ happens at time $T$, then $b$ should not occur at any time $T'$. Informally, SCIFF supports a notion of compliance in terms of expectations and happened events: a sequence of happened events is compliant with a model if for every expected event (**E**) there is indeed a corresponding happened event (**H**). Expectations are generated as the consequence of the triggering of the integrity constraints, that in turn are activated by the happened events. To further clarify the basic concepts of SCIFF, we provide two simple examples of workflow and their translation into SCIFF constraints. The model illustrated in Fig. 2 states that "if an activity $a$ is detected, $b$ should follow" and "if an activity $b$ is detected, $c$ should follow". This can be translated into the statements:

$$\begin{aligned} \mathbf{H}(a, T_a) &\rightarrow \mathbf{E}(b, T_b) \wedge T_b > T_a \\ \mathbf{H}(b, T_b) &\rightarrow \mathbf{E}(c, T_c) \wedge T_c > T_b \end{aligned} \tag{2}$$

A slightly more complex example is shown in Fig. 3 through the formalism by Kumar et al. [18]. Activity A1 addresses the admission of a patient at a hospital, while A2 refers to the detailed collection of information about the patient case. The model suggests that activity A2 follows A1 ("Anamnesis and exams are conducted after the patient admission"), both should last between 5 and 10 time units and it should not

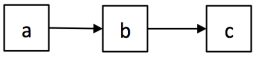


Figure 2: A simple example of workflow model in BPMN language.

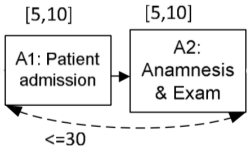


Figure 3: An example of workflow in the formalism of [18].

pass more than 30 time units between the beginning of A1 and the end of A2. As in this case each activity is actually composed of an event start and and event end, the workflow translated into SCIFF constraints results as follows:

$$\begin{aligned}
\mathbf{H}(a1_end, T_{a1_end}) \rightarrow & \mathbf{E}(a2_start, T_{a2_start}) \wedge \\
& \wedge T_{a2_start} > T_{a1_end} \\
\mathbf{H}(a1_start, T_{a1_start}) \rightarrow & \mathbf{E}(a1_end, T_{a1_end}) \wedge \\
& \wedge T_{a1_end} \geq T_{a1_start} + 5 \wedge \\
& \wedge T_{a1_end} \leq T_{a1_start} + 10 \\
\mathbf{H}(a2_start, T_{a2_start}) \rightarrow & \mathbf{E}(a2_end, T_{a2_end}) \wedge \\
& \wedge T_{a2_end} \geq T_{a2_start} + 5 \wedge \\
& \wedge T_{a2_end} \leq T_{a2_start} + 10 \\
\mathbf{H}(a1_start, T_{a1_start}) \rightarrow & \mathbf{E}(a2_end, T_{a2_end}) \wedge \\
& \wedge T_{a2_end} \leq T_{a1_start} + 30
\end{aligned} \tag{3}$$

A complete explanation of SCIFF can be found in [4].

## 2.2 Horizontal and Vertical Distributed Monitoring

In BPM, the concept of *trace* is used to identify a set of events reporting the behaviour (i.e., sequence of carried out activities) of one or more business actors (e.g., users, employees, factory machines, etc.). Given a small log file recording the significant traces happened during a business process, conformance analysis is straightforward. However, in real business cases, the behavioural model may be composed of hundreds of different activities and the log stream may contain millions traces. In this cases, process mining tasks executed on a sigle computing node may be slow in producing meaningful results; while decomposing the conformance checking challenge into smaller problems (that can be distributed on a network of computers), can significantly improve the performance.

As suggested by Van der Aalst [30], there are two basic ways to distribute a conformance checking task:

- *Vertical partitioning.* All nodes receive the complete model and a subset of the whole log. The subset is composed of a collection of complete traces (i.e. each node gets all the events referring to each trace in its subset). In the end the results can to be collected together.
- *Horizontal partitioning.* The traces are partitioned such that, some events of each trace are processed by a node, whereas another part of the same trace is analysed by another node. In this way, each node needs to check all the traces but just focusing on the constraints imposed by a portion of the whole model. In the end the results needs to be merged together.

