

QoS-Driven Management of Business Process Variants in Cloud Based Execution Environments

Rahul Ghosh¹(✉), Aditya Ghose², Aditya Hegde¹, Tridib Mukherjee¹,
and Adrian Mos³

¹ Xerox Research Center India, Bangalore, India

{rahul.ghosh, aditya.hegde, tridib.mukherjee}@xerox.com

² University of Wollongong, Wollongong, NSW 2522, Australia
aditya@uow.edu.au

³ Xerox Research Center Europe, Grenoble, France
adrian.mos@xrce.xerox.com

Abstract. Economy of scale is a key driver behind the Cloud based adoption of a business process. Typically, the management of business process variants focuses on design variants, which permit (ideally small) variations in design (and hence, functionality) for achieving the same (functional) goal, under different functional constraints (such as the compliance obligations that have to be met in different jurisdictions). Little attention has been paid to: (a) variations in process design driven by non-functional considerations (e.g., performance, reliability and cost of operation) and (b) variations in process provisioning in Cloud. This paper seeks to develop means for identifying the correlation between both design and provisioning alternatives and the QoS of business processes deployed in the Cloud. Additionally, we explore the role of the context in determining the performance of a process. We use a set of data mining techniques (specifically decision tree learning, support vector machine and the k-nearest neighbour technique) to mine insights about these correlations. Proposed approaches are evaluated using a synthetic dataset as well as a real dataset.

Keywords: Cloud · Process adaptation · Resource · QoS · Context

1 Introduction

Cloud-based deployment of business processes raises several important challenges. Organizations that deploy business processes in a variety of heterogeneous settings have to manage *process variants*, which modify the original (or *reference*) process design to accommodate the compliance requirements of different jurisdictions or varying client requirements but which still achieve the intended goals of the process [17].

Much of the current literature on the management of business process variants focuses on design variants, which permit (ideally small) variations in design

(and hence, functionality) for achieving the same (functional) goal, under different functional constraints (such as the compliance obligations that have to be met in different jurisdictions). Little attention has been paid to variations in process design driven by non-functional considerations (such as the improvement of processing time or the reduction of cost - factors that we shall henceforth informally describe using the term *QoS*). Little attention has also been paid to variations in process provisioning. Sometimes, the same process design can exhibit very different QoS profiles under different resourcing modes.

Thus, both the design of a process and the resources provided for its execution determine its performance under various QoS measures. The broader research program within which this paper is situated seeks to develop means for identifying and leveraging the correlation between both design and provisioning alternatives and the QoS of business processes deployed in the Cloud. This paper focuses on the provisioning aspect of this question.

This paper additionally explores the role of the context in determining the QoS of a process. We view the context as consisting of exogeneous knowledge about the operating environment of a process that is neither consumed by a process nor generated by it (and hence sitting outside the ambit of what is traditionally described as *process data*). The role of the context in determining the QoS of a process is often mediated by the impact the context has on the resources allocated to a process. For instance, we might find that a process with a significant proportion of human-mediated steps (such as customer complaint handling) leads to positive outcomes (as evidenced by a higher proportion of satisfied customers) when executed earlier in the day. The underlying contextual phenomenon that manifests in this behaviour could be the higher efficiency of the customer contact centre workers at the start of the working day. Sometimes, the context influences process performance in other ways. There is some anecdotal evidence that insurance claims processes progress to completion faster during periods when the economic climate is positive (and take longer to complete when this is not the case). Of particular interest for the purposes of this study is the *load* on hardware and software resources that provision a business process deployed in the Cloud.

There are a number of use cases for the techniques we develop in this paper. Given a description of a context, we are interested in identifying (or *prescribing*) the alternate set of resources with which to provision a process in order to achieve a set of thresholds on the applicable QoS factors e.g., cost, processing time, reliability etc. Given a context, and the set of resources being allocated to a process, we are interested in *predicting* its QoS. We are interested in answering these questions both at the start of the execution of a process (where a process design provides approximate guidance on what tasks are to be executed - the specific task sequence being determined at run-time) and during the execution of a process (where we have, in addition to a process design, a sequence of tasks already executed).

These questions are best answered by using a causal theory that relates aspects of the context, the process and the resources to the QoS of a process. Such

a theory does not exist at this time (although preliminary attempts have been made to use queueing theory to address part of the problem [8]). We therefore propose to adopt a data-driven approach, by using a history of past process executions as a proxy for this theory. We make the (realistic) assumption that we have access to an execution history that records for each executed process instance: (1) the context (consisting of the load, and potentially other contextual factors), (2) the process design, (3) the process instance, (4) the resource set deployed for that instance, and (5) the QoS (in terms of values for a set of QoS factors).

