A distributed approach to compliance monitoring of business process event streams

A Distributed Approach to Compliance Monitoring of Business Process Event Streams

Daniela Loreti[a,*], Federico Chesani[b], Anna Ciampolini[b], Paola Mello[b]

[a] *CIRI - Health Sciences & Technologies. Via Tolara di Sopra, 41/E, Ozzano dell'Emilia (BO), Italy*
[b] *DISI - Department of Computer Science and Engineering, University of Bologna. Viale del Risorgimento 2, Bologna, Italy*

Abstract

In recent years, the significant advantages brought to business processes by process mining account for its evolution as a major concern in both industrial and academic research. In particular, increasing attention has been turned to compliance monitoring as a way to identify when a sequence of events deviates from the expected behaviour. As we are entering the IoT era, an increasing variety of smart objects can be introduced in business processes (e.g., tags to track products in a plant, smartphones and badge swiping to draw the activities of customers and employees in a shopping center, etc.). All these objects produce large volumes of log data in the form of streams, which need to be run-time analysed to extract further knowledge about the underlying business process and to identify unexpected, non-conforming events.

Albeit rather straightforward on a small log file, compliance verification techniques may show poor performances when dealing with big data and streams, thus calling for scalable approaches.

This work investigates the possibility of spreading the compliance monitoring task over a network of computing nodes, achieving the desired scalability. The monitor is realised through the existing SCIFF framework for compliance checking, which provides a high level logic-based language for expressing the properties to be monitored and nicely supports the partitioning

*Corresponding author

Email addresses: daniela.loreti@unibo.it (Daniela Loreti), federico.chesani@unibo.it (Federico Chesani), anna.ciampolini@unibo.it (Anna Ciampolini), paola.mello@unibo.it (Paola Mello)

of the monitoring task. The distributed computation is achieved through a MapReduce approach and the adoption of an existing general engine for large scale stream processing. Experimental results show the feasibility of the approach as well as the advantages in performance brought to the compliance monitoring task.

Keywords:
Business Process Management, Distributed Compliance Monitoring, Stream Processing, MapReduce

---

1. Introduction

During the last few years, Business Process Management (BPM), a novel research area focused on the management of process execution quality, has gained increasing interest in both industry and academia. In particular, a set of log analysis techniques commonly addressed with the name of *process mining* has made digital event data the new crucial raw material for business. As stated in the Process Mining manifesto [1], this research area includes process discovery, conformance checking, predictive analytics, process optimisation and many other techniques that start from the observation of a set of occurred events to extract further crucial information about the business process evolution.

In particular, detecting when a sequence of events deviates from the expected behaviour is a crucial activity for business. It can reveal unwanted deviations (and the consequent need for a better control of the process) or desirable circumstances not yet foreseen (which require a model refinement/update). In this study, we adopt the nomenclature of the work by Ly et al. [2], which uses the terms *compliance monitoring/online auditing* to address the runtime detection (or prediction) of compliance violations, while the terms *offline/post mortem analysis* are employed for a diagnosis that is conducted after the process execution has concluded.

As BPM typically refers to large interconnected environments, the nature of the concepts and properties that needs to be monitored is often very complex, demanding a highly expressive notation to represent the behavioural model. Many approaches for process model definition have been proposed: some of these involve procedural techniques [3, 4, 5, 6] , while some others are based on more declarative approaches [7, 8] . Additionally, some hybrid

solutions have been proposed [9] that combine procedural and declarative notations in order to take advantage of both the approaches.

Irrespective of the chosen method, BPM techniques ask for model definition languages with a high level of expressiveness. Unfortunately, it is often the case that a highly expressive notation increases the complexity of the overall compliance checking system. Such complexity might become an issue when dealing with huge event logs or high data rate streams, as it is in Internet of Things (IoT) application fields. For example, automated identification and data collection technologies (like for example RFID and sensors) together with online stream auditing, allow the companies to better understand what is actually happening in their business processes (e.g., how the plant is performing, what is the current location of assets and products in the supply chain, etc.) [10, 11]. As the data rate of the log streams produced by these devices can vary over time according to several factors, the demand for runtime compliance analysis further exacerbates the need for high performance infrastructures, making scalability a mandatory requirement for any monitoring approach.

Hence, we can identify two crucial and contrasting features that a compliance monitoring system for business process must provide:

(i) high expressiveness of the notation, to provide a rich set of "building blocks" to represent the behavioural model;

(ii) good performance and scalability, to support runtime monitoring of the event streams coming from the business environment.

In order to provide (i), in this work we adopt the Social Constrained IFF (SCIFF) framework [12], a logic-based proof procedure that has been previously applied to the monitoring of various systems and environments [13, 14]. The adoption of this tool brings us a number of advantages. First, thanks to its highly expressive notation, SCIFF is able to operate with both procedural and declarative formalisms to express the behavioural model. Second, since it was originally conceived for runtime checking of agents behaviour to interaction protocols, it already provides many features required for runtime monitoring (for example, it natively support the concept of time, temporal deadlines and constraints referring to data). Third, the monitoring procedure adopted in SCIFF already enjoys a property of *compositionality*, thus paving the way for distributed computing and scalability.

Traditional approaches to online auditing envisage the execution of a compliance monitoring framework on a single node receiving and analysing all the event logs regarding the business process. The execution of complex conformance verification tasks on a single computing node my experience poor performance. For this reason, in order to provide (*ii*), we consider the adoption of a distributed infrastructure. As pointed out in [15], business process analysis techniques dealing with big data need to resort to distributed computing, through the adoption of multi-core systems, grid computing or virtualisation in cloud environments. Nevertheless, the execution of a compliance monitoring task on a distributed network of computers requires to face additional issue when compared with a single-node execution. First of all, a method is needed to exactly define how the work – in terms of input log and business constraints to be checked – must be subdivided between the nodes. Secondly, a distributed approach needs a mechanism to coordinate the various parts in which the job has been splitted. The need to bridge process mining techniques with big data techniques and infrastructures is clearly highlighted by Van der Aalst and Damiani [16] and further confirmed by various works in this field [17, 18, 19, 20]. However, the adoption of distributed computing frameworks in the field of business process compliance monitoring is still at initial stage and cannot disregard the formal definition of the algorithms and methods applied.

An infrastructure for compliance monitoring should support large-scale distributed stream processing at runtime, while providing scalability and fault tolerance. In this regard, during the last years, MapReduce [21] distributed programming model has gained significant diffusion in the big data research community, primarily due to its simplicity and intrinsic scalability. The latest evolutions of distributed computing frameworks, such as Apache Storm [22], Apache Flink [23] and Apache Spark [24], give support to stream processing, thus simplifying the implementation of efficient applications dealing with big data flows. These frameworks allow the developer to implement different distributed models, included but not limited to, MapReduce, while providing autonomous fault tolerant mechanisms as well as runtime infrastructure scaling capabilities.

In this work (representing an evolution of the preliminary ideas presented in [25]), we investigate the issue of implementing the monitoring task over a cluster of computing nodes. In particular, we generalise *vertical* and *horizontal* partitioning, two techniques for distributing the monitoring task introduced by Van der Aalst in [15]. We prove that, under certain conditions,

the SCIFF framework can suitably support these techniques.

Moreover, we consider the case of online stream auditing, as it represents the most frequent (and challenging) scenario in IoT applications. In particular, we discuss how the MapReduce programming model can be employed to efficiently handle different partitioning techniques at runtime over a distributed stream processing engine.

In summary, the contributions of the work at hand are:

- a detailed description of the proposed distributed system for compliance monitoring as well as of its underlying components;
- a theoretical study of the partitioning methodologies that can be applied in order to subdivide the compliance monitoring task over a set of computing nodes;
- an in-depth investigation of the most interesting partitioning strategies identified as well as a detailed description of their implementation through a MapReduce-oriented algorithm;
- an evaluation of the proposed distributed approach through a comparative performance study of the different partitioning methodologies.

The remain of this paper is structured as follows. Section 2 presents a classification and description of other relevant works in the field of BPM and event stream monitoring. Section 3 provides a first insight of the proposed system architecture as well as details of its founding concepts. In Sections 4 and 5 we focus on the implementation of different partitioning methodologies over a distributed platform for stream processing. Section 6 presents an evaluation of the system performance and scalability. Conclusion follows.

2. Related work

In this work, we propose a distributed system for business process compliance monitoring. This term comes from the detailed survey of Ly et al. [2], but other classifications are possible: according to van der Aalst [26], this work operates on *pre-mortem* input logs (as they are referred to current process instances that are ongoing) with a normative *de jure* model (using constraints to specify how process instances should behave); while, considering the definition proposed by Leucker and Schallhart [27], the conducted log analysis can be addressed with the term *runtime verification*, because it only

deals with the online detection of violations or satisfactions of correctness properties.

Taking inspiration from [15], in our solution, the logged events coming from various sources are online partitioned over a network of computing nodes along two dimensions: the log and the model. The architecture leverages a previously implemented proof-procedure called SCIFF [12] and starts from a well known programming model for distributed computation, called MapReduce [21]. From a practical point of view, Apache Spark [24] is employed in order to give support to large-scale event stream processing.

As this short description underlines, the work at hand collects together different orthogonal fields. So, without claiming to be complete, we can compare it with various approaches classified in three categories:

(i) other significant formalisms to define the behavioural model and approaches tackiling compliance checking;

(ii) alternative approaches dealing with event stream monitoring;

(iii) distributed solutions to the compliance monitoring task.

As regards the adopted formalism for defining when a trace is compliant (i), number of different approaches are available in literature [28, 29, 30, 31, 32].

In particular, approaches [33, 28, 29] based on Linear Temporal Logic (LTL) have been investigated as regards the compliance of finite traces. Differently from the SCIFF approach, the work of Maggi et al. [28] does not support the definition of constraints referring to data and resources involved in the business process. Furthermore, being based on LTL, it supports the definition of temporal relations between events (to capture their ordering into compliance roles), but, differently from SCIFF, cannot express quantitative temporal distances between happening events (e.g., deadlines, delays and latency constraints). Other relations between the SCIFF framework and LTL for compliance have been examined in depth in [34].

The work of Basin et al. [29] propose a runtime verification framework for security policies, specified through the Metric First-Order Temporal Logic (MFOTL), which is a variant of Linear Temporal First-Order Logic (LTL-FO) with metric time. The high expressiveness of this logic allows the user to express compliance rules involving advanced temporal constraints, data- and resource- related conditions. To the best of our knowledge,

this approach still lacks a framework to support the execution of the provided tool [35] on a distributed stream processing environment.

Other formalisms involve control patterns triggered by events [31] and Supervisory Control Theory [32]. Differently from SCIFF, both these approaches cannot model quantitative time constraints. Besides, [32] lacks the possibility to support the definition of constraints involving data.

The second dimension that should be considered addresses all those works that treat compliance monitoring though the management of streams of events (ii). The term *stream data management* refers to the processing of data received as a realtime continuous sequence from one or more sources. As underlined in [2], this research area is relevant for compliance monitoring as regards the two aspects of *collecting* and *querying* the event streams related to a certain process.

As regards event collection, in the last few years, Complex Event Processing (CEP) [36, 37] has been proposed as a way to process a stream of raw data from different sources and extract meaningful events from it. CEP can be a support technology in the field of compliance monitoring for all those systems where large amount of events must be analysed with realtime requirements. The work of Awad et al. [38] for example, offers a framework for the definition of compliance requirements in controlled natural language and take advantage of CEP for the aggregation of significant events.