# 3. VERTICAL DISTRIBUTION

The vertical partitioning suggests to cut the input event log by distributing the traces over the network of computers (i.e., all the events of a trace must be sent to the same node for further processing). Moreover, the business process model that each trace should follow must be distributed to all the computing nodes. Fig. 4 specifies the architecture of our distributed monitoring framework in case of vertical partitioning. We assume that each event in the stream is expressed in the form:

$$\mathbf{H}(tr_{id}, a, ts_a) \tag{4}$$

where, $tr_{id}$ is the unique identifier of the trace, $a$ is the identifier of the activity and $ts_a$ is the timestamp of the event. In other words, the occurrence of a record in the form (4) in the event log means that at time $ts_a$ the system has observed the execution of an activity $a$ of trace $tr_{id}$.

As detailed in Fig. 4, the input stream of events in the form (4) is sent to a "Vertical Partitioner & Dispatcher" component that cuts the stream by trace identifier. It sends all the events referring to a trace (or a set of traces) to the same node in the network of computers.

The conformance of each trace to the business model is checked through the SCIFF framework running on each node. To do so, the SCIFF program takes as input the portion of event stream to check and the whole model. To this end, the business process model (procedural or declarative) formalization must be translated into a collection of integrity constraints expressed through SCIFF language. For example, the model in Fig. 4 is translated into the two constraints of Equation 2. We refer these constraints with *cstr1* and *cstr2* in the following.

In the vertical partitioning all the integrity constraints in the model must be sent to all the computing nodes executing a SCIFF. The output of the overall system is split into files distributed over the network, but can be also collected together in any order at a later time.

The implementation of a monitoring system with vertical partitioning as presented in Fig. 4 on a MapReduce architecture is rather straightforward:

1. the input event stream is processed by a *map* function that extract the important information from each logged record and emits a collection of $(key, value)$ pairs in the form $(trace_{id}, (activity_{id}, timestamp))$;

2. the intermediate pairs are sent to the *reducers* that call the SCIFF program (one compliance checking procedure for each trace).

# 4. HORIZONTAL DISTRIBUTION

The horizontal distribution imposes to partition the logs by activity and set of constraints. In fact, the model is divided into parts that can be concurrently checked by different machines (we will call these parts "*sub-models*" in the following). Thus, each computing node receives just the fraction of each trace that contains the activities in its portion of the model. The mechanism of this distribution is clarified in Fig. 5. Although sometimes counterintuitive (especially when dealing with other business model formalizations), this operation is rather straightforward once the model is represented into the SCIFF formalization. In this language indeed, the expected behaviour of the system is expressed through a set of constraints between the activities. In order to mark a trace $t$ as compliant, all this constraints must be verified by the events in $t$.

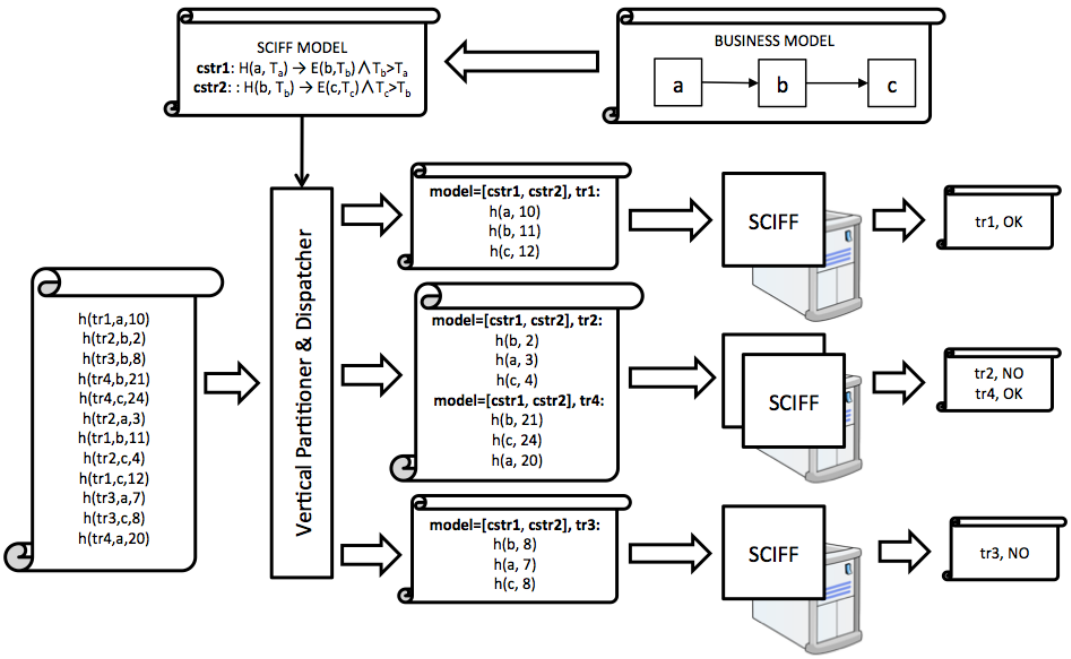


Figure 4: Example of conformance checking executed on a distributed architecture with vertical partitioning.

