A Cloud-Based Prediction Framework for Analyzing Business Process Performances

Eugenio Cesario, Francesco Folino, Massimo Guarascio, Luigi Pontieri

▶ **To cite this version:**

Eugenio Cesario, Francesco Folino, Massimo Guarascio, Luigi Pontieri. A Cloud-Based Prediction Framework for Analyzing Business Process Performances. International Conference on Availability, Reliability, and Security (CD-ARES), Aug 2016, Salzburg, Austria. pp.63-80, 10.1007/978-3-319-45507-5_5 . hal-01635015

HAL Id: hal-01635015

https://inria.hal.science/hal-01635015

Submitted on 14 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

A Cloud-based Prediction Framework for Analyzing Business Process Performances

Eugenio Cesario, Francesco Folino, Massimo Guarascio, Luigi Pontieri

ICAR-CNR, National Research Council of Italy, via P. Bucci 41C, 87036, Rende, CS, Italy
{cesario,ffolino,guarascio,pontieri}@icar.cnr.it

Abstract. This paper presents a framework for analyzing and predicting the performances of a business process, based on historical data gathered during its past enactments. The framework hinges on an inductive-learning technique for discovering a special kind of predictive process models, which can support the run-time prediction of some performance measure (e.g., the remaining processing time or a risk indicator) for an ongoing process instance, based on a modular representation of the process, where major performance-relevant variants of it are equipped with different regression models, and discriminated through context variables. The technique is an original combination of different data mining methods (namely, non-parametric regression methods and a probabilistic trace clustering scheme) and ad hoc data transformation mechanisms, meant to bring the log traces to suitable level of abstraction. In order to overcome the severe scalability limitations of current solutions in the literature, and make our approach really suitable for large logs, both the computation of the trace clusters and of the clusters' predictors are implemented in a parallel and distributed manner, on top of a cloud-based service-oriented infrastructure. Tests on a real-life log confirmed the validity of the proposed approach, in terms of both effectiveness and scalability.

Keywords: Data Mining, Prediction, BPM, Cloud/Grid computing.

1 Introduction

In many real-life application contexts, business processes are bound to the achievement of goals expressed in terms of performance measures (possibly representing an indicator security/risk or a measurement of quality), which are monitored continuously at run-time. Historical log data, gathered during past enactments of a process, are a valuable source of hidden information on the behavior of the process, which can be extracted with the help of process mining techniques [1], and exploited to improve the process, and meet such performance-oriented goals. In particular, it is definitely relevant to this regard the recent stream of research on the automated discovery of predictive process models (see, e.g., [2, 7]), capable to estimate a given performance measure over new instances of a process, as long as they are carried out. Indeed, such performance forecasts can help optimize the process at run time, by possibly developing advanced operational support services, such as task/resource recommendation [11], and the notification of alerts (possibly associated with some form of diagnostics).

Technically, all current approaches to the (log-based) discovery of a performance prediction model rely on abstracting the given log traces into a summarized propositional

form, suitable for recognizing patterns that are likely to be correlated to the analyzed performance measure (constituting the target of the prediction task). For example, in [2], an annotated finite-state machine (AFSM) model is induced from a given log, where the states correspond to different abstract representations of all the sequences of process activities appearing in the log. The discovery of such AFSM models was combined in [7] with a context-driven (predictive) clustering approach, so that different execution scenarios can be discovered for the process, and equipped with distinct local predictors. However, choosing the right abstraction level is a delicate task, requiring to reach an optimal balance between the risks of overfitting and of underfitting.

In order to free the analyst from the responsibility of choosing the right level of abstraction for the log's traces, it was proposed in [8] to induce the prediction model of each discovered trace cluster by applying any standard regression method to propositional representation of the traces falling in the cluster. The context-aware prediction models obtained with such a clustering-based learning strategy were empirically proven to improve the accuracy of previous solutions. However, both the approaches in [7, 8] require that a number of frequent structural patterns are preliminary extracted from the log traces, in order to map the latter onto a space of performance-oriented target features, in order to guide the clustering phase towards the discovery of groups of traces with similar performance trends. Once such clusters have been discovered, a distinct performance predictor can be induced from each cluster, by either using an AFSM-based learner [7] or standard regression algorithms [8]. Such a pattern-based clustering technique suffers from two main drawbacks: *(i)* the computation of structural patterns is not guaranteed to scale well over large logs, seeing as the number of discovered patterns may be, in general, combinatorial in the number of process activities; *(ii)* the discovered trace clusters are not guaranteed to reach the highest predictive clustering model, due to the loss of information determined when converting the log into a propositional dataset.

Contribution Still relying on the core idea of [7, 8] of combining performance prediction with a clustering technique, we try to overcome the two major limitations mentioned above at two levels. First, we resort to a rougher form of log sketch than [7, 8] for clustering the log traces, which can be quickly computed by picking up a fixed number of performance values at predefined positions within the traces, but are yet capable of helping recognize groups of traces exhibiting similar performance over the time —as confirmed by our experimental findings. Moreover, we replace the traditional (hard) clustering of the logics-based predictive clustering framework [3] used in [7, 8] with a probabilistic clustering scheme, in order to reduce the risk of obtaining lowly accurate cluster predictors (due to the greedy clustering algorithm and to the underlying approximated representation of the log). In order to overcome the severe scalability limitations of [7, 8] and make our approach suitable for large logs, both the computation of (probability-aware) trace clusters and of the clusters' predictors are implemented in a parallel and distributed manner, according to the Grid-services-based conceptual architecture defined in [4] for the specification and execution of *Distributed Data Mining* (DDM) tasks. The underlying grid services were developed according to the WSRF (Web Services Resource Framework) specifications of the WS-Core (a component of the Globus Toolkit 4 (GT4) [9]), and deployed onto a private Cloud-computing platform. By the way, the usage of a Cloud infrastructure to automatically deploy virtual

machines hosting a GT4 container is not new (see, e.g., [10, 12]), and it widely reckoned as a successful way of providing a high flexible and customizable environment for transparently and efficiently running costly computational tasks.

2 Preliminaries

Log data As usually done in the literature, we assume that for each process instance (a.k.a "case") a *trace* is recorded, storing the sequence of *events* happened during its unfolding. Let $\mathcal{T}$ be the universe of all (possibly partial) traces that may appear in any log of the process under analysis. For any trace $\tau \in \mathcal{T}$, $len(\tau)$ is the number of events in τ, while $\tau[i]$ is the i-th event of τ, for $i = 1 \,..\, len(\tau)$, with $task(\tau[i])$ and $time(\tau[i])$ denoting the task and timestamp of $\tau[i]$, respectively. We also assume that the first event of each trace is always associated with task A_1, acting as unique entry point for enacting the process. This comes with no loss of generality, seeing as, should the process not have such a such a unique initial task, it could be added artificially at the beginning of each trace, and associated with the starting time of the corresponding process instance.