As regards querying the event stream, some approaches tackle compliance monitoring by analysing the trace of events with special forms of query expressing the compliance rules [39, 40]. These queries are continuously processed on-the-fly involving temporal operators to correlate the events across time. In particular, Mulo et al. [40] propose a method to build a compliance monitoring infrastructure for process-driven SOA. Here, the conditions to be checked over event streams can be filters (which narrow down the number of analysed activities) and assertions (which specify the expected data values in the stream). The work propose a prototype for generating queries in Esper event processing engine [41].

A natural evolution of these CEP querying systems is the deployment of distributed infrastructures able to partition the stream analysis on a collection of nodes [42, 43, 44]. The encouraging results in this field has promoted the usage of cloud environments and the development of simulators to forecast the performance of a given infrastructure when a CEP workload is executed [45].

The third category of approaches related to our work (iii) focuses on

distributed execution environments for compliance checking. One of the most spread practice in this field is the adoption of MapReduce programming model to enable the distribution of a compliance monitoring algorithm over a network of compute nodes. Indeed, as pointed out in [16], MapReduce looks particularly suitable for the implementation of distributed process mining techniques. Some well known algorithms in this field have been already translated into MapReduce implementations. This is the case of the Inductive Miner [46], Alpha Miner [47, 48] and Flexible Heuristic Miner [49]. Moreover, the problem of event correlation discovery has been studied using a MapReduce implementation in the work of Reguieg et al. [17]. The proposed two-stages approach shows good performance gain on both real and synthetic event logs despite the overhead introduced by the shuffle and sort operations executed by the MapReduce framework.

While all these works deal with process and event correlation discovery, our contribution 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 [2]. Furthermore, differently form [46, 47, 48, 49, 17], we do not focus on a particular algorithm, but we make an effort toward a general way to execute a compliance monitoring framework on a MapReduce distributed architecture for stream processing. In terms of implementation, a simple conformance checking task can be easily translated into a Map function [16], but the conversion of more complex checks in terms of map and reduce functions may result challenging: IBM Business Insight Toolkit (BITKit) [50] was a first attempt to automate the distribution of compliance checking tasks over a MapReduce architecture (Apache Hadoop [51]). Indeed, this software allows the user to define her business model in a pseudo-SQL language and make queries that are automatically translated into programs suitable for the execution of a Hadoop infrastructure. However, the expressive power of this tool is restricted to that of a pseudo-SQL language. Thanks to the expressiveness of SCIFF framework, the range of compliance constraints that can be formulated in our system is greatly extended.

Another relevant contribution in the field of compliance monitoring over MapReduce has been brought by the work of Basin et al. [19]. The authors propose two log partitioning techniques: one based on the trace identifier (similar to the MapReduce implementation described later in Section 4.2) and one based on time slices and volume of data.

The works of Barre et al. [20] and Bianculli et al. [52] focus on MapReduce distribution of a conformance checking task expressed through temporal logic. Differently from our solution, they leverage an iterative MapReduce algorithm.

In [53, 54], the authors provide a centralised solution for the monitoring of distributed MapReduce application based on a simple set of behavioural declarative properties. Contrarily, the aim of the work at hand 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 [18, 16], the overall performance of MapReduce depends on data balancing and, ultimately, on the cardinality distribution of the extracted keys. In this work, we aim to make the distribution of the keys (and load) smoother among the computing nodes by proposing a methodology to combine different slicing techniques for the compliance monitoring task.

Finally, it is important to notice that the scalability brought by a MapReduce implementation to a compliance monitoring system can be fully exploited when a large distributed infrastructure is available, which is not always the case, even in modern companies. Fortunately, the advent of cloud computing has brought the possibility to on-demand employ the computing power of large scale data-centers in a transparent way. The work at hand could take advantage of a cloud environment, such that the company can disregard complex infrastructure management tasks [55, 56], and dynamically scale the required computing power, eventually exploiting and combining the big data analytics services offered by different clouds (as in the work [57]). The tight relationship between cloud and business process is also highlighted by the works [58, 59, 60], where a formal approach based on Situation Calculus is employed to translate service requirements into an Intention Workflow Model, used to generate autonomic cloud service composition.

3. Overview of the System Architecture and Background

In Section 1, we introduced the expressiveness of the modelling language and scalability of the overall architecture as two important requirements for any BPM monitoring architecture. In this work, we adopt the SCIFF framework for covering the former requirement: in previous articles we showed that SCIFF is expressive enough to represent well-known formalisms, declarative

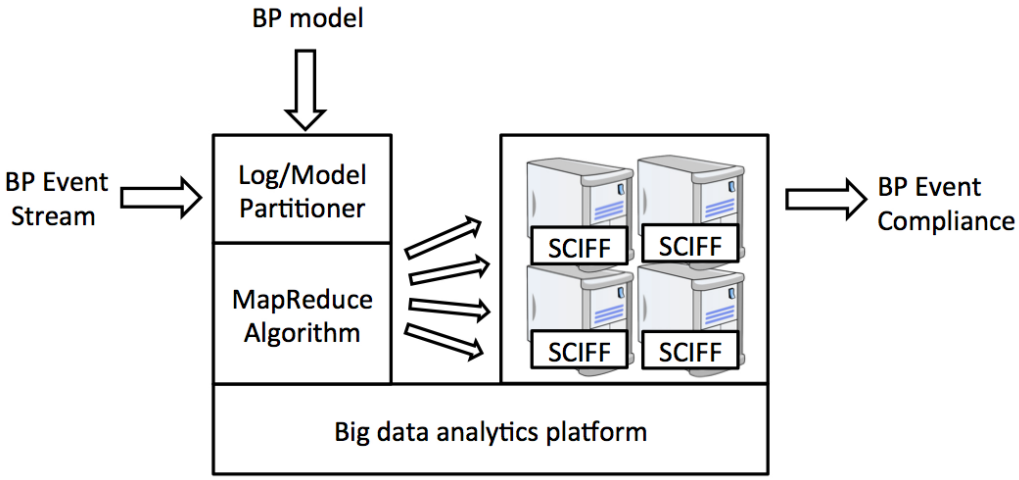


Figure 1: General Architecture of the Monitoring Framework. The platform for big data analytics is employed to enable the implementation of a MapReduce algorithm (which realizes the prescriptions of the Log/Model Partitioner component) and its deployment on a distributed infrastructure. Once the compliance task is correctly subdivided, the system takes advantage of the MapReduce algorithm to activate multiple instances of the SCIFF framework.

[7, 13] as well as procedural [61, 14] ones. The latter requirement is covered through the adoption of a distributed platform for big data analytics and the application of a parallelisation method.

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 Log/Model Partitioner: this component relays on a distributed platform that coordinates the nodes and employs a MapReduce algorithm to subdivide the data according to the partitioning algorithm. The platform is also used to concurrently launch several SCIFF proof procedures over the computing nodes.

In the following we provide further details of the founding elements of the proposed monitoring system: the SCIFF proof procedure and its underlying formalisation (Section 3.1); the MapReduce programming model (Section 3.2); and partitioning approach to compliance monitoring (Section 3.3).

3.1. The SCIFF Monitoring Framework

Compliance monitoring techniques start form the observation of an event log. Each event reported in the log refers to an *activity* – i.e., a well-defined

step in the business process – and a *case* – i.e., an instance of the process. A sequence of events belonging to the same case is called *trace*. In BPM, the concept of trace is often 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.).

The SCIFF constraint abductive logic programming framework [12] is an extension of Fung and Kowalski's IFF proof-procedure for abductive logic programming [62]. In addition to the general notion of abducible, the SCIFF framework also provides the concepts of *happened event*, *positive/negative expectation*, and *compliance* of an observed trace of events with a set of expectations. These notions make SCIFF particularly suitable for dealing with compliance monitoring of event logs.

In SCIFF formalisation, the events are represented as **H** atoms, whereas expectations are modelled by **E**/**EN** atoms. The following form

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

is an 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 a time instant T. Finally, $\mathbf{EN}(Ev, T)$ suggests that *all* the events unifying with Ev are **E**xpected to **N**ever occur at the time instant T. As in Logic Programming [63], terms starting with a capital letter indicates variables. In SCIFF, variables can be also subject to constraints. For example, the following:

$$\mathbf{EN}(a, T) \wedge T \geq 5 \wedge T \leq 10. \tag{2}$$

states that an activity named a is not expected to happen at any time T, with T in the range $[5 \ldots 10]$. Notice that implicitly, this would mean that the activity a is not prohibited outside of the specified temporal interval.

Formally, a SCIFF specification is a triple $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$, where $\mathcal{KB}$ is a knowledge base (a Logic Program as for [63]), $\mathcal{A}$ is a set of abducible predicates (predicates that can be hypothesized: among them, also **E** and **EN** predicates), and $\mathcal{IC}$ is a set of Integrity Constraints (ICs).

The ICs are considered as reactive rules i.e., when the body of an implication becomes true, then the rule fires and the expectations in the head are generated with an abductive process. ICs usually have **H** atoms in the body, and **E**/**EN** atoms in the head. Practically speaking, an IC trigger when events mentioned in the body occur. Consider the example $\mathbf{H}(a, T) \rightarrow$

$\mathbf{EN}(b, T')$. This defines a constraint between the happening of actvitities a and b: if a happens at time T, then b should not occur at any time T'. Obviously, other more complex constraints can be specified. For example, if we want to express that an event b should occur within 300 time units after the happening of another event a, this can be formalized with: $\mathbf{H}(a, T) \rightarrow \mathbf{E}(b, T') \wedge T' > T \wedge T' \leq T + 300$. 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**) and, for every negative expectation **EN**, there is no matching happened event. Expectations are generated as the consequence of the triggering of the ICs, that in turn are activated by the happened events.

To further clarify the SCIFF monitoring framework, we provide two simple examples of workflow and their translation into SCIFF constraint formalization. The model illustrated in Fig. 2 is in BPMN formalism and states that "if an event referring to activity a is detected, b should follow" and "if an event related to activity b is detected, c or d should follow." As the gateway between activities b, c and d is an *exclusive or*, only one of the two events c or d is expected to be found in the trace. This can be translated into the statements:

$$\mathbf{H}(a, T_a) \rightarrow \mathbf{E}(b, T_b) \wedge T_b > T_a. \tag{3}$$

$$\mathbf{H}(b, T_b) \rightarrow (\mathbf{E}(c, T_c) \wedge T_c > T_b) \vee (\mathbf{E}(d, T_d) \wedge T_d > T_b). \tag{4}$$

$$\mathbf{H}(c, \_) \vee \mathbf{H}(d, \_) \rightarrow \bot. \tag{5}$$

Constraint (5) ensures that the trace is considered non compliant if both the two events c and d happens.