We use a set of data mining techniques (specifically decision tree, support vector machines and the k-nearest neighbour technique) to mine insights about these correlations. The overall strategy is to mine correlations between the context, the process, the resource set and the QoS, and use these correlations to support the *prescriptive* and *predictive analytics* use cases discussed above. We also leverage an ontology of resource types to describe the resource sets with which process instances are provisioned. In addition to providing a vocabulary, this *resource ontology* also forms the basis for a *resource set ontology*. We leverage the latter to obtain insights about superclasses (of classes of resource sets that are well-represented in the available datasets) which can then support prescriptive, predictive and diagnostic analytics for other classes of resource sets (potentially under-represented in the available data).

We evaluate our proposal in the context of two datasets: a synthetic dataset and real-life dataset drawn from the Business Process Intelligence (BPI) Challenge [2].

The literature directly related to our proposal is quite sparse. A substantial body of work addresses business process flexibility, but the focus of these approaches is primarily on the generation of design variants (as opposed to the generation of variations in resourcing that we focus on). Some recent work addresses the role of the context in process execution, but not in the kinds of analytics we have outlined above.

2 The Proposed Framework

We present the conceptual basis for our proposal in this section. Our overall strategy is to view an execution history as a proxy, or an alternative, for a causal theory that might inform resource allocation decisions in the given context. We might view the causal theory of interest providing the functions described below. These functions will refer to a *process* P , a *resource set* R , a context C and the QoS of a process, denoted by QoS . The first function of interest is the following:

$$f_{QoS} : C \times P \times R \mapsto QoS \quad (1)$$

This function helps us predict the QoS of a process, given a characterization of the context, the process and the resources allocated to the process. In much of our evaluation, we will focus on the *load* in Cloud-based deployment as the context C , but it is useful to note that our overall framework is far more general,

and could be leveraged in settings with a richer description of the context. We encode a characterization of the process in P . In parts of our evaluation, we use a fairly coarse-grained characterization of a process (in terms of number of tasks, in settings where it is reasonable to assume that the resource requirements of the tasks in a process are fairly uniform). We also consider a more finer-grained characterization in other parts of our evaluation, where we take cognisance of the identities (i.e., task types) of the tasks executed in a process (this characterization is useful in settings where tasks have heterogeneous resource requirements). More generally, we could characterize a process by referring to the actual process instance (which describes the sequence, and not just the set of tasks executed). We could make the characterization even richer by referring to the process design. We use a *resource ontology* to help characterize the resource sets that are used to provision each process instance.

Another function of interest is:

$$f_R : C \times P \times QoS \mapsto R \quad (2)$$

This function helps us predict the resource sets that must be provided to a process instance, in a given context C , in order to ensure that certain thresholds on the QoS, given by QoS are met (note that we have over-used the term QoS , which was used to denote actual QoS measures in the case of function f_{QoS}). C and P would be characterized as per the discussion above.

Our focus in this paper is to mine execution histories to obtain approximations of these two functions. We assume a general schema for the execution history, given by the tuple $\langle C, P, R, QoS \rangle$ (with the provision that C , P , R and QoS might be differently instantiated, as discussed above). Our overall strategy is to use classification techniques such as decision tree learning and support vector machine (SVM) to learn an approximation of function f_{QoS} and to use the k-nearest neighbour technique to learn an approximation of function f_R (which would support the prescriptive analytics discussed in the previous section). The remainder of this paper illustrates the feasibility of our approach via experimental evaluation using both synthetic and real-life datasets.

3 Evaluation

We evaluate the proposed approach using two types of datasets: (a) simulated process execution logs and (b) real process execution logs. Using both simulated and real execution logs, we answer interesting questions around the resource allocation and QoS.

3.1 Insights from Simulated Process Execution Logs

Simulated process logs are generated in the domain of Infrastructure-as-a-Service (IaaS) Cloud and contains a rich variety of Cloud resources. We first describe the resource ontology considered in this context.