Let us also assume that, like in [7], for any trace τ, a tuple $context(\tau)$ of data is stored in the log to keep information about the execution context of τ, ranging from internal properties of the process instance to environmental factors pertaining the state of the process enactment system. For ease of notation, let $\mathcal{A}^{\mathcal{T}}$ denote the set of all the tasks (a.k.a., activities) that may occur in some trace of $\mathcal{T}$, and $context(\mathcal{T})$ be the space of context vectors — i.e., $\mathcal{A}^{\mathcal{T}} = \cup_{\tau \in \mathcal{T}} tasks(\tau)$, and $context(\mathcal{T}) = \{context(\tau) \mid \tau \in \mathcal{T}\}$.

Further, $\tau(i]$ is the *prefix* (sub-)trace containing the first i events of a trace τ and the same context data (i.e., $context(\tau(i] = context(\tau)$), for $i = 0 \,..\, len(\tau)$.

A *log* L is a finite subset of $\mathcal{T}$, while the *prefix set* of L, denoted by $\mathcal{P}(L)$, is the set of all the prefixes of L's traces, i.e., $\mathcal{P}(L) = \{\tau(i] \mid \tau \in L \text{ and } 1 \leq i \leq len(\tau)\}$.

Let $\hat{\mu} : \mathcal{T} \to \mathbb{R}$ be an (unknown) function assigning a performance value to any (possibly unfinished) trace. For the sake of concreteness, we will focus next on a particular instance of such a function, where the performance measure corresponds to the *remaining time* (denoted by μ_{RT}), i.e. the time needed to finish the respective process instance. In general, we assume that performance values are known for all prefix traces in $\mathcal{P}(L)$, for any given log L. This is clearly true for the measure mentioned above. Indeed, for each trace τ, the (actual) remaining-time of $\tau(i]$ is $\hat{\mu}_{RT}(\tau(i]) = time(\tau[len(\tau)]) - time(\tau[i])$.

SOA, OGSA and WSRF The *Service oriented architecture (SOA)* is a model for building flexible, modular, and interoperable software applications. The key aspect of SOA is the concept of *service*, a software block capable of performing a given task or business function. The most popular implementation of SOA is represented by *Web Services*, whose popularity is mainly due to the adoption of widely accepted technologies such as XML, SOAP, and HTTP. Also the Grid provides a SOA framework whereby a great number of services can be dynamically located, balanced, and managed, so that applications are always guaranteed to be securely executed, according to the principles of on demand computing. The Grid community has adopted the *Open Grid Services Architecture (OGSA)* as an implementation of the SOA model within the Grid context. In OGSA every resource is represented as a Web Service that conforms to a set of con-

ventions and supports standard interfaces. OGSA provides a well-defined set of Web Service interfaces for the development of interoperable Grid systems and applications.

WSRF has been defined as an evolution of early OGSA implementations [5]. WSRF defines a family of technical specifications for accessing and managing stateful resources using Web Services, as required by OGSA. In other words, WSRF depicts some specifications to implement OGSA-compliant Web Services. Another way of expressing this relation is that, while OGSA is the architecture, WSRF is the infrastructure on which that architecture is built on [13]. The composition of a Web Service and a stateful resource is termed as *WS-Resource*. The possibility to define a state associated with a service is the most important difference between WSRF–compliant Web Services, and pre-WSRF ones. This is a key feature in designing Grid applications, since WS-Resources provide a way to represent, advertise, and access properties related to both computational resources and applications. In order to implement services in a highly decentralized way, it is commonly used a design pattern that allows a client to be notified when interesting events happen in a server. The *WS-Notification* specification defines a *publish/subscribe* notification model for Web Services, which is exploited to notify interested clients and/or services about changes that occur to the status of a WS-Resource. In other words, a service can publish a set of topics that a generic client can subscribe to; as soon as the topic changes, the client receives a notification of the new status.

Cloud Computing Cloud computing can be defined as a distributed computing paradigm in which all the resources, dynamically scalable and often virtualized, are provided as services over the Internet. Cloud systems are typically classified on the basis of their service model (*Software-as-a-Service*, *Platform-as-a-Service*, *Infrastructure-as-a-Service*) and their deployment model (*public cloud*, *private cloud*, *hybrid cloud*). *Software-as-a-Service* (or SaaS) defines a delivery model in which software and data are provided through Internet to customers as ready-to-use services (e.g. Google Docs or MS Office 365). In this case, both software and data are hosted by providers, and customers typically can access them in a easy way (without using any additional hardware or software) on a pay-per-use basis. *Platform-as-a-Service* (or PaaS) provides users with a programming environment and APIs for creating Cloud-based applications exploiting the computational resources of the platform (like e.g. Google App Engine, and Microsoft Azure). Finally, *Infrastructure-as-a-Service* (or IaaS) is a model under which customers ask for physical resources (e.g. CPUs, disks) to support their computational requirements (e.g., Amazon EC2, RackSpace Cloud). Cloud computing services are further classified according to three main deployment models: *public*, *private*, and *hybrid*. In a public cloud model, providers directly manage their data centers, and publicly delivers services built on top of them through the Internet. In a private cloud schema, operations and functionalities are offered as-a-service and usually hosted in a company intranet or in a remote data center. Finally, a hybrid cloud is obtained by composing two or more (private or public) clouds that, yet linked together, remain different entities.

3 Reference Architecture for Distributed Data Mining

This section is meant to briefly illustrate the service-oriented architecture proposed in [4] for carrying out Distributed Data Mining (*DDM*) tasks over a Grid.

The architecture, shown in Figure 1, supports the specification and execution on the Grid of DDM algorithms, designed according to a master-worker pattern. Specifically, it features two kinds of Grid Services: the *GlobalMiner-WS* and the *LocalMiner-WS*. The architecture was conceived to follow a typical DDM schema contemplating the presence of an entity acting as coordinator (the *GlobalMiner-WS*) and a certain number of entities acting as miners (*LocalMiner-WS*) on local sites. A resource is associated with each service: the *GlobalModel Resource* with the *GlobalMiner-WS* and the *LocalModel Resource* with the *LocalMiner-WS*. Such resources are used to store the state of the services, in this case represented by the computed models (globally and locally, respectively). Additionally, the resources are published also as *topics*, in order to be considered as "items of interest of subscription" for *notifications*. As it appears clearly in Figure 1, the two types of nodes are equipped with code libraries that implement the mining algorithms to be executed, which are named *Global Algorithm Library (GAL)* and a *Local Algorithm Library (LAL)*, respectively. In the following we describe all the steps composing the whole process, by pointing out details of the interactions between entities composing the architecture. Let us suppose that a client wants to execute a distributed mining algorithm on a dataset D, which is split in N partitions, $\{D_1,...,D_N\}$, each one stored on one of the nodes $\{Node_1,...,Node_N\}$. In order to be processed correctly, any request to perform a mining task needs to follow a three-step scheme.