A slightly more complex example is shown in Fig. 3 through the formalism described by Kumar et al. [61]. Activity A1 addresses the admission of a patient at a hospital, while A2 refers to the collection of information about the patient case. The model prescribes 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 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 an event end, the

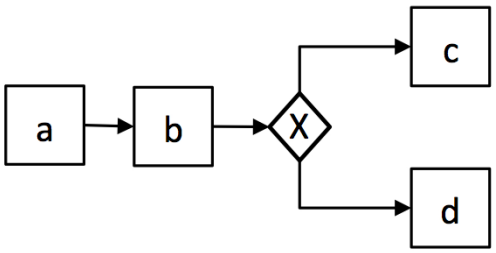


Figure 2: A simple example of workflow expressed in BPMN formalism. The model states that, if an activity unifying with a is detected, b should follow. If an activity b is detected, an activity unifying with c x-or d should follow.

workflow translated into SCIFF constraints results as follows:

$$\mathbf{H}(a1\_end, T_{a1\_end}) \rightarrow \mathbf{E}(a2\_start, T_{a2\_start}) \wedge \wedge\, T_{a2\_start} > T_{a1\_end}. \quad (6)$$

$$\mathbf{H}(a1\_start, T_{a1\_start}) \rightarrow \mathbf{E}(a1\_end, T_{a1\_end}) \wedge \wedge\, T_{a1\_end} \geq T_{a1\_start} + 5 \wedge \wedge\, T_{a1\_end} \leq T_{a1\_start} + 10. \quad (7)$$

$$\mathbf{H}(a2\_start, T_{a2\_start}) \rightarrow \mathbf{E}(a2\_end, T_{a2\_end}) \wedge \wedge\, T_{a2\_end} \geq T_{a2\_start} + 5 \wedge \wedge\, T_{a2\_end} \leq T_{a2\_start} + 10. \quad (8)$$

$$\mathbf{H}(a1\_start, T_{a1\_start}) \rightarrow \mathbf{E}(a2\_end, T_{a2\_end}) \wedge \wedge\, T_{a2\_end} \leq T_{a1\_start} + 30. \quad (9)$$

Equation (6) [1] imposes the sequence between activities A1 and A2. Equations (7) and (8) ensures that the starting of an activity is always followed by its ending. Finally, Equation (9) imposes the temporal constraint between the start of the patient admission (activity A1) and the end of the anamnesis (activity A2).

[1]Notice that, to not confuse activity names with Prolog variables, we substitute upper-case capital letters to the corresponding lower-case letters. E.g., activity A1 depicted in Fig. 2 becomes "a1" in Equations (6)–(9).

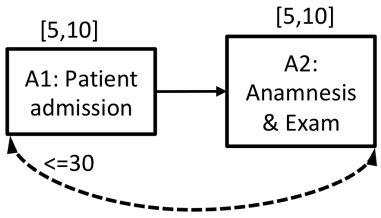


Figure 3: An example of workflow in the formalism of [61]. The model suggests that activity A2 follows A1, both should last between 5 and 10 time units and it should not pass more than 30 time units between the beginning of A1 and the end of A2 . Note that, in this case, each box entails two events: the activity start and activity end.

3.1.1. Temporal Reasoning and Handling of out-of-order events

Out-of-order events are supported by SCIFF by default. Let us suppose that our model is defined by Eq. (3), and that activities a and b are executed in the right order (thus the timestamps satisfy the constraint $T_b > T_a$). Let us suppose also that b is notified to the SCIFF Proof Procedure first, and a is notified later. SCIFF establishes the fulfilment/violation of the expectation generated by Eq. (3) by looking at the timestamps, and ignores when the events have been notified to the proof procedure.

A first remark is about *when* violations are detected. Let us consider again Eq. (3), and let us suppose that a is executed, but the execution of b is not observed: when would it be possible to say that b did not happen, and consequently detect a violation[2]? SCIFF Framework overcomes this issue by supporting a special event *close_history*, whose meaning is that "no more events will ever happen". Upon the observation of such special event, positive expectations that do not have a matching observed event are recognized as violated, while negative expectations (i.e., prohibitions) that do not have a matching observed event are recognized as satisfied.

A second remark is about run-time detection of violations: in several application domains, waiting for a *close_history* event would not be enough, and a more responsive (as soon as possible) detection of violations would be desirable. Let us consider the following IC:

$$\mathbf{H}(a, T_a) \rightarrow \mathbf{E}(b, T_b) \wedge T_b > T_a \wedge T_b \leq T_a + 10 \tag{10}$$

[2]Notice that while we know that $T_b > T_a$, no upper limit has been stated on T_b: thus, activity b is expected at any time in the future.

whose intended meaning is that upon the execution of an activity a, the execution of an activity b is expected to happen after, and within 10 time units from the execution of a. In case a is executed and b is not, a violation would be expected *asap*, i.e. at 11 time units from the observation of a.

The SCIFF framework is not equipped with any sort of "internal clock", but rather relies on special events named *current_time*. Upon the notification of a current_time event, a constraint is added: all time variables must be greater than the timestamp of the current_time event. In other words, the notification of a current_time event tells the SCIFF Proof Procedure that all the following events will happen in the future. Let us consider Eq. (10), and let us suppose that event $\mathbf{H}(a, 27)$ is observed: then, an expectation $\mathbf{E}(b, T_b)$ is generated, with $27 < T_b \leq 37$. When the event $\mathbf{H}(\mathit{current\_time}, 37)$ is notified to the Proof Procedure, a further constraint $T_b > 37$ is added: T_b cannot be at the same time less or equal, and greater than 37, and the expectation is detected as violated.

Summing up, two different behaviours are supported by the SCIFF Framework:

(a) **No current time event is provided**: SCIFF supports events that happened in the right order but are notified out-of-order. The disadvantage is that violation detection is not performed run-time, but possibly only at the notification of the *close_history* event.

(b) **Current time events are generated by an external application**: SCIFF detects violations asap, but out-of-order events are not supported.

A final remark is about the notion of trace, since in the SCIFF events do not have an explicit placeholder for a trace identifier. However, the user is free to provide an event description with more structured information such as, for example, a trace identifier. It would be possible then to both monitor the compliance of each trace (by partitioning the log events by trace identifier and running a *trace-agnostic* SCIFF proof procedure for each) or check constraints among different traces (by wisely partitioning the log into groups of traces and monitor inter-trace constraints, launching a SCIFF for each group). To better clarify this point, consider the example of a compliance monitor that analyses all the loans asked to a bank. A trace represents all the events related to a single loan request from a client. For the granting process, it is important to ensure that: (i) the loan process is compliant with

a model (e.g., all the documentation provided, the solvency probability is over a threshold, etc.); (ii) the bank has not already received another loan request from the same client. Since the SCIFF framework does not require each event to be associated with its trace identifier, it can check constraints (i) by considering the events of a single loan request. At the same time, providing the SCIFF with all the events, each reporting the information of the related trace, it is also able to check the inter-trace constraint (ii).

3.2. The MapReduce programming model

MapReduce is a well know and widespread programming model for distributed computation, which allows the programmer to control the complexity of parallelisation.

According to the MapReduce approach, a large input dataset can be partitioned into an arbitrary number of parts, each exclusively processed by a different computing task, the *mapper*. Each mapper produces intermediate results (in the form of *key/value* pairs) that are collected and processed by other tasks, called *reducers*, in charge of calculating the final results by merging the values associated to the same key.

The developer is only asked to reformulate her software in terms of these two functions *map* and *reduce*. Indeed, the programs implemented according to the MapReduce model can be automatically parallelised and easily executed on a distributed infrastructure.

Over the last few years, several platforms for MapReduce and big data analytics in general have been proposed [64]. The aim of these platforms is to provide the developer with a distributed infrastructure able to automatically spread the tasks across the computing nodes, detect and recover from failures, supply mechanisms to scale-up/-down the infrastructure (i.e., adding/removing nodes) when variations in the computing power are needed (e.g., to face an increase of the input data rate, or a strict deadline when analysing large batches of data).

These features offered by the distributed platforms make the employment of MapReduce programming model the best candidate to satisfy the performance and scalability requirement of our the online compliance monitor.

3.3. Partitioning the compliance monitoring task

Given a small log file recording the significant traces happened during a business process, compliance 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 of events and traces. In this cases, process mining tasks executed on a single computing node may be slow in producing meaningful results; while decomposing the compliance checking task into smaller problems (that can be distributed on a network of computers), can significantly improve the performance.

Van der Aalst [15] suggests two basic ways to distribute a conformance checking task depending on the slicing technique applied to the event log:

- *Vertical partitioning.* The log is partitioned considering the trace identifier of each event so that the events are grouped into their corresponding traces. The model instead is not partitioned. Thus, each computing node receives the complete model and a subset of the whole log (i.e., all the events referring to one or more traces).

- *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.

Finally, in both partitioning cases, the results from each node are collected together, and the event log is considered compliant w.r.t. the model by applying a function to the partial results. In our vision, vertical and horizontal methods can be combined together, such that each node checks a subset of all the traces (by trace identifier) against a subset of all the constraints that compose the model. In practice, Van der Aalst's classification of log slicing methodologies can be generalised considering two different dimensions for partitioning: the event log and the business model (see Table 1).

Log partitioning refers to the possibility to cut the event log into groups of traces (similarly to vertical partitioning), whose compliance with the model can be checked on different computing nodes. Obviously, different degrees of partitioning are possible in this case, from one – i.e., the log is not divided – to the number of traces in the log – i.e., each node is responsible for the compliance monitoring of one trace: *Fine-grain Log* partitioning.

On the other dimension, *model partitioning* prescribe to cut the model into a number of *sub-models*. Each sub-model is distributed to a different computing node, which takes care of verifying if the log is compliant with that specific sub-model. In this case, the maximum degree of partitioning

corresponds to the number of constraints that constitute the model i.e., each node is responsible for the compliance monitoring of the whole log with one specific constraint. We address this as *Fine-grain Model* partitioning[3].

As Table 1 suggests, the compliance monitoring task can be either applied considering only one dimension (*plain* partitioning strategies are reported in blue cells) or mixing the two dimensions (red cells in Table 1). The *Finest* possible partitioning consists of assigning to each machine the task of checking the compliance of a specific trace with a single constraint. The maximum possible degree of parallelism is therefore limited to the product $\#traces \times \#constraints$.

A key point for both log and model partitioning is the function that, given the partial compliance results of each single node, computes the log compliance w.r.t. the model. Let us consider first the log partitioning. As an example, such a function might be seen as a logical AND of the partial compliance results: the log is compliant to the model if all the traces are compliant to the model. Of course, depending on the specific domain, alternatives can be imagined: for example, the log might be deemed as compliant if at least 95% of the traces are compliant, etc.. Whatever function is chosen, such approach is possible if we hypothesise that the partitioning of the log into traces is completely orthogonal and independent from the compliance task. For example, suppose that the model contains constraints involving different execution traces: the log partitioning using the trace criteria cannot be applied, due to the presence of such constraints and the need for keeping the related traces together. In other words, the partitioning function must exhibit a property, that we will call *compositionality of the log partitioning function*.

In the model-oriented approach instead, the partitioning function generates a number of sub-models. Again, one might wonder how is made the function that takes partial results (concerning the compliance against each sub-model), and provides the compliance of the log with the complete model. An immediate answer would be that a logical AND is applied to the partial results: a log is compliant to a model if it is compliant to all sub-models. This

[3]We might notice however that distributing the whole log to each node might not be significant, nor efficient: intuitively, each node is interested in the portion of log that is related to the sub-model under verification. For example, it is not efficient to send the complete trace of events *"abc"* in Fig. 2 to the node that checks only the constraint "if a happens, b should follow". The part *"ab"* of the trace is sufficient.