As shown in Fig. 5, the model partitioning can be realized by simply giving a constraint (or a set of constraints constituting the sub-model) to each computing node. The input event stream is partitioned in the same way: each computing node receives all the events referring to the activities that are present in its sub-model. Note that, this scenario entails the shipping of some events to more than one node, when the correspondent activity is present in more than one constraint. This is typically the case of activities on the border of the sub-model (e.g., activity $B$ in Fig. 5: all the events referring to $B$ are sent to both the nodes).

The results emitted by each SCIFF report the compliance of each trace to just a portion of the initial model, thus requiring a final "AND" operation to merge the results. Indeed, a trace can be considered compliant only if it is compliant to all the identified sub-models.

The implementation on a MapReduce architecture of a monitoring system with horizontal partitioning as presented in Fig. 5 is not so simple as the vertical distribution is. The following steps can be identified:

1. the input event stream is processed by a *flatMap* function that extract the important information from each logged record and adds to that the list of sub-models in which the event is present. This function is also responsible to replicate the events that occur in different sub-models. For example, looking at Fig. 5, the record $h(tr2, event(b), 10)$ generates two records: *(sub_model1, h(tr2, event(b), 10))* and *(sub_model2, h(tr2, event(b), 10))* because the activity $B$ is present in both the sub-models;

2. the emitted records goes to the *map* function that extracts the sub-model identifier and the trace identifier from each record. The combination of these two information creates the keys of the emitted pairs. So, the resulting $(key, value)$ pair is in the form

$$((sub\_model_{id}, trace_{id}), (activity_{id}, timestamp)) \quad (5)$$

For example the previous records become: *((sub_model1, tr2), (event(b), 10))* and *((sub_model2, tr2), (event(b), 10))*;

3. the pairs are sent to the *reducers* that call the SCIFF program. Therefore, the resulting system launches one SCIFF compliance checking procedure for each couple $(sub\_model_{id}, trace_{id})$.

## 5. RELATED WORK

In this work, we propose a distributed monitoring system for business process conformance checking. The logged events coming from various sources are partitioned over a network of computing nodes in two main ways as suggested in [30]. The architecture leverages a previously implemented proof-procedure called SCIFF [4] and a well known programming model for distributed computation, called MapReduce [13]. Considering the definition proposed by Leucker and Schallhart [19], the work at hand focuses on *runtime verification* because it only deals with the detection of violations or satisfactions of correctness properties.

As pointed out in [31], MapReduce looks particularly suitable for the implementation of distributed process mining algorithms. Indeed, some well known process mining algorithms have been already translated into MapReduce implementations [33, 15, 14, 17, 25]. While all these works deal with process and event correlation discovery, our contribution only focuses on event logs in order to run-time identify deviations from a predefined behavioural model. Nevertheless, the importance of compliance monitoring in business is ascertained and further underlined by the plenty of academic and industrial research in this field [21]. Furthermore, differently form [33, 15, 14, 17, 25], we do not focus on a particular algorithm, but we make an effort toward a general way to execute a conformance checking framework on a MapReduce distributed architecture. In terms of implementation, a simple conformance checking task can be easily