In the first step, a client wanting to submit a request must invoke the `createResource` operation of the *GlobalMiner-WS* to create a *GlobalModel Resource*. In turn, the *GlobalMiner-WS* dispatches this operation in a similar way: for each local site with a running *LocalMiner-WS* (assuming the *GlobalMiner-WS* to hold a list of them), it invokes the `createResource` operation (of the *LocalMiner-WS*) to create a *LocalModel Resource*. At this point, the client invokes the `subscribe` method to subscribe a listener (inside the Client) as consumer of notifications on the *GlobalModel* topic. Finally, the *GlobalMiner-WS* invokes the `subscribe` method to subscribe a listener as consumer of notifications on the *LocalModel* topic. As an effect of such subscription steps, as soon as a resource changes, its new value will be delivered to its listener.

The second step constitutes the core of the application, i.e., the execution of the mining process and the result return. The Client invokes the `submitGlobalTask` operation, by passing a *GlobalTaskDescriptor*, i.e. a complete description of the task to be executed (algorithm name, parameters, initialization type, etc.). By interacting with the *Global Algorithm Library*, the *GlobalMiner-WS* runs the code executing the steps to be done. During this phase, suitable data structures are initialized and a suitable set of *LocalTaskDescriptor* are created. It invokes the `submitLocalTask` operation, by passing a LocalTaskDescriptor. At this stage, each i^{th} *LocalMiner-WS* begins the analysis of the local dataset for computing a local model and local statistics. Such a task is executed, with the support of the *LocalAlgorithmLibrary*, concurrently on different Grid sites. While every local task is in progress, the *GlobalMiner-WS* does not need to periodically poll if they are terminated: the notification mechanism, indeed, is able to deliver asynchronous messages warning about the termination of local tasks. As soon as a local computation terminates, the value of the *LocalModel Resource* is set by the value of the computed local model. The changes in this resource are automatically notified to the listener on the *GlobalMiner-WS*; this way, the local model computed at the i^{th}

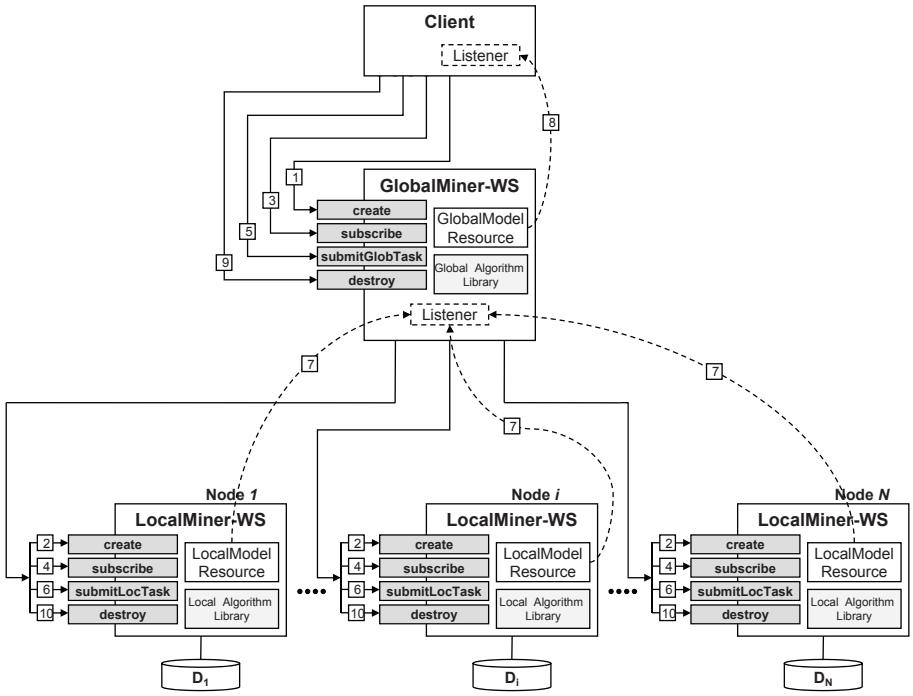


Fig. 1: Architectural Model of Distributed Data Mining Services.

local site is delivered to the global site. As soon as all the local models are delivered to the *GlobalMiner-WS*, the integration of all these local models is performed. According to the logic of the chosen algorithm (provided by the *GlobalAlgorithmLibrary*), the *GlobalMiner-WS* evaluates whether the algorithm is terminated (or not). If it is, the global model computed by the *GlobalMiner-WS* is stored on the *GlobalModel Resource*, which is immediately delivered (via the notification mechanism) to the client. Otherwise, the *GlobalMiner-WS* asks for further processing, by invoking one more time a `submitLocalTask` operation and waiting for the delivering of the result. Such latter actions are executed as many times as the *GlobalMiner-WS* needs, until the computation reaches some convergence condition.

In the final step, as soon as the computation terminates and its results have been notified to the Client, the Client invokes the `destroy` operation of the *GlobalMiner-WS*, which eliminates the resource previously created (*GlobalModel*). Similarly, the *GlobalMiner-WS* asks for the destruction of the *LocalModel*.

From a practical (user-side) point-of-view, any new DDM task can be executed on this architecture provided that it suitably implements the interfaces `globalAlgorithm` and `localAlgorithm`. Specifically, they export the signatures for those methods each distributed algorithm running on this architecture needs to invoke, i.e. *initialize*, *computeGlobalModel*, *needsMoreIteration* and *finalize* for the `globalAlgorithm`; *initialize*, *computeLocalModel* and *finalize* for the `localAlgorithm`.

4 Formal Framework for Process Performance Prediction

The ultimate goal of this work is to devise a scalable approach to the discovery of a (predictive) *Process Performance Model (PPM)*, capable to accurately predict the performance outcome of any ongoing process case, based on the information stored in its associated (partial) trace. Such a model can be viewed as a function $\mu : \mathcal{T} \rightarrow \mathbb{R}$ that provides an estimate for $\hat{\mu}$ all over the trace universe —including the prefix traces of all possible process instances. Discovering a PPM is an inductive learning task, where the training set takes the form of a log L, and the value $\hat{\mu}(\tau)$ of the target measure is known for each (sub-)trace $\tau \in \mathcal{P}(L)$. Notably, current approaches to this problem [2, 7, 8] rely on preliminary converting the given process traces in to a propositional form, with the help of some suitable trace abstraction function. The rest of this section provides the reader with a few technical details on the trace abstraction functions considered in our approach, as well as on the specific (clustering-based and probability-aware) kind of PPM that we want to eventually learn from the abstracted traces.