		Log partitioning		
Model partitioning	**# partitions**	**1**	**# groups of traces**	**# traces**
	1	No partitioning	... Plain Log (PL) ...	Plain Fine-grain Log (PFL)
	# sub-models	⋮ Plain Model (PM) ⋮	⋱ Coarse-grain Log Coarse-grain Model (CLCM) ⋱	⋮ Fine-grain Log Coarse-grain Model (FLCM) ⋮
	# constraints	Plain Fine-grain Model (PFM)	... Coarse-grain Log Fine-grain Model (CLFM) ...	Finest partitioning

Table 1: Classification of partitioning methods w.r.t. the Log and Model dimensions.

is possible if the partitioning function enjoys a property of *compositionality of the model partitioning function*.

Albeit all the classes of partitioning methodologies in Table 1 are possible, real cases often require to provide information about the compliance (or non-compliance) of each trace in the log. In these cases the most interesting approaches are those that apply a fine-grained log partitioning (i.e., the log is subdivided with trace granularity as in the last column of Table 1). For this reason, in the remain of this paper we focus on *Plain Fine-grain Log (PFL)*, *Fine-grain Log Coarse-grain Model (FLCM)*, and *Finest* partitioning.

4. Plain Fine-grain Log partitioning

The PFL partitioning suggests to cut the input event log by grouping the events on the base of the trace identifier, and 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 is not partitioned, so all the constraints in the model must be distributed to all the computing nodes.

Fig. 4 specifies the architecture of our distributed monitoring framework in case of PFL partitioning.

We assume that each event in the log stream is expressed in the form:

$$tr : \mathbf{H}(a, ts) \tag{11}$$

where, tr is the unique identifier of the trace, a is the identifier of the activity and ts is the timestamp of the event. In other words, the occurrence of a record in the form (11) in the event log means that at time ts the system has observed the execution of an activity a related to trace tr.

As detailed in Fig. 4, the input stream of events in the form (11) is sent to a Log/Model partitioner component that implements PFL and therefore cuts the stream by trace identifier. It sends all the events referring to a trace to the same node in the network of computers.

The compliance of each trace to the business model is checked through the SCIFF framework running on each node. The SCIFF program takes as input the portion of event stream to check (one trace) and the behavioural model. For example, the model in Fig. 4 is represented in SCIFF as the two ICs (3) and (4). We refer these constraints with *c1* and *c2* in the following.

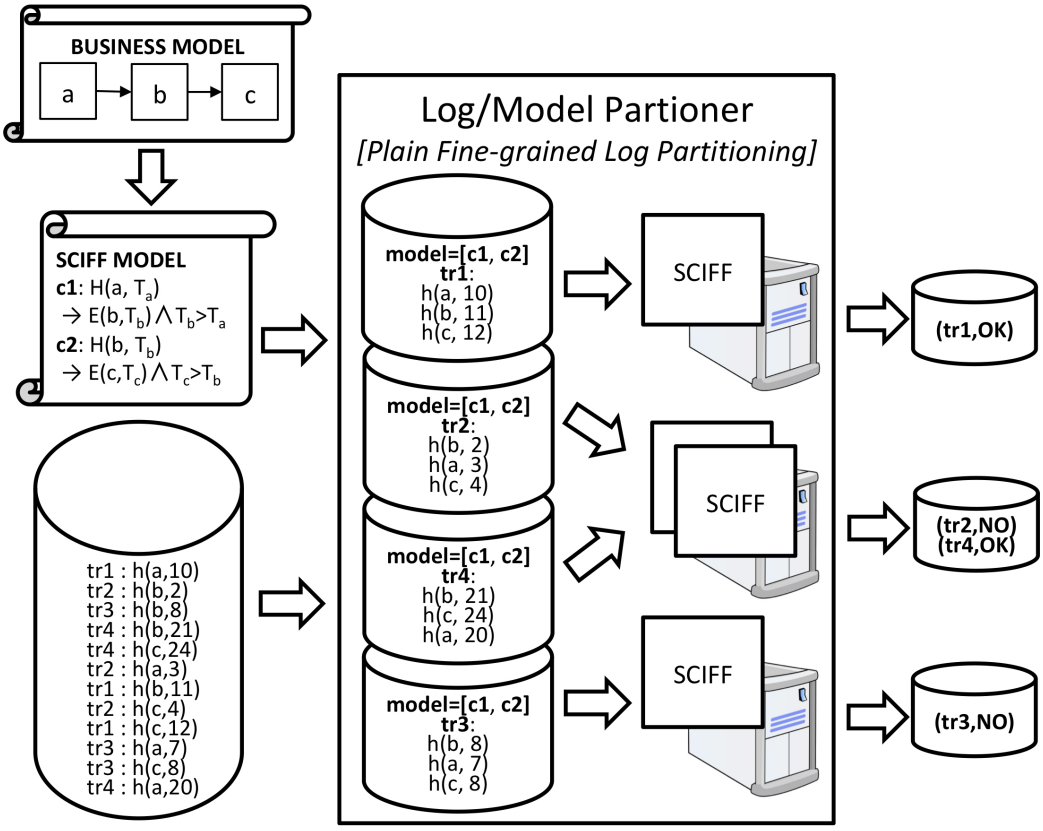


Figure 4: Example of conformance checking executed on a distributed architecture with PFL partitioning.

4.1. Partitioning methodology

In the PFL partitioning all the ICs in the model are 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.

As discussed in Section 3.3, we are interested in log partitioning functions that ensure a property of *compositionality* with respect to the notion of log compliance. In this work we make the following assumption: the model will contain only constraints about activities in the same trace. I.e., there will not be inter-trace constraints. As a consequence, each single node can assess the compliance of a trace w.r.t. a model by inspecting the model and that trace alone. Under this condition, the *compositionality* property of the PFL

partitioning method is straightforward, since it guarantees that each node will access exactly a model and a trace.

What if, for example, the model would contain inter-trace constraints (i.e., constraints that involve two or more traces)? Obviously a computing node, in order to assess the compliance, would need to access all the relevant information. In such a case, a computing node would need to access all the traces involved by the model constraints. I.e., PFL would not be applicable anymore, but we would fall into the case of PL partitioning methodology, as shown in Table 1.

4.2. MapReduce translation

The implementation of a monitoring system with PFL partitioning as presented in Fig. 4 on a MapReduce architecture is rather straightforward. Firstly, 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 $<tr, h(a, ts)>$. Later, the intermediate pairs are sent to the *reducers* that call the SCIFF program (one compliance checking procedure for each trace). The emitted results are again in the form of a $<key, value>$ pair where the *key* is the trace identifier and the *value* is the indication of compliance/non-compliance of that trace. The *map* and *reduce* procedures are further clarified though pseudo-code in Algorithm 1. Note that the reduce function is a state-full operation and the SCIFF proof procedure is not called if a previous reduce operation has already marked that trace as *non-compliant*. Indeed, since a trace is compliant iff it is compliant with all its ICs, if the SCIFF proof procedure has found a trace *non-compliant*, there is no need to further check the remaining constraints.

As the Map function of Algorithm 1 considers all the events in the input stream one by one and emits a $<key, value>$ pair for each, its time complexity as well as the cardinality of its output are $O(E)$, where E is the total number of events in the input stream. The complexity of the subsequent Reduce function is instead dependent on the SCIFF proof procedure i.e., on the complexity of the abduction reasoning process. It has been proven that the complexity of the main abductive decision problem (i.e., to determine wether an explanation exists) is located at the second level of the polynomial hierarchy (Σ_2^P-complete) [65]. The reduce phase emits a number of $<key, value>$ pairs proportional to the total T of traces in the input stream ($O(T)$).

Algorithm 1 PFL partitioning

Input: $inputStream$, a continuous stream of events in the form (11)
Output: a list $<tr, result>$pairs, where tr is the trace id and $result$ is a boolean value indicating weather tr is compliant with the model M.

1: **procedure** MAP($inputStream$)
2: **for each** $event$ **in** $inputStream$ **do**
3: $<tr, a, ts> = parseEvent(event)$
4: Emit $<k, v> = <tr, h(a, ts)>$
5: **end for**
6: **end procedure**
7:

Require: $M = [c_1...c_n]$, list of model constraints broadcasted to all nodes
8: **procedure** REDUCE($k = tr, values = [h(a_1, ts_1)...h(a_n, ts_n)], state = s$)
9: **if** $s.get == false$ **then**
10: **return** $<tr$,**false**$>$
11: **else**
12: $result = SCIFF(M, values)$
13: $s.update(result)$
14: Emit $<k, v> = <tr, result>$
15: **end if**
16: **end procedure**

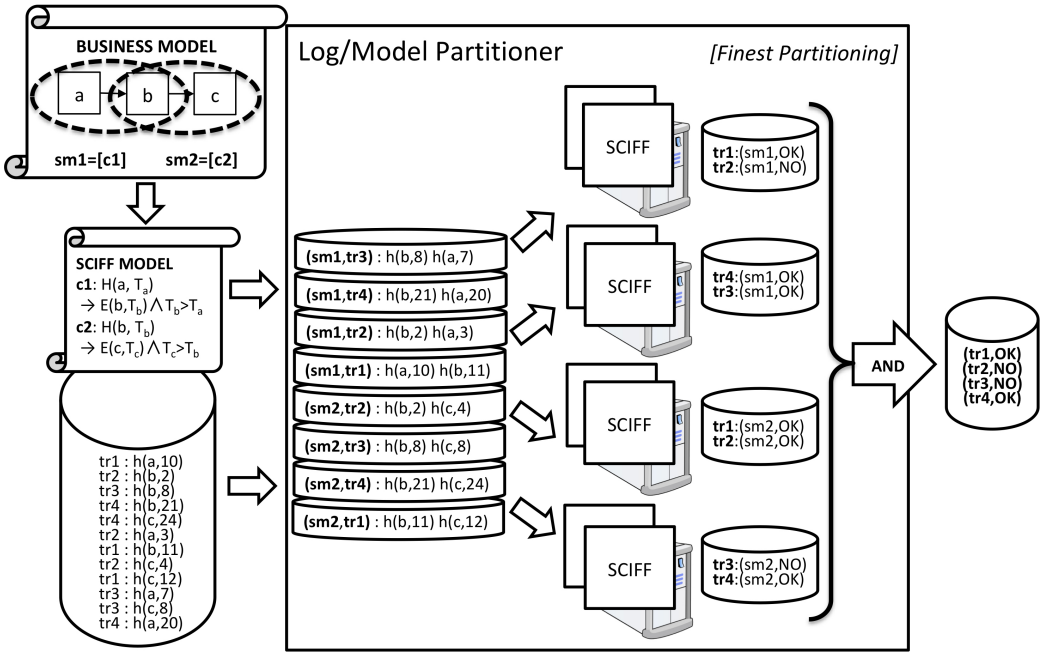


Figure 5: Example of conformance checking executed on a distributed architecture with Finest partitioning.

5. Fine-grain Log Coarse-grain Model and Finest partitioning

The model partitioning focuses on the division of the model into sub-models, that will be checked then by different machines. As real cases mostly require a per trace compliance, in the following, we treat model partitioning associated with fine-grain log partitioning (last column of Table 1). The result is FLCM (when the model is divided into sub-models) or Finest partitioning (when model partitioning reaches also single constraint granularity).