Resource Ontology. As shown in Fig. 1, infrastructure resources in the Cloud can be divided into three classes: physical machines (PM), virtual machine (VM), operating system containers (OC). For each of the above classes, there are three sub-classes - large, medium and small, which denote the amount of computing capacity given to a resource instance. For example, a large PM can have 32 CPU cores, 16 GB of memory and 1 TB of disk, while a small PM can have 4 CPU cores, 4 GB of memory and 256 GB disk. Further, using the resource types (r_t) shown in the *primary ontology* R_p in Fig. 1, we create a *derived ontology* R_d . A derived ontology consists of a number of *resource sets* R_s . Let c_t denotes the cardinality of resource type (r_t). The resource set can then be given as:

$$R_s = \{(x, y) : x = r_t, y = c_t\} \quad (3)$$

Examples of R_s are: (i) {3 large PMs}, (ii) {1 large PM, 2 medium VMs}, (iii) {2 large VMs, 1 large OC} and so on. For tractability of the dataset, we vary the value of c_t from 1 to 10. A singular element of R_s is called an *atomic resource unit* r_a . Examples of r_a are: 1 large PM, 1 small VM, 1 medium OC and so on. The universal resource set R_u is the power set of R_s .

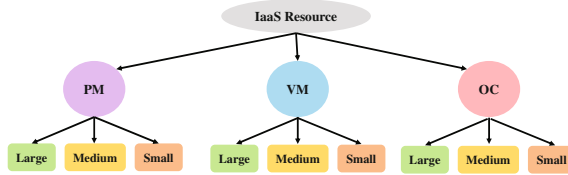


Fig. 1. Primary resource ontology for IaaS Cloud.

Generation of Synthetic Logs. Figure 2 shows the conceptual process of synthetic log generation. For a given sequential Cloud based BP, we randomly generate n tasks which constitute a BP instance. For each task, we randomly generate the total number of resources needed (r_{total}). The values of n and r_{total} are generated by drawing samples from uniform distributions $U(3, 10)$ and $U(18, 45)$ respectively. Once the value of r_{total} is generated, we draw a resource set R_s from R_u such that the cardinality of R_s is same as r_{total} . For instance, when the value r_{total} is 3, possible resource sets could be: (a) 3 large PMs, (b) 3 small OCs, (c) 1 PM, 1 VM, 1 OC etc.

The next step is to generate the task level QoS. To keep things realistic, we adopt a statistical model to compute QoS from resources as described in [10]. Given an atomic resource unit, a value of load (λ), it can be shown that there is a linear relationship between the throughput and the atomic resource unit. Response time can then be computed as the inverse of throughput. Note that, while we adopt a linear model for QoS computation, we randomly vary the values of slope and y-axis intercept for an atomic resource unit. Let T_i and S_i denote

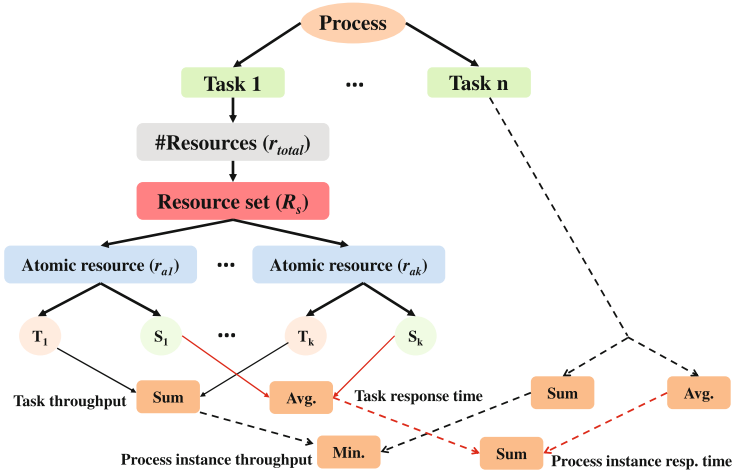


Fig. 2. Conceptual process showing the generation of synthetic dataset.

the throughput and response time of i -th atomic resource unit. The throughput of a task is then computed as the sum total throughput of each atomic resource unit. This is intuitive as we put more resources onto a task, overall task throughput increases. The response time of a task is computed as the average response time of individual atomic resource units. Next, we aggregate the task level data to compute the process instance level QoS. Since we assume that the tasks are sequential in nature, overall throughput is determined by the *slowest* (bottleneck) task. Thus, process instance level throughput is computed by the minimum of individual task throughputs. Overall response time is computed as the sum of individual task level response times. In total, 25000 process instances are simulated and the generated data is used to provide the following insights. We use the `scikit-learn` [3] package in Python for all the data mining approaches used in this paper.

Classification Based Approaches for QoS Prediction. We use two classification approaches: (1) decision tree based learning and (2) SVM based learning for answering the two questions related to QoS prediction.

Q1. Given a QoS threshold and a process that is about to start, can we predict if the process will meet its QoS?