Trace Abstraction An abstracted (structural) view of a trace summarizes the tasks executed during the corresponding process enactment. Two simple ways to build such a view consist in regarding the trace as a tasks' set or multiset (a.k.a. bag), as follows.

Definition 1 (Structural Trace Abstraction). Let $\mathcal{T}$ be a trace universe and $A_1, \ldots, A_n$ be the tasks in $\mathcal{A}^{\mathcal{T}}$. A *structural (trace-) abstraction function* $struct^{mode} : \mathcal{T} \rightarrow \mathcal{R}_{\mathcal{T}}^{mode}$ is a function mapping each trace $\tau \in \mathcal{T}$ to an an *abstract representation* $struct^{mode}(\tau)$, taken from an *abstractions' space* $\mathcal{R}_{\mathcal{T}}^{mode}$. Two concrete instantiations of the above function, denoted by $struct^{bag} : \mathcal{T} \rightarrow \mathbb{N}^n$ (resp., $struct^{set} : \mathcal{T} \rightarrow \{0,1\}^n$), are defined next, which map each trace $\tau \in \mathcal{T}$ to a bag-based (resp., set-based) representation of its structure: *(i)* $struct^{bag}(\tau) = \langle count(A_1, \tau), \ldots, count(A_n, \tau) \rangle$, where $count(A_i, \tau)$ is the number of times that task A_i occurs in τ; and *(ii)* $struct^{set}(\tau) = \langle occ(A_1, \tau), \ldots, occ(A_n, \tau) \rangle$, where $occ(A_i, \tau) = \texttt{true}$ iff $count(A_i, \tau) > 0$, for $i = 1, \ldots, n$. □

The two concrete abstraction "modes" (namely, *bag* and *set*) defined above summarize any trace τ into a vector, where each component corresponds to a single process task A_i, and stores either the number of times that A_i appears in the trace τ, or (respectively) a boolean value indicating whether A_i occur in τ or not. Notice that, in principle, we could define abstract trace representations as sets/bags over another property of the events (e.g., the executor, instead of the task executed), or even over a combination of event properties (e.g., the task plus who performed it).

As a matter of fact, the structural abstraction functions in Def. 1 are similar to those used in previous approaches to the discovery of predictive process models [2, 7, 8].

PPM discovery as predictive clustering: limitations of current solutions From a conceptual point of view, we rephrase the induction of a PPM, out of a given log, as a predictive clustering [3] task, similarly to [7, 8]. Essentially, the core idea underlying predictive clustering approaches is that suitably partitioning the training instances into clusters and inducing a specific predictor from each cluster helps obtain better predictions than using a single predictor (learnt from the whole training set). At run-time,

for any new instance, the prediction is built by first assigning the instance to one of the clusters, and then making it undergo the predictor of that cluster. More precisely, these approaches assume that two kinds of features are available for any element z in the reference instance space, say $Z = X \times Y$: *descriptive* features and *target* features (to be predicted), denoted by $descr(z) \in X$ and $targ(z) \in Y$, respectively. Then, a *predictive clustering model* (PCM), for a given training set $L \subseteq Z$, is a function $q : X \to Y$ of the form $q(x) = p(c(x), x)$, where $c : X \to \mathbb{N}$ is a partitioning function that assigns x to a cluster, and $p : \mathbb{N} \times X \to Y$ is a (possibly multi-target) prediction function. Clearly, whenever there are more than one target features, q encodes a multi-regression model.

Following this general scheme, and similarly to [7, 8], we adopt a modular kind of PPM that consists of two different types of components: a partitioning function c that maps any trace to a distinguished cluster, and a collection $\mu_1, \ldots, \mu_k$ of PPMs, one per cluster. A propositional encoding $enc(\tau)$is exploited for any possible trace τ, which mixes up the context data possibly available for τ with an abstracted view of the history of τ, obtained with the help of one of the abstraction function in Definition 1. The features appearing in $enc(\tau)$ play the role of descriptive attributes, while clearly considering the the performance values $\mu(\tau)$ as the target of prediction. Clearly, applying standard predictive clustering solutions to such data would not make sense, since we are not interested in predicting the final performance measure of a complete trace (as are those stored in the above-described training set), but rather in making predictions on all possible prefixes (i.e. partial incremental versions) of any new trace as far as it unfolds, in a step-by-step fashion —as discussed in [7], learning the clustering model based on all partial traces in $\Pr(L)$ is costly and hardly effective. Therefore, both the approaches in [7, 8] faced the problem heuristically, by dividing it into two sub-problems: *(1)* induce a predictive clustering model out of a summarized representation (named "log sketch") of the given log, where each (full) trace in the log is used as a single clustering instances, associated with a number of target values that capture the evolution of the μ at several relevant ("pivot") stages of the process *(2)* induce a PPM out of each discovered cluster. In particular, in [7, 8], the computation of the target attributes for the traces relied on the preliminary extraction of frequent structural patterns chosen greedily, based on their apparent capability to discriminate among different performance profiles.

However, such a approach suffers from two main drawbacks: *(i)* the computation scheme cannot scale over large logs, mainly due to the fact that the extraction of structural patterns may take a time that is combinatorial in the number of process activities; *(ii)* the discovered clusters of traces are not guaranteed to reach the highest predictive clustering model (due to both the very two-phase optimization scheme, and to the loss of information determined by converting the original log into a propositional dataset).

A new kind of (clustering-based) PPMs: `PCB-PPM` We here try to overcome both these limitations at two levels. First, we resort to a rougher form of log sketch than [7, 8] for clustering the log traces, which does not require the computation of frequent structural patterns, but simply consists on extracting a fixed number of performance values at predefined positions of the traces. These additional (target-oriented) features, which are formally defined in the next Section (cf. Definition 3), are meant to introduce a bias towards the discovery of groups of traces exhibiting similar performance patterns over the time. On the other hand, we extend the traditional (hard) clustering of standard pre-

dictive clustering settings with a probabilistic clustering scheme. This choice is meant to curb the risk of obtaining a poorly accurate PPM as a result of the heuristics clustering strategy adopted in [7, 8] (due to both the very greedy clustering method and the usage of an approximated representation of the traces). In particular, as discussed in the following section, such a probabilistic clustering model is computed efficiently trhough a parallelized distributed version of the well-known *EM* method [6], hence solving the scalability limitations of [7, 8].