As a straightforward optimisation, we partition also each trace, by grouping events on the basis of the sub-models: each computing node receives just the fraction of each trace that contains the activities mentioned in the assigned sub-model. Fig. 5 highlights an example of Finest partitioning (because sub-models $sm1$ and $sm2$ are both composed of a single constraint), but the analogous for FLCM is straightforward.

Here, the first step of log slicing is not limited to a trace identifier-based partitioning (as it was in Fig. 4) but a model subdivision is also taken into account. Thus, each trace is sliced into sets of events according to the sub-models they belong to.

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 replicated and paired with both $sm1$ and $sm2$).

As previously discussed, we focus on log partitioning with trace granularity (last column of Table 1) and we deem a log compliant if all the contained traces are compliant. Hence, without loss of generality, we further restrict our attention on the compliance of a single trace.

Differently from Fig. 4, the results obtained after the SCIFF phase in Fig. 5 are finally redirected to a module that performs a logical AND of the results for each trace. Indeed, like in [15], a trace is compliant to a model if it is compliant to all sub-models i.e., the compliance w.r.t. the complete model is defined as the logical AND of the compliance of a trace w.r.t. each sub-model. Such behaviour can be ensured by choosing a proper model partitioning function that guarantees the *compositionality* property introduced in Section 3.3.

5.1. Partitioning methodology

As discussed in Section 3.3, we are interested in model partitioning functions that enjoy a property of *compositionality* with respect to the notion of log compliance and, more in particular, w.r.t. the notion of trace compliance. With this respect, the SCIFF framework enjoys a noteworthy result: under specific syntactic conditions, *any* partitioning of the model ensures the compositionality. To this end, we will briefly recall few Definitions and Theorems from previous works. First of all, we formally define the notion of *compositionality* for a SCIFF specification:

Definition 1 (Compositionality). *(Definition 4.4.2 in [66], page 110) A SCIFF specification $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$ is compositional (modulo compliance) if and only if for all traces $\mathcal{T}$, $\forall \mathcal{IC}_1$ and $\forall \mathcal{IC}_2$ s.t. $\mathcal{IC} = \mathcal{IC}_1 \cup \mathcal{IC}_2$:*

$$\begin{aligned} &\mathcal{T} \text{ is compliant with } \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC}_1 \rangle \\ \wedge\ &\mathcal{T} \text{ is compliant with } \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC}_2 \rangle \iff \mathcal{T} \text{ is compliant with } \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle \end{aligned}$$

Notice that this definition of compositionality is quite strong, since it requires that compliance holds for any possible partition of the ICs set $\mathcal{IC}$.

Then, we define a syntactical property of the ICs:

Definition 2 (Unchained Specification). *(Definition 4.4.3 in [66], page 111) A SCIFF specification $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$ is unchained iff no IC in $\mathcal{IC}$ contains an abducible in its body.*

Finally, we report here the Theorem that ensure the compositionality of the SCIFF specification[4]:

Theorem 1. *[Compositionality of Unchained Specifications] (Theorem 4.4.1 in [66], page 111) If a SCIFF specification is unchained, then it is also compositional.*

Theorem 1 ensures that upon given syntactical conditions on the SCIFF specification, any model partitioning ensures the compositionality property. This is an important feature of the SCIFF framework, since no matter how the model will be partitioned, it will be possible to distribute it between different computing nodes. The partitioning functions envisaged by the FLCM and Finest partitioning methods are simply two possible ways of partitioning a model, and thus they are compositional w.r.t. the notion of log/trace compliance.

The reader might wonder the impact of the syntactic conditions to be met to ensure unchained specifications. In previous works we discussed how different process modelling languages can be expressed formally in SCIFF: for example, DecSerFlow was addressed in [67]; in [66] a further declarative language for business process, CLIMB, was introduced; in [61, 14] we focused on procedural languages for workflows. These languages have been all equipped with SCIFF unchained specifications, thus ensuring the compositionality.

5.2. MapReduce translation

The implementation on a MapReduce architecture of a monitoring system with mixed log/model partitioning as presented in Fig. 5 is not as simple as the plain log partitioning is.

The following steps can be identified:

1. as for the plain log partitioning, the input event stream must be processed by a *map* function that extracts the important information and emits a collection of $<key, value>$ pairs in the form $<tr, h(a, ts)>$;

[4]The interested reader can find the proof of the theorem in [66].

Algorithm 2 FLCM partitioning

Input: *inputStream*, a continuous stream of events in the form (11)
Output: a list $<tr, finalResult>$ pairs expressing the compliance of the trace tr to the model as a whole.

procedure MAP(*inputStream*)
  for each *event* **in** *inputStream* **do**
    $<tr, a, ts> = parseEvent(event)$
    Emit $<k, v> = <tr, h(a, ts)>$
  end for
end procedure

Require: SM=$[sm_1...sm_n]$, list of submodels, each is a list of constraints
procedure FLATMAP($k = tr, v = h(a, ts)$)
  $submodelList = getSubmodels(a, SM)$
  for each *sm* **in** *submodelList* **do**
    Emit $<k, v> = <(sm, tr), h(a, ts)>$
  end for
end procedure

procedure SCIFFREDUCE($k = (sm, tr)$,
  $values = [h(a_1, ts_1), ..., h(a_n, ts_n)], state = s$)
  if $s.get == false$ **then**
    return $<tr$,**false**$>$
  else
    $result = SCIFF(sm, values)$
    $s.update(result)$
    Emit $<k, v> = <tr, (sm, result)>$
  end if
end procedure

procedure ANDREDUCE($k = tr$,
  $values = [(sm_1, result_1), ..., (sm_n, result_n)]$)
  for each $(sm, result)$ **in** *values* **do**
    $finalResult = finalResult$ && $result$
  end for
  Emit $<k, v> = <tr, finalResult>$
end procedure

2. the pairs are processed by a *flatMap* function that adds to each $<tr, h(a, ts)>$ the list of sub-models sm in which the event a is present. The output of the *flatMap* function is a list of $<(sm, tr), h(a, ts)>$ pairs. This function is also responsible to replicate the events that occur in different sub-models. For example, looking at Fig. 5, the record $tr2 : h(b, 2)$ generates two records: $<(sm1, tr2), h(b, 2)>$ and $<(sm2, tr2), h(b, 2)>$ because the activity b is present in both the sub-models;

3. the *reducers* receive all the pairs with the same (sm, tr) key and pass all the $h(a, ts)$ values to a SCIFF proof procedure. The output of this reduce phase is therefore a pair that expresses the compliance of a specific trace with a specific sub-model: $<tr, (sm, result)>$. Analogously to PFL, this operation is performed in a state-full fashion, so that non-compliant cases are suddenly identified and no further useless computation is performed;

4. the compliance results are grouped by tr key and the final *reducers* perform an AND function of the *results* associated to each sm. The output is therefore a $<key, value>$ pair where the key is the trace identifier and the $value$ is the indication of compliance/non-compliance of that trace with the model as a whole.

These four steps are further clarified though pseudo-code in Algorithm 2.

As regards the complexity of Algorithm 2, analogously to PFL in Algorithm 1, the *map* phase has time complexity of order $O(E)$ and emits $O(E)$ $<key, value>$ pairs. Each subsequent *flatMap* considers an event and iterates over the sub-models where the activity of that event is present. In the worst case, when an activity is present in all the sub-models, the *flatMap* complexity is $O(|SM|)$ – where SM is the set of the sub-models, each one is a list of constraints – and the number of emitted $<key, value>$ pairs has order of $O(E|SM|)$.

Similarly to the Reduce procedure of Algorithm 1, the time complexity of the *SCIFFReduce* procedure in Algorithm 2 is dependent on the abductive task (Σ_2^P-complete) [65]. *SCIFFReduce* emits $O(T|SM|)$ $<key, value>$ pairs. Finally the *AndReduce* iterates over the values in its input causing a time complexity of order $O(|SM|)$. The number of emitted pairs is $O(T)$.

6. Experimental Evaluation

Recalling the two crucial features that a compliance monitoring system for business process should expose, we can notice that, while the high expressiveness of SCIFF notation has been already studied in previous works [13, 14], the scalability of the proposed distributed approach still need to be investigated.

6.1. Simulation setup

In order to evaluate our distributed compliance monitoring approach we need to resort to a MapReduce platform. The existing variety of big data analytics platforms can be classified according to the characteristics of the treated input: some focus on the analysis of large volumes of data in an offline fashion [51], while some others deal with stream processing [22, 68, 69]. Moreover, some hybrid platforms are able to deal with both runtime and offline analytics [24, 23]. Apache Spark [24] in particular, has gained increasing attention thanks to the generality of the proposed approach (including but not limited to the MapReduce paradigm) and the good performance shown on both batch and stream processing benchmarks [70]. Although our work mainly focuses on the runtime monitoring of streams of event log, conformance checking techniques in BPM may also involve offline analysis. Therefore, aiming to provide a general framework, we adopt the Apache Spark platform.

We evaluate our architecture for distributed compliance monitoring on a cluster of 80 physical machines (1 Spark master and 79 worker nodes). Each one is equipped with 4 CPU, 8GB RAM and 400GB disk. The computing nodes are interconnected with a 100Mbit network.

6.2. Comparison approaches

As we focus on runtime compliance monitoring of event streams, we assume to have an input flow of events with a certain rate. In this scenario it is important that the system is able to perform the monitoring computation while keeping up with the input flow. Indeed, if the distributed compliance monitor is slow w.r.t. the input stream, an increasing delay is introduced in the computation making the overall processing system unstable. Furthermore, as the Spark platform stores the data waiting to be processed in a buffer area (on memory or disk), a processing task always slower than the input rate may also produce a saturation of the buffer.

In order to evaluate the scalability of our approach we test the performance with an increasing number of physical nodes and we determine for each case which is the maximum input rate that the monitoring system can manage without exhibiting an increasing delay in computation.

The evaluation is conducted on three business process scenarios:

- *Scenario A. Real-life model and artificial log.* The input log for testing is composed of a collection of synthetic traces previously generated by the SCIFF itself, exploiting its abduction feature [71]. As a first step, we generated 600 traces representing the treatment of a femoral fracture compliant with the real-life business model proposed in Fig. 3 of the work [61]. The generated traces are used to foster an event stream with configurable rate.

- *Scenario B. Artificial model and log.* We artificially create a model composed of 810 subsequent activities, each one with a duration between 5 and 10 time units. Translating this model into SCIFF's ICs, we obtain the following constraints for each couple (A_i, A_{i+1}) of subsequent activities in the model:

$$\begin{aligned} \mathbf{H}(a_i\_start, T_{a_i\_start}) \rightarrow & \mathbf{E}(a_i\_end, T_{a_i\_end}) \wedge \\ & \wedge T_{a_i\_end} \geq T_{a_i\_start} + 5 \wedge \\ & \wedge T_{a_i\_end} \leq T_{a_i\_start} + 10. \end{aligned} \tag{12}$$

$$\begin{aligned} \mathbf{H}(a_i\_end, T_{a_i\_end}) \rightarrow & \mathbf{E}(a_{i+1}\_start, T_{a_{i+1}\_start}) \wedge \\ & \wedge T_{a_{i+1}\_start} > T_{a_i\_end}. \end{aligned} \tag{13}$$

  As a consequence, the artificial SCIFF model is composed of 1619 ICs. We use the SCIFF abduction feature to generate 5 very long traces, all compliant with such model.