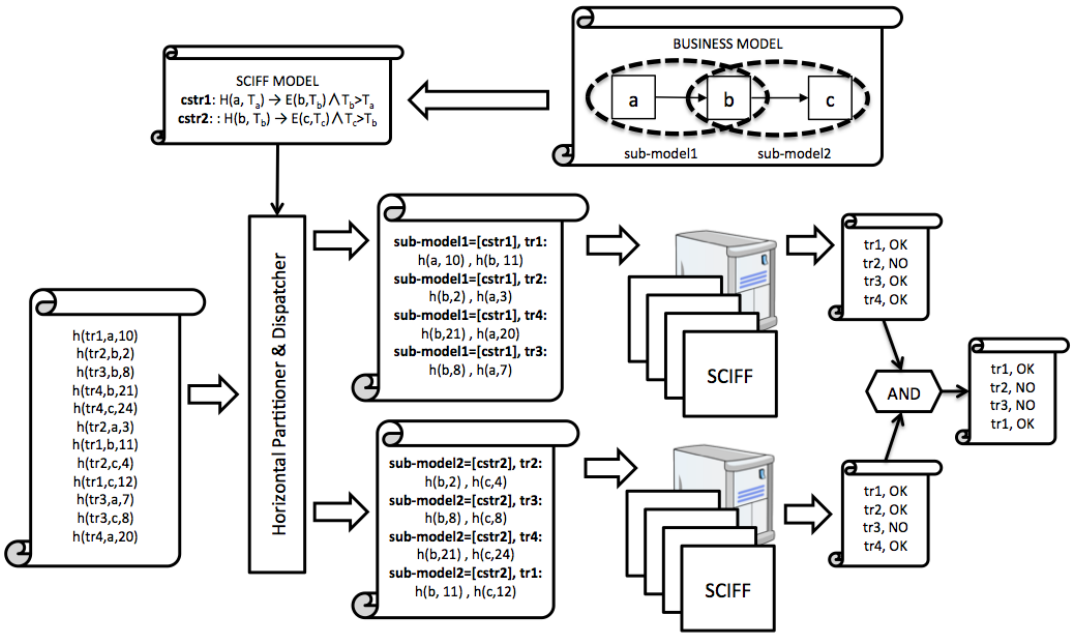


Figure 5: Example of conformance checking executed on a distributed architecture with horizontal partitioning.

translated into a Map function [31], but the conversion of more complex checks in terms of map and reduce functions may result challenging.

Another relevant contribution in the field of conformance checking over MapReduce has been brought by the work of Basin et al. [8]. The authors propose two log partitioning techniques, one based on trace identifier (similar to our MapReduce implementation of vertical distribution) and one based on time slices and volume of data. The works of Barre et al. [7] and Bianculli et al. [9] focus on MapReduce distribution of a conformance checking task expressed through temporal logic. Differently from our solution, they leverage an iterative MapReduce algorithm. In [10], the authors provide a centralized solution for the monitoring of distributed MapReduce application based on a simple set of behavioural declarative properties. Contrarily, the aim of this work is to exploit a distributed MapReduce architecture for the monitoring of business process compliance. An open issue in the field of business process over MapReduce is the problem of load balancing. Indeed, as suggested by different surveys [6, 31], the overall performance of MapReduce depends on data balancing and, ultimately, on the cardinality distribution of the extracted keys. Launching a reduce task for each $trace_{id}$ and for each couple ($sub\_model_{id}$, $trace_{id}$) (in vertical and horizontal partitioning respectively), we aim to make the distribution of the keys (and load) smoother among the computing nodes.

A further dimension that should be considered is about the adopted formalism for defining when a trace is compliant, together with the reasoning tool. A number of different approaches are available in literature [22, 20, 5]. In particular, approaches [22] based on Linear Temporal Logic (LTL) have been investigated within the compliance setting of finite traces. Relations between the SCIFF framework and LTL for compliance have been examined in depth in [23]. Complex Event Processing (CEP) frameworks (able to refine and combine low-level events into more complex, higher-level events [5]) can support compliance monitoring in all those systems dealing with real-time analysis of large-scale streams of events. A comparative study of the formalisms that can be applied to compliance monitoring is out of the scope of this work.

## 6. CONCLUSION AND FUTURE WORK

In this contribution, we present a first attempt to enhance the performance of a business process compliance monitoring tool through the adoption of a distributed architecture. In particular, we focus on the execution of SCIFF proof procedure over a MapReduce environment. We tested the proposed architecture on a network of 5 virtual computing nodes (each with 2 VCPU and 4 GB of dedicated RAM) organized as a Apache Spark [1] MapReduce cluster .The preliminary results of compliance verification over 25 GB of synthetically generated event logs show the expected increase in performance when compared to the execution of SCIFF framework on a single node architecture. The artificial traces have been generated leveraging the SCIFF itself abduction capabilities.

For the future, we plan to study the scalability of our solution when increasing the volumes of log data stream and the dimension of the virtual cluster. Thus to clarify the shortcoming that can derive from MapReduce shuffle phases in both vertical and horizontal partitioning scenarios. Similar tests will be carried out over real life business process data, aiming to compare the resulting performance measurements with those of other works in this field [8, 7, 9].

## 7. REFERENCES

[1] Apache Spark. http://spark.apache.org.
[2] BPEL. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.
[3] BPMN. http://www.bpmn.org/.
[4] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction

in abductive logic programming: The SCIFF framework. *ACM Trans. Comput. Log.*, 9(4), 2008.

[5] A. Artikis, A. Skarlatidis, F. Portet, and G. Paliouras. Logic-based event recognition. *Knowledge Eng. Review*, 27(4):469–506, 2012.