We are now in a place to formally define the novel, probabilistic-aware clustering-based, type of PPM that our learning approach is meant to extract from a given log.

Definition 2 (Probabilistic Clustering-Based PPM Model (`PCB-PPM`)). Let L be a log (over $\mathcal{T}$), with context features $context(\mathcal{T})$, structural abstract representations $struct^{mode}(\mathcal{T})$ (where $mode \in \{set, bag\}$), and $\hat{\mu} : \mathcal{T} \rightarrow \mathbb{R}$ be a performance measure, known for all $\tau \in \mathcal{P}(L)$. Let also $enc(\tau) = context(\tau) \oplus struct^{mode}(\tau)$ be the propositional encoding for each $\tau \in \mathcal{P}(L)$, where $\oplus$ stands for tuple concatenation. Then a *probabilistic clustering-based performance prediction model* (`PCB-PPM`) for L is a pair $\mathcal{M} = \langle c, \langle \mu_1, \ldots, \mu_k \rangle \rangle$ (where k is the number of clusters found for L) encoding a probabilistic predictive clustering model, where *(i)* $c : enc(\mathcal{T}) \times \{1, \ldots, k\} \rightarrow [0, 1]$ is a probabilistic clustering function that assigns any (possibly partial) new trace τ to each cluster with a certain probability (based on the encoding of τ), and *(ii)* $\mu_i : \mathcal{T} \rightarrow \mathbb{R}$ is a *PPM* associated with the i-th cluster, for $i \in \{1, \ldots, k\}$. Model $\mathcal{M}$ is meant to estimate the unknown performance function $\hat{\mu}$ as follows: for any trace $\tau \in \mathcal{T}$, the corresponding performance value $\hat{\mu}(\tau)$ is computed as $\sum_{j=1}^{k} c(enc(\tau), j) \cdot \mu_j(enc(\tau))$. □

In such a model, each cluster is equipped with a separate PPM, tailored to capture how $\hat{\mu}$ depends on both the structure and context of any trace that may be assigned to the cluster. The prediction for any new trace τ is computed as a linear combination of the predictions made by the PPMs of all the clusters τ is estimated to possibly belong to, with their respective membership probabilities used as weights. In general, such an articulated kind of PPM can be built by inducing a predictive clustering model and multiple PPMs (as the building blocks implementing c and all μ_i, respectively). In our approach, we face this sub-task by resorting to standard regression methods, defined for propositional data. As observed in [8], indeed, this frees the analyst from the burden of defining a suitable level of details over the representation of the relevant states of the process (as required in the case of the AFSM models used in [2, 7]).

5 Solution Approach to the Discovery of a `PCB-PPM`

Figure 2 illustrates the main steps of our approach to the discovery of a `PCB-PPM` model, in the form of an algorithm, named `PCB-PPM Discovery`. Essentially, the problem is approached in three main phases. In the first phase (Step 1), the given log L is transformed into a propositional dataset D_{traces}, containing one distinguished tuple for each trace of the log. As in [7, 8], the tuple encoding any trace τ in L stores both the context-oriented attributes of τ and the summarized structural view $struct^{mode}(\tau)$, produced according to the abstraction criterion *mode* (specified as input to the algorithm) —these two kinds of information are here represented as two different sub-tuples, concatenated

Input: A log L over some trace universe $\mathcal{T}$, with associated target performance measure $\hat{\mu}$, an abstraction mode $mode \in \{set, bag\}$, a regression method *REGR*, the number K of desired trace clusters, and the number M of local miners for the clustering.
Output: A `PCB-PPM` model for L (fully encoding $\hat{\mu}$ all over $\mathcal{T}$).
Method: Perform the following steps:
1 $D_{traces} := \{\ context(\tau) \oplus struct^{mode}(\tau) \oplus pProf(\tau) \mid \tau \in L\ \}$; *// see Def. 3.*
2 Randomly split D_{traces} into M (nearly) equally sized datasets $D_1, \ldots, D_M$;
3 $C :=$ `P-EM`$(D_1, \ldots, D_M, K)$; *// compute a probabilistic clustering model C via EM.*
4 $D_{prefixes} := \{\ context(\tau) \oplus struct^{mode}(\tau) \oplus \hat{\mu}(\tau) \mid \tau \in Pr(L)\ \}$;
5 Let $D'_1, \ldots, D'_K$ be the K clusters of trace prefixes resulting from applying C to $D_{prefixes}$; *// each $z \in D_{prefixes}$ is assigned to cluster D'_i iff $i =$* $\underset{j \in \{1,\ldots,K\}}{\text{argmax}}\ C(z, j)$.
6 $\langle r_1, \ldots, r_K \rangle :=$ `P-REGR`$(D'_1, \ldots, D'_K, REGR)$; *// r_i is the regression model obtained by applying the learning method REGR to the tuples in D'_i, while regarding their last field as target variable, and all of the other ones as predictor variables.*
7 Encode C into a probabilistic clustering model $\hat{c} : enc(\mathcal{T}) \times \{1, \ldots, K\} \to [0, 1]$, and each r_i into a prediction model $\mu_i : \mathcal{T} \to M$; *// where $enc(\mathcal{T}) = context(\mathcal{T}) \times struct^{mode}(\mathcal{T})$.*
8 **return** $\langle\ \hat{c}, \langle \mu_1, \ldots, \mu_K \rangle\ \rangle$.

Fig. 2: **Algorithm** `PCB-PPM Discovery`.

one with the other. A series of performance-oriented attributes (denoted as $pProf(\tau)$) are stored as well in the propositional representation of τ, in order to bias the clustering phase towards the discovery of groups of traces that exhibit similar performance patterns over the time. These trace attributes are formally defined next.

Definition 3 (Performance profile (for trace clustering)). Let $\mathcal{T}$ be a trace universe, and $\hat{\mu} : \mathcal{T} \to \mathbb{R}$ be a performance measure. Let h be a number in $\mathbb{N}^+$ chosen by the analyst[1]. For any trace τ in $\mathcal{T}$, the *performance profile* of τ (w.r.t. $\hat{\mu}$ and m) is an h-sized (sub-)tuple $pProf(\tau) = \langle pProf(\tau, 1), pProf(\tau, 2), \ldots, pProf(\tau, h) \rangle$, such that, for each $j \in \{1, \ldots, h\}$, it holds:

$$pProf(\tau, j) = \begin{cases} \tau[i_j], & \text{if } i_j < len(\tau) \\ \texttt{NULL}, & \text{otherwise} \end{cases}$$

where $i_j = (j-1) \times \left\lfloor \frac{len(\tau)}{h} \right\rfloor + 1$. □

Notably, computing such a representation of a trace's performance profile simply amounts to picking up a fixed number of performance values at predefined positions of the trace, which can be done in linear time in the size of the input log. By converse, the propositional log sketch used in [7, 8] for trace clustering requires the preliminary extraction of frequent structural patterns, which is far more expensive (possibly combinatorial in the number of process activities) in terms of computation time.