- *Scenario C. Real-life model and log.* We consider the real-life input log provided in eXtensible Event Stream (XES) format [72] for the International Business Process Intelligence Challenge of 2012, an initiative featured every year since 2011 by the BPI workshop [73]. The event log [74] is taken from a Dutch Financial Institute and records a series of application processes for loans and overdrafts. It contains 262,200 events referring to 36 different activities and belonging to 13,087 traces. The model is composed of 30 ICs written in SCIFF formalism on the

basis of the 15 requirements expressed in natural language in the work by Ly et al. (requirements from R17 to R31 in [2]).

6.3. Results

As regards *Scenario A*, Fig. 6 shows the performance of the distributed infrastructure when checking the compliance of fracture treatment traces employing PFL, FLCM and Finest partitioning. As prescribed by the model in [61], the system must check 28 ICs for each of the 600 traces. As they are all artificially generated from the same model, they are all compliant. Therefore, the system cannot take advantage of the state-full reducers in Algorithms 1 and 2 but must check each trace till the end before being able to say that it is compliant. So, this always-compliant input stream allow us to evaluate the performance of our system in the worst-case scenario from the computational point of view.

FLCM is implemented by subdividing the model into 4 sub-models, while Finest partitioning prescribes 28 sub-models (each one responsible for a single constraint). Both PFL and FLCM methodologies exhibit good scalability feature, because the maximum input stream rate managed by the architecture is approximatively proportional to the number of infrastructure nodes. Finest partitioning as well shows a proportionality of the maximum input rate w.r.t. the number of nodes employed in computation, but the performance is significantly lower when compared to PFL and FLCM.

Considering PFL, the compliance checking system on a single node is able to keep up with an input stream rate of 500 Byte/s, approximately correspondent to 1,4 fracture treatment traces arriving each second. When executed on 79 workers the maximum stream rate is around 30000 Byte/s, which is approximately 83 traces per second.

In this scenario, there is an evident difference in performance between the partitioning methodologies: PFL always outperforms FLCM and Finest. This can be explained considering that the traces in this example have a limited length, such that a single node can process more than one complete trace per second. In this case, there is no need to subdivide the compliance task into sub-models. On the contrary, the model partitioning introduces an unnecessary overhead of communication because the input events on the border of the sub-models need to be replicated. The model subdivision in fact, shows its advantages only on particular domains. For example, if the number or traces to be considered is limited but there is a high number of

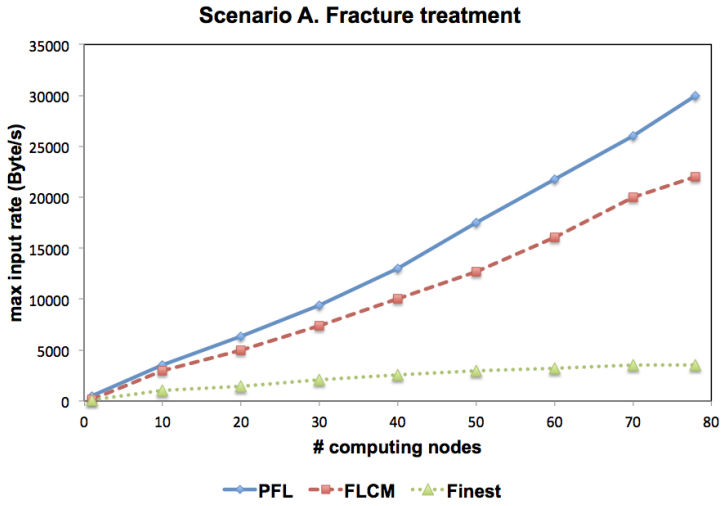


Figure 6: Performance of the PFL and FLCM and Finest partitioning on artificial traces representing the workflow of fracture treatment as presented in Fig. 3 of [61]

constraints that have to be checked for each trace. In that case, adding model partitioning to the PFL should improve the performance.

To verify this hypothesis, we repeat the experiment on *Scenario B*, where the log is composed by 5 very long artificially generated traces, each one to be checked for compliance against a model composed by 1619 ICs. The performance of the system is reported in Fig. 7. In this case, PFL will launch only 5 SCIFF proof procedures, each receiving all the events of the assigned trace. On the same input, we repeat the experiments with both FLCM and Finest partitioning. FLCM is implemented dividing the model into 15 sub-models, while Finest prescribes to launch a SCIFF proof procedure for each couple *(trace,constraint)* i.e., each SCIFF will check one single constraint of the model on one single trace.

As expected, the scalability of the system is limited with PFL partitioning (the maximum input rate manageable by the system cannot overcome 4000 Byte/s, even when employing all the 79 workers), while FLCM increases its performance proportionally with the number of employed nodes. Finest partitioning instead shows the worst performance because the compliance monitoring task results divided into too many pieces, generating an overhead of communication due to the replicated events.

Finally, we focus on *Scenario C*. The real-life log is analysed employing all the nine partitioning methodologies highlighted in Table 1. Fig. 8 shows

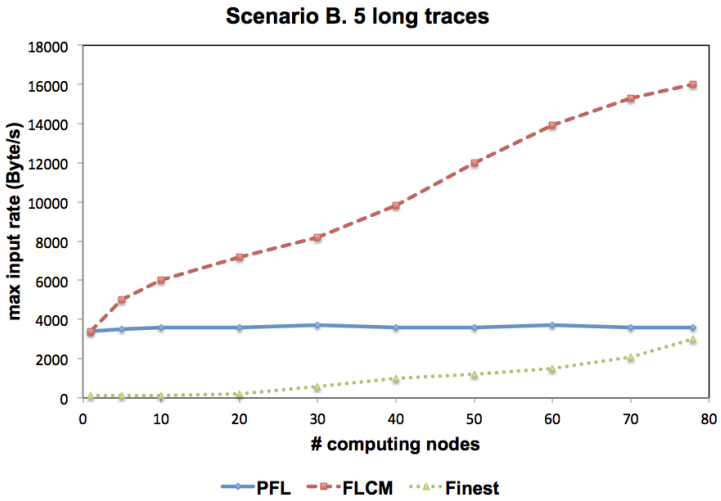


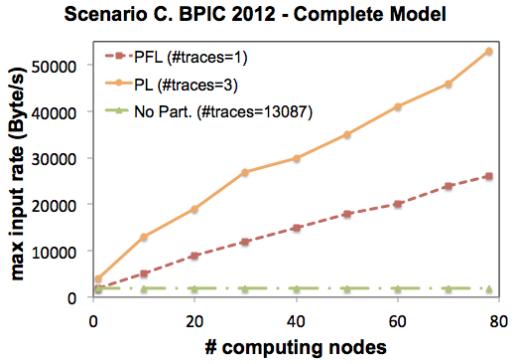
Figure 7: Performance of the PFL, FLCM and Finest partitioning on a flow of 5 artificial traces representing a synthetic model composed of 810 subsequent activities

three different graphs reporting the performance of the system when: • no model partitioning is applied (Fig. 8a), • the model is cut into six sub-models (Fig. 8b), each one responsible for five ICs, • the model is sliced into single ICs generating 30 sub-models (Fig. 8c).

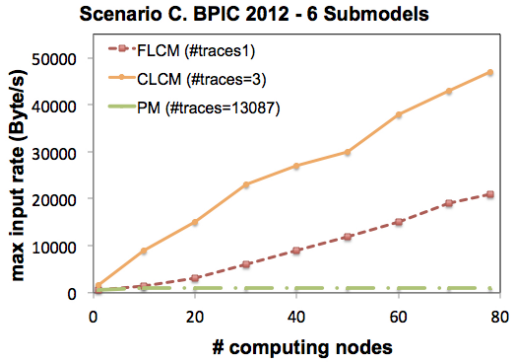
In all these three cases, we compare the performance of the system when: • the log is sliced by trace identifier (orange solid lines in Fig. 8), • the log is partitioned into groups of three traces each (red dashed lines), • no log partitioning is applied (green dot-dashed lines).

In all the graphs, the best performance is reached when the log is sliced into groups of traces. If the compliance checking is conducted by trace identifier, we observe a decrease in the maximum input rate that the monitor can keep up with, while the worst case occurs when no log partitioning is applied. So, contrary to what is expected, in this example, the best performance are not reached when the log is partitioned at its finest grane (i.e., by trace id), but when it is sliced into groups of traces.

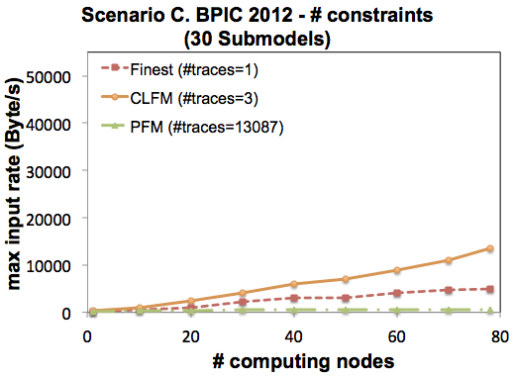
This behaviour could be explained taking into account that the system must face a constant time overhead when launching a SCIFF proof procedure. In the case of the red dashed lines of Fig. 8, each trace is examined by a dedicated SCIFF, while for orange solid lines, the number of launched proof procedures decreases (as well as the time overhead they cause) because we call a SCIFF every three traces. Obviously, this kind of performance



(a)



(b)



(c)

Figure 8: Performance comparison of all the partitioning methodologies highlighted in Table 1 applied to the analysis of a real-life event log stream as described in Scenario C.

enhancement is not always possible. For example, if the business process monitoring task requires to provide an information about the compliance of each trace, partitioning the log into groups of traces – as in PL, CLCM and CLFM of Fig. 8 – is not a good solution because the compliance results would not have the desired single-trace grane.

Nevertheless, Fig. 8 shows that an extreme increase in the number of traces processed by each SCIFF (as for series No Part., PM and PFM, where each SCIFF receives all the traces in the input log) does not correspond to a further increase in performance. Indeed, if no log partitioning is applied, the maximum sustainable input rate does not augment with the number of nodes employed in computation, but the scalability of the system results limited to the number of model partitions. This behaviour could be caused by the excessive number of compliance monitoring operations requested to a limited number of SCIFF proof procedures.

In general, this experimental evaluation highlights that the performance of a compliance monitoring system can be significantly enhanced through the introduction of a mechanism to conveniently partition and distribute the monitoring task over a network of computing nodes. For example, in case of CLCM in Fig. 8b, the maximum input rate with one node is 1600 Byte/s, while on 79 nodes the system can keep up with a 47000 Byte/s input stream. Although deciding which is the optimal number of log/model partitions that brings the best performance might not be straightforward (it requires a dedicated analysis, which is left for future work), it is interesting to notice that – excluding the cases where partitioning results limited (e.g., PM) or absent (e.g., No Partitioning) – the performance of the proposed solution increases with the number of employed computing nodes, showing a remarkable scalability.

7. Conclusion

In this contribution, we present an approach 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 the SCIFF framework over a MapReduce environment.

Firstly, we propose a methodology for the repartition of the compliance checking task that is based on two partitioning dimensions: the event log and the business model. We highlight different possibilities to combine these two dimensions and we propose a classification.

Secondly, we focus on the most common case in real business process, that require to provide an information about the compliance of each trace and we describe our MapReduce implementations together with a formal definition of the properties that the partitioning function should exhibit.