Considering response time as the QoS, we first replace the QoS value in our dataset to binary variable *isQoSsatisfied*. For a given process instance, if the value of QoS is less than or equal to the value of QoS threshold, we set the value of *isQoSsatisfied* as *True*, otherwise, we set it to *False*. Thus, given *isQoSsatisfied* as the target (or dependent) variable, the dataset contains the following independent variables (predictor): load (λ), #tasks (n), #total resources (r_{total}), #small PMs, #medium PMs, #large PMs, #small VMs, #medium VMs, #large VMs, #small OCs, #medium OCs, #large OCs. Table 1 shows the feature importance

Table 1. Predictor importance in synthetic data when QoS is predicted for a process that is about to start.

Predictor	Importance (decision tree)	Importance (SVM)
Load (λ)	0.591	4.236
#tasks (n)	0.226	-3.132
#resources (r_{total})	0.013	0.002
#small PMs (PM-small)	0.010	0.345
#medium PMs (PM-medium)	0.010	0.392
#large PMs (PM-large)	0.013	0.441
#small VMs (VM-small)	0.073	-1.256
#medium VMs (VM-medium)	0.027	-0.429
#large VMs (VM-large)	0.013	-0.099
#small OCs (OC-small)	0.007	0.107
#medium OCs (OC-medium)	0.006	0.217
#large OCs (OC-large)	0.009	0.288

for both decision tree and SVM based approaches. The weights of a linear SVM can have both positive and negative values because the result of a linear SVM is a hyperplane that separates the classes as best as possible. The weights form the coordinates of a vector which is orthogonal to the hyperplane. The values of the weights/coefficients relative to the other ones give an indication of how important the feature was for the separation.

The accuracy of decision tree and SVM based approach is 94.52% and 97.32% respectively.

Q2. Given a QoS threshold and a partially executed process log, can we predict if the QoS will be met at the end of process execution?

For answering this question we have a similar data set as used for *Q1*. Instead of using aggregated resource set, we have introduced task level resource sets. So the independent variables are: load (λ), #tasks (n), #total resources (r_{total}), resources required for a task, i.e., #small PMs, #medium PMs, #large PMs, #small VMs, #medium VMs, #large VMs, #small OCs, #medium OCs, #large OCs and task response time. To construct the target (dependent) variable, we transform the process level QoS (i.e., process response time) to a binary variable *isQoSsatisfied* indicating a response time threshold. For an instance of the dataset, *isQoSsatisfied* is set to *True* if the underlying process met its QoS else it is set to *False*. The training dataset consists of the history of complete executions of process instance along with the task related information. The test dataset consists of partial executions of the process instance, i.e., a set of tasks. The developed model (decision tree or SVM) predicts if the underlying process can meet its QoS or not when the tasks are submitted to the model. Since the process model consists of sequential tasks, the test tasks are fed to the model one after another. If the prediction is *False*, subsequent tasks for the test process

Table 2. Predictor importance for synthetic data when QoS is predicted for a partially executed process.

Feature	Importance (decision tree)	Importance (SVM)
Load (λ)	0.679	1.649
#tasks (n)	0.183	-1.196
#resources (r_{total})	0.041	0.019
#small PMs for a task	0.005	0.016
#medium PMs for a task	0.004	0.021
#large PMs for a task	0.003	0.007
#small VMs for a task	0.002	-0.046
#medium VMs for a task	0.004	-0.0.013
#large VMs for a task	0.004	0.002
#small OCs for a task	0.004	0.006
#medium OCs for a task	0.005	0.008
#large OCs for a task	0.005	0.016
Task response time	0.056	-0.481

are no longer fed to the model. Table 2 shows the predictor performance for both decision tree and SVM based approaches. The accuracy of decision tree and SVM based approach is 88.44 % and 92.46 % respectively.

Clustering Based Approach for Resource Prediction. Using kNN, we answer the three questions related to resource prediction. For all the questions, we replace the QoS value in our dataset to binary variable *isQoSsatisfied*. The value of *isQoSsatisfied* is set to be 1 in case the QoS is met w.r.t. a given threshold, otherwise it is set to 0.

Q3. Given a process instance that met its QoS, can we predict an alternate resource set such that it meets the QoS?