[6] A. Azzini and E. Damiani. Process mining in big data scenario. In *Proceedings of SIMPDA 2015*, volume 1527, pages 149–153, 2015.

[7] B. Barre, M. Klein, M. Soucy-Boivin, P. Ollivier, and S. Hallé. Mapreduce for parallel trace validation of LTL properties. In *Runtime Verification, RV 2012*, volume 7687 of *LNCS*, pages 184–198. Springer, 2012.

[8] D. A. Basin, G. Caronni, S. Ereth, M. Harvan, F. Klaedtke, and H. Mantel. Scalable offline monitoring of temporal specifications. *Formal Methods in System Design*, 49(1-2):75–108, 2016.

[9] D. Bianculli, C. Ghezzi, and S. Krstic. Trace checking of metric temporal logic with aggregating modalities using mapreduce. In *Software Engineering and Formal Methods - 12th International Conference, SEFM 2014*, volume 8702 of *LNCS*, pages 144–158. Springer, 2014.

[10] F. Chesani, A. Ciampolini, D. Loreti, and P. Mello. Process mining monitoring for map reduce applications in the cloud. In *Proceedings of CLOSER 2016*, volume 1, pages 95–105, 2016.

[11] F. Chesani, R. De Masellis, C. D. Francescomarino, C. Ghidini, P. Mello, M. Montali, and S. Tessaris. Abducing compliance of incomplete event logs. In *AI*IA 2016: Advances in Artificial Intelligence*, volume 10037 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2016.

[12] F. Chesani, P. Mello, M. Montali, S. Storari, and P. Torroni. On the integration of declarative choreographies and commitment-based agent societies into the SCIFF logic programming framework. *Multiagent and Grid Systems*, 6(2):165–190, 2010.

[13] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[14] J. Evermann. Scalable process discovery using map-reduce. *IEEE Trans. Services Computing*, 9(3):469–481, 2016.

[15] J. Evermann and G. Assadipour. Big data meets process mining: implementing the alpha algorithm with map-reduce. In *Symposium on Applied Computing, SAC 2014*, pages 1414–1416. ACM, 2014.

[16] T. H. Fung and R. A. Kowalski. The Iff Proof Procedure for Abductive Logic Programming. *Logic Programming*, 33(2):151–165, 1997.

[17] S. Hernández, S. J. van Zelst, J. Ezpeleta, and W. M. P. van der Aalst. Handling big(ger) logs: Connecting prom 6 to apache hadoop. In *Proceedings of the BPM Demo Session 2015*, volume 1418 of *CEUR Workshop Proceedings*, pages 80–84, 2015.

[18] A. Kumar, S. R. Sabbella, and R. Barton. Managing controlled violation of temporal process constraints. In *Business Process Management BPM 2015*, volume 9253 of *LNCS*, pages 280–296. Springer, 2015.

[19] M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[20] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[21] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. P. van der Aalst. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Inf. Syst.*, 54:209–234, 2015.

[22] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst. Runtime verification of ltl-based declarative process models. In *Runtime Verification RV 2011*, volume 7186 of *LNCS*, pages 131–146. Springer, 2011.

[23] M. Montali, F. Chesani, M. Gavanelli, E. Lamma, and P. Mello. Evaluating compliance: from LTL to abductive logic programming. In *Proceedings of the 30th Italian Conference on Computational Logic*, volume 1459 of *CEUR Workshop Proceedings*, pages 101–116. CEUR-WS.org, 2015.

[24] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. Declarative workflow. In ter Hofstede et al. [26], pages 175–201.

[25] H. Reguieg, F. Toumani, and H. Nezhad. Using mapreduce to scale events correlation discovery for business processes mining. In *Business Process Management BPM 2012*, volume 7481 of *LNCS*, pages 279–284. Springer, 2012.

[26] A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, and N. Russell, editors. *Modern Business Process Automation - YAWL and its Support Environment*. Springer, 2010.

[27] W. Van Der Aalst, A. Adriansyah, A. K. A. de Medeiros, and F. Arcieri. Process mining manifesto. In *Business Process Management Workshops*. Springer Berlin Heidelberg, 2012.

[28] W. M. P. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.

[29] W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[30] W. M. P. van der Aalst. Distributed process discovery and conformance checking. In *Fundamental Approaches to Software Engineering - FASE 2012*, volume 7212 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2012.

[31] W. M. P. van der Aalst and E. Damiani. Processes meet big data: Connecting data science with process science. *IEEE Trans. Services Computing*, 8(6):810–819, 2015.

[32] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.

[33] W. M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.