The tests conducted on a Spark datacenter of 80 computing nodes show promising results in terms of scalability for the proposed approach, albeit in case of model partitioning, the communication overhead inevitably introduced by data subdivision negatively influences the results.

In general, further investigation is needed to highlight the connections between the log/model partitioning methodologies and the efficiency of the distributed system, i.e., to determine the best number of sub-models and constraints in each; and – if trace grouping is possible – decide the best cardinality of each set of traces. For this reason, we are currently focusing on the definition of a reliable log/model partitioning method, that takes into account several factors (including the number of possible paths in the model, computational cost of each constraint, and specificities of the workflow, etc.) and determines the optimal number of log/model partitions that should be employed on a certain distributed infrastructure to obtain the best performance.

Another interesting extension of the work is the implementation of the proposed approach over a platform for big data analytics alternative to Spark. In recent time indeed, among all the existing distributed platforms, Apache Flink [23] has gained increasing interest in the research community, thanks to promising preliminary performance tests and a more coherent management of data in case of stream processing. The implementation of Log/Model Partitioner over a Flink architecture is therefore matter of forthcoming work.

8. Acknowledgements

We thank Alessio Ferretti and Luca Ghedini (Computer Center of School of Engineering and Architecture, University of Bologna) for the precious assistance in the setup of the evaluation environment.

9. References

[1] W. Van Der Aalst, A. Adriansyah, A. K. A. de Medeiros, F. Arcieri, Process mining manifesto, in: Business Process Management Workshops, Springer Berlin Heidelberg, 2012.

[2] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, W. M. P. van der Aalst, Compliance monitoring in business processes: Functionalities, application, and tool-support, Inf. Syst. 54 (2015) 209–234. doi:10.1016/j.is.2015.02.007.
URL http://dx.doi.org/10.1016/j.is.2015.02.007

[3] A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, N. Russell (Eds.), Modern Business Process Automation - YAWL and its Support Environment, Springer, 2010.
URL http://www.yawlbook.com/home/

[4] W. M. P. van der Aalst, Verification of workflow nets, in: P. Azéma, G. Balbo (Eds.), Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings, Vol. 1248 of Lecture Notes in Computer Science, Springer, 1997, pp. 407–426. doi:10.1007/3-540-63139-9_48.
URL http://dx.doi.org/10.1007/3-540-63139-9_48

[5] W. M. P. van der Aalst, The application of petri nets to workflow management, Journal of Circuits, Systems, and Computers 8 (1) (1998) 21–66. doi:10.1142/S0218126698000043.
URL http://dx.doi.org/10.1142/S0218126698000043

[6] BPEL: Business Process Execution Language, https://www.oasis-open.org/committees/tc_home.php\?wg_abbrev=wsbpel (accessed 06.08.2017).

[7] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, Declarative workflow, in: ter Hofstede et al. [3], pp. 175–201. doi:10.1007/978-3-642-03121-2_6.
URL http://dx.doi.org/10.1007/978-3-642-03121-2_6

[8] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Computer Science - R&D 23 (2) (2009) 99–113. doi:10.1007/s00450-009-0057-9.
URL http://dx.doi.org/10.1007/s00450-009-0057-9

[9] BPMN: Business Process Model and Notation, http://www.bpmn.org/ (accessed 06.08.2017).

[10] S. Haller, S. Karnouskos, C. Schroth, The internet of things in an enterprise context, in: J. Domingue, D. Fensel, P. Traverso (Eds.), Future

Internet - FIS 2008, First Future Internet Symposium, FIS 2008, Vienna, Austria, September 29-30, 2008, Revised Selected Papers, Vol. 5468 of Lecture Notes in Computer Science, Springer, 2008, pp. 14–28. doi:10.1007/978-3-642-00985-3_2.
URL http://dx.doi.org/10.1007/978-3-642-00985-3_2

[11] S. Meyer, A. Ruppen, C. Magerkurth, Internet of things-aware process modeling: Integrating iot devices as business process resources, in: C. Salinesi, M. C. Norrie, O. Pastor (Eds.), Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings, Vol. 7908 of Lecture Notes in Computer Science, Springer, 2013, pp. 84–98. doi:10.1007/978-3-642-38709-8_6.
URL http://dx.doi.org/10.1007/978-3-642-38709-8_6

[12] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, Verifiable agent interaction in abductive logic programming: The SCIFF framework, ACM Trans. Comput. Log. 9 (4). doi:10.1145/1380572.1380578.
URL http://doi.acm.org/10.1145/1380572.1380578

[13] F. Chesani, P. Mello, M. Montali, S. Storari, 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) (2010) 165–190. doi:10.3233/MGS-2010-0147.
URL http://dx.doi.org/10.3233/MGS-2010-0147

[14] F. Chesani, R. De Masellis, C. D. Francescomarino, C. Ghidini, P. Mello, M. Montali, S. Tessaris, Abducing compliance of incomplete event logs, in: AI*IA 2016: Advances in Artificial Intelligence, 2016, pp. 208–222. doi:10.1007/978-3-319-49130-1_16.
URL http://dx.doi.org/10.1007/978-3-319-49130-1_16

[15] W. M. P. van der Aalst, Distributed process discovery and conformance checking, in: de Lara and Zisman [75], pp. 1–25. doi:10.1007/978-3-642-28872-2_1.
URL http://dx.doi.org/10.1007/978-3-642-28872-2_1

[16] W. M. P. van der Aalst, E. Damiani, Processes meet big data: Connecting data science with process science, IEEE Trans. Services Computing

8 (6) (2015) 810–819. doi:10.1109/TSC.2015.2493732.
URL http://dx.doi.org/10.1109/TSC.2015.2493732

[17] H. Reguieg, F. Toumani, H. R. M. Nezhad, B. Benatallah, Using mapreduce to scale events correlation discovery for business processes mining, in: A. P. Barros, A. Gal, E. Kindler (Eds.), Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings, Vol. 7481 of Lecture Notes in Computer Science, Springer, 2012, pp. 279–284. doi:10.1007/978-3-642-32885-5_22.
URL http://dx.doi.org/10.1007/978-3-642-32885-5_22

[18] A. Azzini, E. Damiani, Process mining in big data scenario, in: P. Ceravolo, S. Rinderle-Ma (Eds.), Proceedings of the 5th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2015), Vienna, Austria, December 9-11, 2015., Vol. 1527 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 149–153.
URL http://ceur-ws.org/Vol-1527/paper14.pdf

[19] D. A. Basin, G. Caronni, S. Ereth, M. Harvan, F. Klaedtke, H. Mantel, Scalable offline monitoring of temporal specifications, Formal Methods in System Design 49 (1-2) (2016) 75–108. doi:10.1007/s10703-016-0242-y.
URL http://dx.doi.org/10.1007/s10703-016-0242-y

[20] B. Barre, M. Klein, M. Soucy-Boivin, P. Ollivier, S. Hallé, Mapreduce for parallel trace validation of LTL properties, in: S. Qadeer, S. Tasiran (Eds.), Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers, Vol. 7687 of Lecture Notes in Computer Science, Springer, 2012, pp. 184–198. doi:10.1007/978-3-642-35632-2_20.
URL http://dx.doi.org/10.1007/978-3-642-35632-2_20

[21] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113. doi:10.1145/1327452.1327492.
URL http://doi.acm.org/10.1145/1327452.1327492

[22] Apache Storm, http://storm.apache.org (accessed 06.08.2017).

[23] Apache Flink, https://flink.apache.org (accessed 06.08.2017).

[24] Apache Spark, http://spark.apache.org (accessed 06.08.2017).

[25] D. Loreti, F. Chesani, A. Ciampolini, P. Mello, Distributed compliance monitoring of business processes over mapreduce architectures, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, ICPE '17 Companion, ACM, New York, NY, USA, 2017, pp. 79–84. doi:10.1145/3053600.3053616.
URL http://doi.acm.org/10.1145/3053600.3053616

[26] W. M. P. van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, Springer, 2011. doi:10.1007/978-3-642-19345-3.
URL http://dx.doi.org/10.1007/978-3-642-19345-3

[27] M. Leucker, C. Schallhart, A brief account of runtime verification, J. Log. Algebr. Program. 78 (5) (2009) 293–303. doi:10.1016/j.jlap.2008.08.004.
URL http://dx.doi.org/10.1016/j.jlap.2008.08.004

[28] F. M. Maggi, M. Westergaard, M. Montali, W. M. P. van der Aalst, Runtime verification of ltl-based declarative process models, in: Khurshid and Sen [76], pp. 131–146. doi:10.1007/978-3-642-29860-8_11.
URL http://dx.doi.org/10.1007/978-3-642-29860-8_11

[29] D. A. Basin, F. Klaedtke, S. Müller, B. Pfitzmann, Runtime monitoring of metric first-order temporal properties, in: R. Hariharan, M. Mukund, V. Vinay (Eds.), IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India, Vol. 2 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008, pp. 49–60. doi:10.4230/LIPIcs.FSTTCS.2008.1740.
URL http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2008.1740

[30] M. Montali, F. M. Maggi, F. Chesani, P. Mello, W. M. P. van der Aalst, Monitoring business constraints with the event calculus, ACM TIST 5 (1) (2013) 17. doi:10.1145/2542182.2542199.
URL http://doi.acm.org/10.1145/2542182.2542199

[31] K. Namiri, N. Stojanovic, Pattern-based design and validation of business process compliance, in: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I, Vol. 4803 of Lecture Notes in Computer Science, Springer, 2007, pp. 59–76. doi:10.1007/978-3-540-76848-7_6.
URL http://dx.doi.org/10.1007/978-3-540-76848-7_6

[32] E. A. P. Santos, R. Francisco, A. D. Vieira, E. D. F. R. Loures, M. A. B. de Paula, Modeling business rules for supervisory control of process-aware information systems, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part II, Vol. 100 of Lecture Notes in Business Information Processing, Springer, 2011, pp. 447–458. doi:10.1007/978-3-642-28115-0_42.
URL http://dx.doi.org/10.1007/978-3-642-28115-0_42

[33] F. M. Maggi, M. Montali, W. M. P. van der Aalst, An operational decision support framework for monitoring business constraints, in: de Lara and Zisman [75], pp. 146–162. doi:10.1007/978-3-642-28872-2_11.
URL http://dx.doi.org/10.1007/978-3-642-28872-2_11

[34] M. Montali, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, Evaluating compliance: from LTL to abductive logic programming, in: D. Ancona, M. Maratea, V. Mascardi (Eds.), Proceedings of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015., Vol. 1459 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 101–116.
URL http://ceur-ws.org/Vol-1459/paper8.pdf

[35] D. A. Basin, M. Harvan, F. Klaedtke, E. Zalinescu, MONPOLY: monitoring usage-control policies, in: Khurshid and Sen [76], pp. 360–364. doi:10.1007/978-3-642-29860-8_27.
URL http://dx.doi.org/10.1007/978-3-642-29860-8_27

[36] A. Artikis, A. Skarlatidis, F. Portet, G. Paliouras, Logic-based event recognition, Knowledge Eng. Review 27 (4) (2012) 469–506. doi:10.1017/S0269888912000264.
URL http://dx.doi.org/10.1017/S0269888912000264

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

[38] S. Dreyer, D. Olivotti, B. Lebek, M. H. Breitner, Towards a smart services enabling information architecture for installed base management in manufacturing, in: J. M. Leimeister, W. Brenner (Eds.), Towards Thought Leadership in Digital Transformation: 13. Internationale Tagung Wirtschaftsinformatik, WI 2017, St.Gallen, Switzerland, February 12-15, 2017., 2017.
URL http://aisel.aisnet.org/wi2017/track01/paper/3

[39] M. H. Ali, C. Gerea, B. S. Raman, B. Sezgin, T. Tarnavski, T. Verona, P. Wang, P. Zabback, A. Kirilov, A. Ananthanarayan, M. Lu, A. Raizman, R. Krishnan, R. Schindlauer, T. Grabs, S. Bjeletich, B. Chandramouli, J. Goldstein, S. Bhat, Y. Li, V. D. Nicola, X. Wang, D. Maier, I. Santos, O. Nano, S. Grell, Microsoft CEP server and online behavioral targeting, PVLDB 2 (2) (2009) 1558–1561.
URL http://www.vldb.org/pvldb/2/vldb09-1019.pdf

[40] E. Mulo, U. Zdun, S. Dustdar, Domain-specific language for event-based compliance monitoring in process-driven soas, Service Oriented Computing and Applications 7 (1) (2013) 59–73. doi:10.1007/s11761-012-0121-3.
URL http://dx.doi.org/10.1007/s11761-012-0121-3

[41] EsperThech Esper: Event Processing for Java, http://www.espertech.com/products/esper.php (accessed 06.08.2017).