In the second phase (Steps 2-3), a probabilistic clustering model is mined out of D_{traces} by exploiting a distributed version of popular algorithm *EM* [6], implemented

[1] In all the tests described here, we simply set h as the 40th percentile of the log's traces length.

by function P-EM according to the data-parallel scheme of Figure 1. This function, described in details later on, requires the data to be preliminary split into different datasets $\{D_1, \ldots, D_M\}$, based on some partitioning strategy. Specifically, here we simply divided the tuples of D_{traces} into groups of (approximately) the same size (Step 2).

In the third phase (Steps 4-5), a regression model is induced from each of the discovered trace clusters, by applying the learning method *REGR* specified as input to the algorithm. In order to obtain a predictive model that can be applied to (new) ongoing process instances, the learner is provided with a set of training instances representing partial (prefix) traces, labelled each with the respective performance measurement. To this purpose, all the prefix traces in $Pr(L)$ are preliminary stored into a propositional dataset $D_{prefixes}$), which encodes all the context-oriented and structure-oriented features of each trace (in addition to its target performance value). The discovered clustering model C is then applied to $D_{prefixes}$ in a "hard" way, in order to obtain a partition $D'_1, \ldots, D'_K$ of it, where each tuple $z \in D_{prefixes}$ is assigned to the cluster it appears the most likely to belong to (Step 5). These clusters of prefix traces are then exploited to train a list of regression models, one per cluster. This is accomplished by the distributed function P-REGR (Step 6), following the same distributed paradigm as P-EM —further details on the function too are given later on. Clearly, in such a regression task, for each prefix trace τ, the features in $context(\tau)$ and $struct^{mode}(\tau)$ are considered as input values, while the performance measurement $\hat{\mu}(\tau)$ is the target variable to be predicted.

The rest of the algorithm is simply meant to put the discovered clustering model and regression models in the form of a PCB-PPM, and to eventually return the latter.

Function P-EM *Expectation Maximization* (*EM*) is a well-known method for inducing a probabilistic clustering model. The method relies on the assumption that the given dataset were generated by a *mixture* of K probability distributions of some given form (e.g., Gaussian), representing each a distinguished cluster. The clustering task then consists in trying to maximize the fit between the dataset and such a probabilistic model, by suitably setting, the membership probabilities of the data instances and the parameters of the overall generative model —indeed, this information is not assumed to be known in advance, and it must be estimated from the given data. Classic *EM* algorithm needs to be provided with an initial (typically randomly chosen) setting of the mixture model's parameters. Each EM iteration computes new estimates for these parameters that are proven not to decrease the likelihood of the model, denoted hereinafter as $Perf_{EM}$. The process is repeated until the value of $Perf_{EM}$ converges to a local maximum.

Different distributed versions of EM algorithm have been presented in the literature. Our approach essentially exploits the distributed computation scheme defined in [4], in accordance with the DDM framework of Figure 1.

In a nutshell, function P-EM takes as input different subsets, say $D_1, \ldots, D_N$, of the dataset, and the maximum number of clusters, say K. The clustering process is distributed among N local miners (i.e., nodes equipped with an instance of *LocalMiner-WS*, as explained in Section 3), storing each one of the sub-datasets $D_1, \ldots, D_N$, with a further node acting as coordinator (provided with an instance of a *GlobalMiner-WS*).

Specifically, in the initialization phase (method *initialize*), the coordinator randomly initializes, for each of cluster C_k (with $k = 1, \ldots, K$), the center m_k, the covariance matrix Σ_k and the mixing probability $p(m_k)$, and sends a copy of these data to each local node.

The method *computeLocalModel*, implemented by each local miner, consists of three steps that are described next. First, the local miner estimates (for each $x \in D$, for each $k = 1,\ldots,K$) the probability $p(m_k|x)$ that x belongs to cluster C_k —precisely, it computes $p(m_k|x) = \frac{p(x|m_k)\cdot p(m_k)}{\sum_{k=1}^{K} p(x|m_k)\cdot p(m_k)}$, where $p(m_k)$ is the mixing probability and $p(x|m_k)$ is the prior probability, taking the specific form of a Gaussian distribution. Then, it computes a collection $SS^{(i)}$ of local statistics, consisting of the following values: $f^{(i)} = \sum_{x\in D_i}[-\log\sum_{k=1}^{K} p(x|m_k)p(m_k)]$; $s1_k^{(i)} = \sum_{x\in D_i} p(m_k|x)$; $s2_k^{(i)} = \sum_{x\in D_i} p(m_k|x)x$; and $s3_k^{(i)} = \sum_{x\in D_i} p(m_k|x)(x-m_k)^T(x-m_k)$, (with $k = 1,\ldots,K$). As a last step of *computeLocalModel*, these local statistics are sent to the coordinator.

In order to implement method *computeGlobalModel*, the coordinator needs to combine local information stored in the data summaries $SS^{(1)},\ldots SS^{(N)}$ (received from the local nodes), and update the clustering model. In particular, for each cluster C_k, the updated versions of the center m_k, the covariance matrice Σ_k, the mixing probabilities $p(m_k)$ for $k,k = 1,\ldots,K$ are produced —namely, $m_k = \sum_{i=1}^{N} s2_k^{(i)} / \sum_{i=1}^{N} s1_k^{(i)}$, $\Sigma_k = \sum_{i=1}^{N} s3_k^{(i)} / \sum_{i=1}^{N} s1_k^{(i)}$, and $p(m_k) = \sum_{i=1}^{N} s1_k^{(i)} / |D|$. Moreover, a performance measure $Perf_{EM}$ is computed for the resulting clustering model as follows: $Perf_{EM} = \sum_{i=1}^{N} f^{(i)}$.

At the end of each iteration of this computation scheme, the coordinator must check (through an invocation of method *needsMoreIteration*) if the measure $Perf_{EM}$ has converged to a (local) optimum or the number of iterations has reached a given bound. If one of these conditions holds, the algorithm finishes; otherwise a new iteration of the algorithm starts, where the coordinator sends a copy of the novel clustering parameters (i.e., m_k, Σ_k and $p(m_k)$, for $k = 1,\ldots,K$) to the local miners, and so on.

Function `P-REGR` Due to space limitations, we do not describe this function in detail. In fact, rephrasing this function in terms of the framework in Figure 1 is simple enough for being explained without any precise formalization.