There are several motivations behind such a question including reducing the cost of resource procurement [9], lack of availability of certain types of resources etc. Using response time as the QoS, we leverage the full process logs from the historical data to answer this question. The attributes of the dataset are in the following order: (load, #tasks, #total resources, #small PMs, #medium PMs, #large PMs, #small OCs, #medium OCs, #large OCs, #small VMs, #medium VMs, #large VMs, *isQoSsatisfied*). We use the given process instance as a test data, where *isQoSsatisfied* is set to 1 (as the process instance met the QoS). From historical data, we then determine the process instances that are closest to the test process and for which *isQoSsatisfied* is set to 1. For example, consider the test process instance z :

$$z = (537, 4, 11, 1, 1, 0, 2, 0, 2, 1, 3, 1, 1) \quad (4)$$

This instance has 4 tasks which got executed on 2 PMs (1 small, 1 medium), 4 OCs (2 small, 2 large), and 5 VMs (1 small, 3 medium, 1 large). Using kNN approach, the top 2 closest process instances which meet the QoS are given by:

$$z_1 = (535, 3, 10, 1, 1, 0, 2, 0, 1, 2, 3, 0, 1), z_2 = (535, 4, 12, 0, 3, 0, 1, 0, 2, 1, 3, 2, 1)$$

Without compromising the QoS target, these resource sets of these alternate process instances can then be opportunistically used by a Cloud provider.

We also study another variant of this question when different types of tasks are considered within a process. In this case, the dataset has an additional attribute: #tasks for a given type. We re-generate our synthetic dataset considering 5 types of tasks. The attributes are then given by: (load, #tasks, #tasks for type 1, #tasks for type 2, #tasks for type 3, #tasks for type 4, #tasks for type 5, #total resources, #small PMs, #medium PMs, #large PMs, #small OCs, #medium OCs, #large OCs, #small VMs, #medium VMs, #large VMs, *isQoSsatisfied*). For example, consider the test process instance y :

$$y = (986, 12, 3, 3, 3, 1, 2, 34, 1, 2, 7, 4, 4, 2, 3, 7, 4, 1) \quad (5)$$

This instance has 3 tasks for type 1, 2, and 3 each, 1 task for type 4, 2 tasks for type 5. Using kNN approach, the top 2 closest process instances which meet the QoS are given by:

$$y_1 = (989, 11, 3, 2, 1, 3, 2, 34, 1, 3, 5, 3, 6, 4, 5, 2, 5, 1), \\ y_2 = (986, 10, 3, 2, 1, 1, 3, 30, 4, 3, 6, 1, 2, 4, 2, 4, 4, 1)$$

Q4. Given a process instance that did not meet its QoS, can we predict alternate resource set such that it meets the QoS?

The analysis carried out for this question is similar to that of Q3. The main difference is that, when a process instance is considered as a test data, *isQoSsatisfied* is set to 0 (as the process instance did not meet QoS). From historical data, we then determine the process instances that are closest to the test process and for which *isQoSsatisfied* is set to 1. For example, consider the test process instance t :

$$t = (343, 6, 21, 3, 3, 1, 2, 3, 2, 1, 4, 2, 0) \quad (6)$$

This instance has 6 tasks which got executed on 7 PMs (3 small, 3 medium, 1 large), 7 OCs (2 small, 3 medium, 2 large), and 7 VMs (1 small, 4 medium, 2 large). Using kNN approach, the top 2 closest process instances which meet the QoS are given by:

$$t_1 = (348, 5, 17, 2, 1, 2, 1, 5, 2, 0, 2, 2, 1), t_2 = (342, 5, 15, 1, 1, 4, 2, 3, 4, 0, 0, 0, 1)$$

The resource allocation in any of these neighboring process instances can be used as an alternate resource set for the test process instance to meet the QoS.

When different types of tasks are involved within a process instance we run an analysis similar to the variant described in Q3. Consider the test process instance w :

$$w = (397, 13, 3, 3, 2, 2, 3, 40, 7, 4, 3, 1, 1, 7, 6, 4, 7, 0) \quad (7)$$

This instance missed its QoS target and has 3 tasks for type 1, 2, and 5 each, 2 tasks for type 3 and 4 each. Using kNN approach, the top 2 closest process instances which meet the QoS are given by:

$$w_1 = (399, 11, 2, 2, 1, 3, 3, 37, 5, 3, 8, 3, 3, 3, 3, 2, 7, 1),$$

$$w_2 = (396, 11, 2, 2, 2, 2, 3, 34, 3, 3, 7, 4, 2, 5, 1, 6, 3, 1)$$

Q5. Given a partially executed process instance, can we predict an alternate resource set such that it can meet the QoS at the end of entire process execution?