[42] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, C. Soriente, P. Valduriez, Streamcloud: An elastic and scalable data streaming system, IEEE Trans. Parallel Distrib. Syst. 23 (12) (2012) 2351–2365. doi:10.1109/TPDS.2012.24.
URL http://dx.doi.org/10.1109/TPDS.2012.24

[43] R. Barazzutti, P. Felber, C. Fetzer, E. Onica, J. Pineau, M. Pasin, E. Rivière, S. Weigert, Streamhub: a massively parallel architecture for high-performance content-based publish/subscribe, in: S. Chakravarthy, S. D. Urban, P. R. Pietzuch, E. A. Rundensteiner (Eds.), The 7th ACM International Conference on Distributed Event-Based Systems, DEBS '13, Arlington, TX, USA - June 29 - July 03, 2013, ACM, 2013, pp.

63–74. doi:10.1145/2488222.2488260.
URL http://doi.acm.org/10.1145/2488222.2488260

[44] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, Z. Zhang, Timestream: reliable stream computation in the cloud, in: Z. Hanzálek, H. Härtig, M. Castro, M. F. Kaashoek (Eds.), Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013, ACM, 2013, pp. 1–14. doi:10.1145/2465351.2465353.
URL http://doi.acm.org/10.1145/2465351.2465353

[45] W. A. Higashino, CEPSim: A Simulator for Cloud-Based Complex Event Processing.; BigData Congress, 2015. doi:10.1109/BigDataCongress.2015.34.

[46] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1128–1142. doi:10.1109/TKDE.2004.47.
URL http://dx.doi.org/10.1109/TKDE.2004.47

[47] J. Evermann, G. Assadipour, Big data meets process mining: implementing the alpha algorithm with map-reduce, in: Y. Cho, S. Y. Shin, S. Kim, C. Hung, J. Hong (Eds.), Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014, ACM, 2014, pp. 1414–1416. doi:10.1145/2554850.2555076.
URL http://doi.acm.org/10.1145/2554850.2555076

[48] J. Evermann, Scalable process discovery using map-reduce, IEEE Trans. Services Computing 9 (3) (2016) 469–481. doi:10.1109/TSC.2014.2367525.
URL http://dx.doi.org/10.1109/TSC.2014.2367525

[49] S. Hernández, S. J. van Zelst, J. Ezpeleta, W. M. P. van der Aalst, Handling big(ger) logs: Connecting prom 6 to apache hadoop, in: F. Daniel, S. Zugal (Eds.), Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015., Vol. 1418 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 80–84.
URL http://ceur-ws.org/Vol-1418/paper17.pdf

[50] H. Ossher, R. K. E. Bellamy, D. Amid, A. Anaby-Tavor, M. Callery, M. Desmond, J. de Vries, A. Fisher, T. Frauenhofer, S. Krasikov, I. Simmonds, C. Swart, Business insight toolkit: Flexible prerequirements modeling, in: 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Companion Volume, IEEE, 2009, pp. 423–424. doi:10.1109/ICSE-COMPANION.2009.5071043.
URL http://dx.doi.org/10.1109/ICSE-COMPANION.2009.5071043

[51] Apache Hadoop, https://hadoop.apache.org/ (accessed 06.08.2017).

[52] D. Bianculli, C. Ghezzi, S. Krstic, Trace checking of metric temporal logic with aggregating modalities using mapreduce, in: D. Giannakopoulou, G. Salaün (Eds.), Software Engineering and Formal Methods - 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings, Vol. 8702 of Lecture Notes in Computer Science, Springer, 2014, pp. 144–158. doi:10.1007/978-3-319-10431-7_11.
URL http://dx.doi.org/10.1007/978-3-319-10431-7_11

[53] F. Chesani, A. Ciampolini, D. Loreti, P. Mello, Process mining monitoring for map reduce applications in the cloud, in: J. S. Cardoso, D. Ferguson, V. M. Muñoz, M. Helfert (Eds.), CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science, Volume 1, Rome, Italy, April 23-25, 2016., SciTePress, 2016, pp. 95–105. doi:10.5220/0005864000950105.
URL http://dx.doi.org/10.5220/0005864000950105

[54] F. Chesani, A. Ciampolini, D. Loreti, P. Mello, Map Reduce Autoscaling over the Cloud with Process Mining Monitoring, Springer International Publishing, Cham, 2017, pp. 109–130. doi:10.1007/978-3-319-62594-2_6.
URL https://doi.org/10.1007/978-3-319-62594-2_6

[55] M. C. S. Filho, C. C. Monteiro, P. R. M. Inácio, M. M. Freire, Approaches for optimizing virtual machine placement and migration in cloud environments: A survey, J. Parallel Distrib. Comput. 111 (2018) 222–250. doi:10.1016/j.jpdc.2017.08.010.
URL https://doi.org/10.1016/j.jpdc.2017.08.010

[56] D. Loreti, A. Ciampolini, A distributed self-balancing policy for virtual machine management in cloud datacenters, in: International Conference on High Performance Computing & Simulation, HPCS 2014, Bologna, Italy, 21-25 July, 2014, IEEE, 2014, pp. 391–398. doi:10.1109/HPCSim.2014.6903712.
URL https://doi.org/10.1109/HPCSim.2014.6903712

[57] D. Loreti, A. Ciampolini, Mapreduce over the hybrid cloud: A novel infrastructure management policy, in: I. Raicu, O. F. Rana, R. Buyya (Eds.), 8th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2015, Limassol, Cyprus, December 7-10, 2015, IEEE Computer Society, 2015, pp. 174–178. doi:10.1109/UCC.2015.33.
URL http://doi.ieeecomputersociety.org/10.1109/UCC.2015.33

[58] T. Baker, O. F. Rana, R. Calinescu, R. Tolosana-Calasanz, J. Á. Bañares, Towards Autonomic Cloud Services Engineering via Intention Workflow Model, Springer International Publishing, Cham, 2013, pp. 212–227. doi:10.1007/978-3-319-02414-1_16.
URL https://doi.org/10.1007/978-3-319-02414-1_16

[59] T. Baker, M. Mackay, M. Randles, A. Taleb-Bendiab, Intention-oriented programming support for runtime adaptive autonomic cloud-based applications, Computers & Electrical Engineering 39 (7) (2013) 2400 – 2412. doi:https://doi.org/10.1016/j.compeleceng.2013.04.019.
URL http://www.sciencedirect.com/science/article/pii/S0045790613001158

[60] Y. Karam, T. Baker, A. Taleb-Bendiab, Security support for intention driven elastic cloud computing, in: Sixth UKSim/AMSS European Symposium on Computer Modeling and Simulation, EMS 2012, Malta, November 14-16, 2012, IEEE, 2012, pp. 67–73. doi:10.1109/EMS.2012.17.
URL https://doi.org/10.1109/EMS.2012.17

[61] A. Kumar, S. R. Sabbella, R. R. Barton, Managing controlled violation of temporal process constraints, in: H. R. Motahari-Nezhad, J. Recker, M. Weidlich (Eds.), Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings, Vol. 9253 of Lecture Notes in Computer Science,

Springer, 2015, pp. 280–296. doi:10.1007/978-3-319-23063-4_20.
URL http://dx.doi.org/10.1007/978-3-319-23063-4_20

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

[63] J. W. Lloyd, Foundations of Logic Programming, 2nd Edition, Springer, 1987.

[64] D. Singh, C. K. Reddy, A survey on platforms for big data analytics, Journal of Big Data 2 (1) (2015) 8. doi:10.1186/s40537-014-0008-6.
URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4505391/

[65] T. Eiter, G. Gottlob, The complexity of logic-based abduction, J. ACM 42 (1) (1995) 3–42. doi:10.1145/200836.200838.
URL http://doi.acm.org/10.1145/200836.200838

[66] M. Montali, Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, Vol. 56 of Lecture Notes in Business Information Processing, Springer, 2010. doi:10.1007/978-3-642-14538-4.
URL http://dx.doi.org/10.1007/978-3-642-14538-4

[67] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, S. Storari, Declarative specification and verification of service choreographiess, TWEB 4 (1) (2010) 3:1–3:62. doi:10.1145/1658373.1658376.
URL http://doi.acm.org/10.1145/1658373.1658376

[68] Apache Samza, http://samza.apache.org (accessed 06.08.2017).

[69] Google Cloud Dataflow, https://cloud.google.com/dataflow/ (accessed 06.08.2017).

[70] J. Samosir, M. Indrawan-Santiago, P. D. Haghighi, An evaluation of data stream processing systems for data driven applications, in: M. Connolly (Ed.), International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, Vol. 80 of Procedia Computer Science, Elsevier, 2016, pp. 439–449. doi:10.1016/j.procs.2016.05.322.
URL http://dx.doi.org/10.1016/j.procs.2016.05.322

[71] F. Chesani, A. Ciampolini, D. Loreti, P. Mello, Abduction for generating synthetic traces, accepted for publication in First Workshop on BP Innovations with Artificial Intelligence, @ Business Process Management Conference (BPM) 2017, 10 - 15 September 2017, Barcelona, Spain (2017).

[72] IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams (Nov 2016). doi:10.1109/IEEESTD.2016.7740858.

[73] 8th international workshop on business process intelligence 2012 (2012).
URL http://www.win.tue.nl/bpi/doku.php?id=2012:challenge

[74] B. van Dongen, Bpi challenge 2012. eindhoven university of technology. dataset. (2012). doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
URL https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

[75] J. de Lara, A. Zisman (Eds.), Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings, Vol. 7212 of Lecture Notes in Computer Science, Springer, 2012. doi:10.1007/978-3-642-28872-2.
URL http://dx.doi.org/10.1007/978-3-642-28872-2

[76] S. Khurshid, K. Sen (Eds.), Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers, Vol. 7186 of Lecture Notes in Computer Science, Springer, 2012. doi:10.1007/978-3-642-29860-8.
URL http://dx.doi.org/10.1007/978-3-642-29860-8