Basically, the method *initialize* of interface `globalAlgorithm` simply consists in assigning the given K datasets to different local miners (i.e. instances of *LocalMiner-WS*). Each local miner is responsible for extracting a single regressor out of one of the datasets by using the selected regression algorithm *REGR* (hence playing as the actual implementation of method *computeLocalModel* in `localAlgorithm`). Once each local regressor has been induced, it is passed to the global miner (i.e. the node implementing an instance of *GlobalMiner-WS*), which does not need to perform any further computation —method *needsMoreIteration* in `globalAlgorithm` is "immaterial" in that it always returns `false`. Finally, the method *computeGlobalModel* of `GlobalModel` only must store all the local regressors received from the remote sites, into a list, ensuring that they follow the same mutual order as the clusters they have been induced from.

6 Experimental Results

In order to test our approach, we implemented both the distributed functions `P-EM` and `P-REGR` (see Section 5) as a composition of Grid Services, according to the model of Figure 1. The services have been developed by using the Java WSRF library provided by the WS-Core, a component of the Globus Toolkit 4 [9]. These GT4-based Grid Services

Table 1: Average errors made by PCB-PPM Discovery (here denoted as Ours) and the competitors, for different abstraction modes (namely, *BAG* and *SET*). The best outcomes are in bold.

Metric	BAG					SET				
	Ours (IB-k)	Ours (RepTree)	AA-TP (IB-k)	AA-TP (RepTree)	CA-TP	Ours (IB-k)	Ours (RepTree)	AA-TP (IB-k)	AA-TP (RepTree)	CA-TP
rmse	0.217	0.213	0.205	**0.203**	0.291	0.299	0.294	0.287	0.286	0.750
mae	0.061	**0.059**	0.064	0.073	0.142	0.102	0.098	0.105	0.112	0.447
mape	0.117	**0.111**	0.119	0.189	0.704	0.225	0.189	0.227	0.267	2.816

were deployed onto a private Cloud computing infrastructure, by instantiating a pool of virtual machines (one for each grid node). Precisely, we used a physical Grid consisting of 18 nodes, each running a Linux CentOS 7.0 distribution and equipped with 1TB (SATA) hard drive, a dual-core processor Intel Xeon E2 2650 2GHz, and 128GB of RAM. For scalability analyses, different subsets of these nodes, namely, of size $N = 1, 2, 4, \ldots, 16$, were allocated to the execution of the whole PCB-PPM Discovery.

The *LocalMiner-WS* services (using either in P-EM or in P-REGR) were distributed as uniformly as possible over a number of dedicated virtual machines in the Cloud – in practice, M local miners were deployed onto V virtual machines by assigning M/V local miners to each machine. Each virtual machine, instead, is instantiated on a distinct node of the Grid. It is worth noticing that two distinguished nodes were kept reserved for the *GlobalMiner-WS* services of the functions P-EM and P-REGR, and another one for the main algorithm PCB-PPM Discovery —acting as a client of those *GlobalMiner-WS*'s.

The rest of the section discusses a series of experimental activities that we conducted to assess the validity of our approach, in terms of both effectiveness and scalability. To this purpose, we used a collection of 5336 log traces generated by a real transshipment system. Each trace stores a sequence of major logistic activities (4 on the average) that were applied to a distinguished container, passing through the system in the first third of year 2006. Basically, each container is unloaded from a ship, temporarily placed by the dock, and then carried to a yard slot for being stocked. Symmetrically, at boarding time, the container is first placed close to the dock, and then loaded on a cargo. Different kinds of vehicles can be used for moving a container, including, e.g., cranes, "straddle-carriers", and "multi-trailers". This basic life cycle may be extended with further transfers, devoted to make the container approach its final embark point or to leave room for other ones. Several data attributes are available for each container as context data, which include: the origin and final ports, its previous and next calls, various properties of the ship unloading it, physical features (such as, e.g., size, weight), and some information about its contents. As in [7], we also considered several "environmental" context features for each container: the hour (resp., day of the week, month) when it arrived, and the total number of containers that were in the port at that moment.

In all the tests described next, the remaining processing time (for all the prefix traces extracted by this log) was considered as the target performance measure to predict.

Effectiveness Results Since the ultimate goal of our approach is to predict a performance measure (namely, the remaining processing time) over partial log traces, the effectiveness of our approach was evaluated by computing (via 10-fold cross validation) three standard error metrics, quantifying all how much the predicted values differ

from the real ones in the average: *root mean squared error (rmse)*, *mean absolute error (mae)*, and *mean absolute percentage error (mape)*. For ease of interpretation, the results have been divided by the average processing time (over all the log's traces).

As a term of comparison we considered two approaches that were defined in the literature for the discovery of a clustering-based PPM: `AA-TP` [8], and `CA-TP` [7]. By the way, it is known from the literature such these techniques were all capable to neatly improve the achievements of non clustering-based predictors. In a sense, the quality of the forecasts that a clustering-based PPM eventually provides can be considered as an indirect indicator of the validity of its underlying clustering model (which has indeed a "predictive clustering" nature, and serves the purpose of supporting the prediction task). Following this line of reasoning, no quality measure is shown here for the discovered clusters, in addition to the very accuracy of the predictions that they allowed to make.

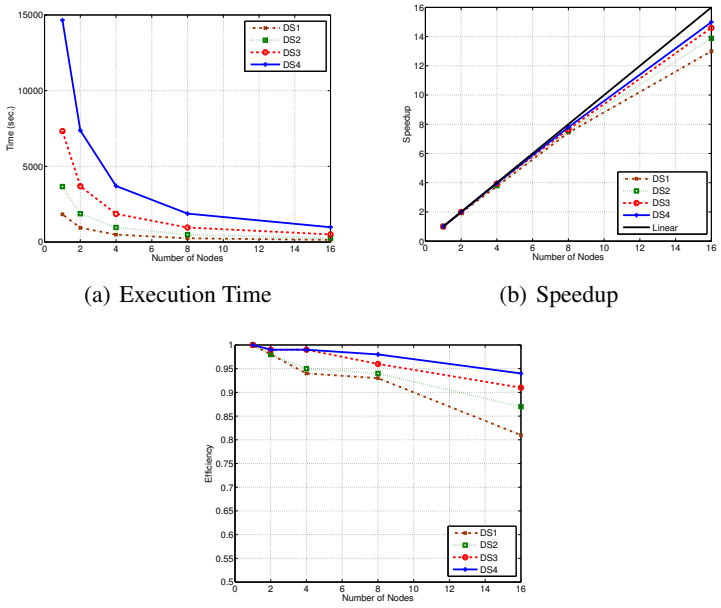
For our empirical effectiveness analysis, we run `PCB-PPM Discovery` by heuristically setting the number K of clusters to $\lceil log_2(|L|) \rceil$, where $|L|$ is the number of traces in the given log, while always keeping $M = 16$.