To answer this question, instead of using aggregated resource set, we have introduced task level resource sets. So the features are: load, #tasks, #total resources (r_{total}), resources required for a task, i.e., #small PMs, #medium PMs, #large PMs, #small OCs, #medium OCs, #large OCs, #small VMs, #medium VMs, #large VMs, task response time, value of *isQoSsatisfied*. In this case, a sequence of tasks belonging to the same process are considered as the test data. The goal is to determine the *similar* tasks with alternate resource allocations that can meet the QoS. Consider the following two test tasks:

$$v_1 = (259, 6, 18, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0.504, 0),$$

$$v_2 = (259, 6, 18, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0.300, 0)$$

Both tasks belong to a process instance for which load is 259 requests/sec, #tasks is 6, #total resources is 18 and the QoS is not met (i.e., *isQoSsatisfied* = 0). Task v_1 requires 1 small and medium OCs and has a response time of 0.504 ms. Task v_2 requires 1 medium PM and 1 large OC and has a response time of 0.300 ms. Using kNN approach, the top 2 closest process instances v_1 which meet the QoS are given by:

$$v_{11} = (261, 6, 17, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0.357, 1),$$

$$v_{12} = (261, 6, 17, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0.555, 1)$$

Similarly, the top 2 closest process instances v_2 which meet the QoS are given by:

$$v_{21} = (259, 6, 18, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0.300, 1),$$

$$v_{22} = (261, 6, 17, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0.293, 1)$$

3.2 Insights from Real Process Execution Logs

The real dataset comes from Dutch financial institute provided as part of BPI challenge 2012 [2]. The event logs are for an application process for a personal loan or overdraft within a global financial organization. The log contains 2,62,200 events in 13,087 cases. Every process instance starts from a specific event and ends up in any one of the 13 different end events. Based on this observation we divide the process instances into 13 different instance types (*itype*). We calculate the response time of the process instance as the time difference between the end event and the start event. We define a threshold for the response time. The target

(dependent) variable *isQoSsatisfied* is created using the defined threshold. If the response time is below the threshold we set *isQoSsatisfied* to *True* otherwise it is set to *False*. Unlike the synthetic data, this dataset does not have the richness around resource information. Individual resources are identified by a unique resource ID. For every resource ID we mark the instance with *True* if the corresponding resource participates in the instance. There are 68 unique resources who work on this process across various instances. We extract the following predictors from this event log: loan amount, instance type (*itype*), #tasks (*n*), #total resources (*r_{total}*), resource ID.

Q1. Given a QoS threshold and a process that is about to start, can we predict if the process will meet its QoS?

Following a similar approach taken for the synthetic logs, we use decision tree and SVM to answer this question. Table 3 shows the predictor performance for both decision tree and SVM based approaches. As the predictor space for resource ids is large we show the importance of only a subset of resource ids. The accuracy of decision tree and SVM based approach is 94.32 % and 96.32 % respectively.

Table 3. Predictor importance for real data when QoS is predicted for a process that is about to start.

Predictor	Importance (decision tree)	Importance (SVM)
Loan amount	0.025	0.045
Instance type	0.027	-0.621
#tasks	0.866	-1.317
#resources	0.022	-0.194
Resource ID 11180	0.002	-0.092
Resource ID 11181	0.001	-0.048
Resource ID 10929	0.000	-0.022

Q2. Given a process instance that met its QoS, can we predict an alternate resource set such that it meets the QoS?

The analysis carried out for this question is similar to that of Q3 of Sect. 3.1. From historical data, we determine the process instances that are closest to the test process and have met their QoS. For example, consider the test process instance *t* (the semicolon separated items are resource Ids):

$$t = (5862, 3, 6, 2, 112; 11019, 1) \quad (8)$$

The loan amount and instance type for process instance *t* are 5862 and 3 respectively. In total, 6 tasks are performed by 2 resources with ids 112 and 11339. Using kNN approach, the top 2 closest process instances which meet the QoS are given by:

$$t_1 = (5867, 4, 16, 4, 112; 11119; 10982; 11202, 1), t_2 = (5900, 3, 6, 2, 112; 10881, 1)$$

Q3. Given a process instance that did not meet its QoS, can we predict alternate resource set such that it meets the QoS?

Here, when a process instance is considered as a test data, *isQoSsatisfied* is set to 0. From historical data, we then determine the process instances that are closest to the test process and for which *isQoSsatisfied* is set to 1. For example, consider the test process instance t :

$$t = (5000, 2, 23, 4, 10609; 10899; 112; 11201, 0) \quad (9)$$

This type 2 instance has the loan amount of 5000 and has 23 tasks which are performed by 4 resources with ids 10609, 10899, 112 and 11201. Using kNN approach, the top 2 closest process instances which meet the QoS are given by

$$t_1 = (5000, 2, 22, 2, 112; 10629, 1), t_2 = (5000, 2, 25, 2, 112; 10629, 1)$$

4 Related Research

The field of software performance modelling and prediction is vast. A comprehensive survey of modelling approaches for performance prediction is presented in [4]. Important contributions have been presented in [1, 18, 20, 21] reporting significant results in the improvement of the software development process, specifically the use of Software Performance Engineering methods aided by related tools such as SPE-ED [18]. The techniques and the supporting tools require developers to create software and/or system models of the application under development. These models must have performance parameters such as I/O utilisation, CPU cycles or network characteristics, specified by the developers in order for the performance predictions to generate meaningful results. It has been proved that such techniques and tools like SPE-ED help in achieving performance goals and reducing performance related risks for general object-oriented systems and even for distributed systems. However, BPM environments exhibit an inherent complexity, which analysts find hard if not impossible to quantify even in simple models. Complex contextual and provisioning details such as resource allocation, design options, underlying engine requirements, virtualisation and many others contribute to an improved and at the same time highly unpredictable run-time environment. It is therefore impractical for the various stakeholders involved in building and maintaining process based applications to create performance models where they manually specify the mapping of methods to processes or instances to processors, I/O characteristics or CPU utilisation. A related approach to modelling systems in UML is presented in [11]. OAT is a tool that implements a framework for performance modelling of distributed systems using UML. It consists of a method for decomposition of models and performance modelling techniques. UML models, created at different development stages can be mapped to queuing networks and solved to predict application performance. UML diagrams, in particular sequence diagrams, can be seen as similar to process description in BPMN so the approach could provide starting points for process

performance analysis and prediction. However, users must create and maintain the models and augment them with performance annotations leading to a similar disadvantage with that of the SPE-ED [18] approach. In addition, it is not clear how this approach can be used for large systems, as it does not address issues such as large-scale model management. Since BPM environments embed complex server software they share similarities with traditional middleware systems. Predicting the performance of middleware-based systems has been approached in the past. Among the most common techniques are Petri-Nets [7] and Layered Queuing Network (LQN) [16,20] models. Similarly to the generic software performance prediction approaches, such work may not scale particularly well to the complexities of process environments with the rich contextual and provisioning variability.

Closer to the BPM space there has been a lot of work in monitoring processes for performance understanding and analysis, while not particularly targeting QoS correlation with provisioning and design. In [6], authors present potential openings driven by associating semantics with traditional BPM, which could potentially be used as further input in the context descriptions; as done in the approach presented here. Related work that leverages semantic descriptions to boost process understanding and monitoring includes [15] and the associated COBRA terminology is presented in [14]. Other monitoring and analysis work targeting complex and large scale process and service environments is presented in [13]. These approaches, while comprehensive in the analysis scope in particular with regard to service dependencies and process structure, do not particularly target Cloud-based provisioning aspects, nor do they propose QoS oriented performance prediction.

Specifically, in process performance prediction, existing work [19] has looked at using forecasting techniques to generate dynamic predictive models. The contribution is certainly useful but does not take into account the correlation between design, complex contextual data, provisioning and performance in a unified way that would allow the understanding of the reasons for performance results, in contrast to the work presented in this paper. In contrast, the authors in [5] propose a mechanism based on LQN to predict performance of processes at design time based on an extended BPMN profile called PyBPMN. Such predictions can be used to quickly simulate overall performance based on expected characteristics, but it does not take into account runtime information from previous executions nor does it combine them with provisioning and contextual data.

While not Cloud-specific, previous work in correlative process monitoring presented in [12] uses domain-specific information to aggregate non-functional properties extracted from process execution at various levels including BPM engine and enterprise service bus platforms. This provides an important capability for better understanding previous executions in terms that are easier to understand by business stakeholders. As such, this work could provide an additional layer of interpretation when correlated with the predictive work presented in this paper. This would allow specification of QoS requirements and matching of predictive results using a consistent terminology related to the business domain, rendering such data easier to use in practice.

5 Conclusions and Future Work

In this paper, we develop an approach for identifying the correlation between both design and provisioning alternatives and the QoS of business processes deployed in the Cloud. Specifically, we view an execution history as a proxy for a causal theory that might inform resource allocation decisions in the given context of a Cloud based business process. Data mining techniques, specifically, decision tree, SVM and kNN are used to provide an approximation of the underlying causal theory that helps predicting the QoS or resource requirements in Cloud. Classification techniques such as decision tree learning and SVM are used to approximate the function that leads to QoS prediction. For both synthetic as well as real dataset, one interesting insight that comes out from our analysis is that, among the classification based approaches, SVM performs better than decision tree in terms of accuracy of prediction. While this paper focuses on QoS and resource prediction problems, in future, we plan on developing similar approach for predicting alternate process variants as well. We believe that similar approximation model for predicting process variants can significantly improve the design of business processes from non-functional aspect.