Tables 1 shows the average errors made by our algorithm, for two different instantiations of the regression method *REGR* (namely, and *IB-k* and *RepTree*) —in both cases we resorted to the implementations available in the popular Weka [14] library. In particular, the first half of the table regards the case when the *bag* mode is used for abstracting traces, whereas the second half concerns the usage of *set* abstractions.

The values shown for `AA-TP` were computed by averaging the ones obtained with different settings of its parameters, namely $minSupp \in \{0.1, \ldots 0.4\}$, $kTop \in \{4, \infty\}$, and $maxGap \in \{0, 4, 8, \infty\}$. For `CA-TP`, instead, we report the average of the results that it obtained with different values of the horizon parameter h (precisely, $h = 1, 2, 4, 8, 16$), and the best-performing setting for all the remaining parameters.

In order to correctly interpret the results in Table 1, it is important to notice that the kind of clustering used by `AA-TP` and `CA-TP` is more precise than the one adopted in our approach, as discussed in Section 4. In spite of this, `PCB-PPM` still achieves better results in terms of *mae* and *mape* than `AA-TP` (our *rmse* outcomes, instead, are slightly worse than `AA-TP`'s ones), and more accurate outcomes (over all error metrics) than `CA-TP`, irrespective of the abstraction function (i.e. *BAG* or *SET*) used. The former behavior can be explained as an effect of the probabilistic clustering scheme that underlies our `PCB-PPM`'s, and makes the prediction of trace performances more robust to the discovery of sub-optimal trace clusters. The lower prediction errors produced by both `PCB-PPM` and `AA-TP`, w.r.t. `CA-TP`, find a justification on their capability to fully exploit the context information available for the log traces in both the clustering and prediction phases —whereas `CA-TP` uses them during the clustering step only.

Efficiency Results To test the scalability of our approach, we first generated 4 distinct datasets with different sizes (namely, *DS*1, *DS*2, *DS*3, *DS*4), obtained by replicating every trace in the given log L for 256, 512, 1024, and 2048 times, respectively. We studied the scalability, speedup and efficiency performances when varying the number of grid nodes (from 1 to 16) that were actually used. For space reasons, we next focus on the case where $REGR = RepTree$ and $mode = BAG$ —similar trends were obtained, indeed, with the *SET* abstraction and *IB-k*. Similarly to what done in the effectiveness tests, the number of desired trace clusters was set to $\lceil log_2(|L|) \rceil$.



(a) Execution Time (b) Speedup

(c) Efficiency

Fig. 3: Efficiency results of PCB-PPM for different data sizes when *RepTree* is used as regressor.

Figure 3 (a) shows the *execution times* spent against each of the datasets, when using 1 to 16 Grid nodes. We can simply observe that these times strongly decrease for all datasets when incresing the number of available nodes. It is important to observe that the time needed to process the dataset *DS*4 (i.e., the biggest one, with more than 10M traces and 43M of events) is longer than 4 hours when using a single node, while it decreases to only 16.3 minutes when exploiting 16 nodes. In Figure 3 (b) the execution *speed-up* curves are depicted for each of the datasets. The speed-up is almost linear for all datasets up to the case where 8 Grid nodes are used; an appreciable trend of gain is maintained over higher numbers of nodes. Notably, a speed-up value of 15 is obtained for the extreme case of dataset *DS*4 when using all of the 16 nodes, hence substantiating the scalability of our distributed computation strategy and of the underlying framework. Finally, Figure 3 (c) shows the system *efficiency*, vs the number of nodes and for different datasets. As shown in the figure, a good efficiency trend can be seen as long as the number of nodes increases. As a matter of fact, for the largest dataset *DS*4, the efficiency on 8 nodes is equal to 0.99, whereas on 16 nodes it is equal to 0.94, i.e. the 99% and 94% of the computing power of each used node is exploited, respectively.

7 Conclusions

We have presented a novel context-aware clustering-based approach to the discovery of predictive models for supporting the forecast of process performance measures. The ap-

proach overcomes the severe scalability limitations of similar solutions currently available in the literature, and takes advantage of a distributed implementation of its more expensive computational tasks, based on a collection of ad hoc grid services, deployed on top of a cloud-computing platform. It is worth noticing that, despite the approach introduces some approximation in the computation of trace clusters (for the sake of efficiency), the accuracy of the predictions obtained in a real-life application scenario is quite satisfactory. This is likely due to the capability of our probabilistic clustering scheme to compensate such a loss of precision in the clustering phase.

As future work, we plan to integrate more powerful regression methods into our approach, as well as investigate the possibility to exploit sequence-oriented kernel-based methods for both the clustering of the traces and for inducing each cluster's predictor.

References

1. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. Data & Knowledge Engineering 47(2), 237–267 (2003)
2. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450–475 (2011)
3. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
4. Cesario, E., Talia, D.: Distributed data mining patterns and services: an architecture and experiments. Concurrency and Computation: Practice and Experience 24(15), 1751–1774 (2012)
5. Czajkowski, K., et al.: From open grid services infrastructure to ws-resource framework: Refactoring & evolution (2004)
6. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society 39(1), 1–38 (1977)
7. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: Proc. of 20th Int. Conf. on Cooperative Information Systems (CoopIS'12). pp. 287–304 (2012)
8. Folino, F., Guarascio, M., Pontieri, L.: A data-adaptive trace abstraction approach to the prediction of business process performances. In: Proc. of 15th Int. Conf. on Enterprise Information Systems (ICEIS'13). pp. 56–65 (2013)
9. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. In: Proc. of the 2005 IFIP Int. Conf. on Network and Parallel Computing (NPC'05). pp. 2–13 (2005)
10. Moltó, G., Hernández, V.: On demand replication of wsrf-based grid services via cloud computing. In: Proc. of 9th Int. Meeting on High Performance Computing for Computational Science (VecPar'10) (2010)
11. Schonenberg, H., Weber, B., Dongen, B., van der Aalst, W.P.M.: Supporting flexible processes through recommendations based on history. In: Proc. of the 6th International Conference on Business Process Management (BPM'08). pp. 51–66 (2008)
12. Sempolinski, P., Thain, D.: A comparison and critique of eucalyptus, opennebula and nimbus. In: Proc. of the 2nd IEEE Int. Conf. on Cloud Computing Technology and Science (CLOUDCOM '10). pp. 417–426 (2010)
13. Sotomayor, B., Childers, L.: Globus® Toolkit 4: Programming Java Services. Morgan Kaufmann (2006)
14. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, Second Edition. Morgan Kaufmann Publishers Inc. (2005)