References

1. Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison Wesley Longman Publishing Co., Inc., Redwood City (2002)
2. BPI Challenge 2012 (2016). <https://www.win.tue.nl/bpi/doku.php?id=2012:challenge>
3. Scikit-learn: Machine Learning in Python (2016). <http://scikit-learn.org/stable/index.html>
4. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. *IEEE Trans. Softw. Eng.* **30**(5), 295–310 (2004). <http://dx.doi.org/10.1109/TSE.2004.9>
5. Bocciarelli, P., D’Ambrogio, A.: A model-driven method for enacting the design-time QoS analysis of business processes. *Softw. Syst. Model.* **13**(2), 573–598 (2014)
6. de Medeiros, A.K.A., Pedrinaci, C., van der Aalst, W.M.P., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An outlook on semantic business process mining and monitoring. In: Meersman, R., Tari, Z. (eds.) *OTM-WS 2007, Part II*. LNCS, vol. 4806, pp. 1244–1255. Springer, Heidelberg (2007)
7. Dimitrov, E., Schmietendorf, A., Dumhe, R.: Uml-based performance engineering possibilities and techniques. *IEEE Softw.* **19**(1), 74–83 (2002)
8. Ghosh, R.: Scalable stochastic models for cloud services. Ph.D. thesis, Duke University (2012)
9. Ghosh, R., Longo, F., Xia, R., Naik, V., Trivedi, K.: Stochastic model driven capacity planning for an infrastructure-as-a-service cloud. *IEEE Trans. Serv. Comput.* **7**(4), 667–680 (2014)
10. Jung, G., Sharma, N., Goetz, F., Mukherjee, T.: Cloud capability estimation and recommendation in black-box environments using benchmark-based approximation. In: *IEEE CLOUD* (2013)

11. Kähkipuro, P.: UML based performance modeling framework for object-oriented distributed systems. In: France, R.B. (ed.) UML 1999. LNCS, vol. 1723, pp. 356–371. Springer, Heidelberg (1999)
12. Mos, A.: Domain specific monitoring of business processes using concept probes. In: Toumani, F., et al. (eds.) ICSOC 2014 Workshops. LNCS, vol. 8954, pp. 213–224. Springer, Berlin (2015)
13. Mos, A., Pedrinaci, C., Rey, G.A., Gomez, J.M., Liu, D., Vaudaux-Ruth, G., Quaireau, S.: Multi-level monitoring and analysis of web-scale service based applications. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 269–282. Springer, Heidelberg (2010)
14. Pedrinaci, C., Domingue, J., Alves de Medeiros, A.K.: A core ontology for business process analysis. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 49–64. Springer, Heidelberg (2008)
15. Pedrinaci, C., Lambert, D., Wetzstein, B., Van Lessen, T., Cekov, L., Dimitrov, M.: Sentinel: a semantic business process monitoring tool. In: Proceedings of 1st International Workshop on Ontology-Supported Business Intelligence, p. 1. ACM (2008)
16. Petriu, D.C., Shen, H.: Applying the UML performance profile: graph grammar-based derivation of LQN models from UML specifications. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 159–177. Springer, Heidelberg (2002)
17. Ponnalagu, K., Ghose, A.K., Narendra, N.C., Dam, H.: Cloud capability estimation and recommendation in black-box environments using benchmark-based approximation. In: BPM (2015)
18. Smith, C.U., Williams, L.G.: Performance and scalability of distributed software architectures: an SPE approach. *Parallel Distrib. Comput. Pract.* **3**(4), 74106–80700 (2002)
19. Solomon, A., Litoiu, M.: Business process performance prediction ONA tracked simulation model. In: Proceedings of 3rd International Workshop on Principles of Engineering Service-Oriented Systems, PESOS 2011, pp. 50–56. ACM, New York (2011). <http://doi.acm.org/10.1145/1985394.1985402>
20. Williams, L.G., Smith, C.U.: Performance evaluation of software architectures. In: Proceedings of 1st International Workshop on Software and Performance, WOSP 1998, pp. 164–177. ACM, New York (1998). <http://doi.acm.org/10.1145/287318.287353>
21. Woodside, M., Petriu, D.C., Merseguer, J., Petriu, D.B., Alhaj, M.: Transformation challenges: from software models to performance models. *Softw. Syst. Model.* **13**(4), 1529–1552 (2014). <http://dx.doi.org/10.1007/s10270-013-0385-x>