2013 IEEE 6th International Conference on Service-Oriented Computing and Applications

Workflow Scheduling and Resource Allocation for Cloud-based Execution of Elastic Processes

Philipp Hoenisch, Stefan Schulte, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology, Austria
Email: {p.hoenisch, s.schulte, dustdar}@infosys.tuwien.ac.at

***Abstract*—Today's extensive business process landscapes make it necessary to handle the execution of a large number of workflows. Especially if workflow steps require the invocation of resource-intensive applications or a large number of applications needs to be carried out concurrently, process owners may have to allocate extensive computational resources, leading to high fixed costs. Instead, process owners could make use of Cloud-based computational resources, dynamically leasing and releasing resources on demand, which could lead to lower costs.**

In the work at hand, we propose a resource-efficient workflow scheduling algorithm for business processes and Cloud-based computational resources. Through the integration into the Vienna Platform for Elastic Processes and an evaluation, we show the practical applicability and the benefits of our contributions. Specifically, we find that our approach reduces the resource demand if compared with an ad hoc approach.

***Index Terms*—Workflow Scheduling, Elastic Processes, Cloud Computing, Business Process Execution**

I. Introduction

Business processes are composed from human and software services in order to represent and execute a particular business logic [1], [2]. In recent years, Business Process Management (BPM) has become a matter of great importance in many different industrial sectors, e.g., the financial industry as means to carry out trade settlement or execution control [3], or in the energy domain as means to carry out essential decision processes [4], [5]. Parts of business processes which can be carried out using computational resources are also known as workflows [6]. One of the most important aspects of BPM is the automatic execution of workflows and a large number of different approaches for modeling and executing such workflows have been presented in the past [7], [8].

Efficient and effective resource allocation is a key issue in workflow executions [9], and a corresponding Business Process Management System (BPMS) has to be able to administrate, schedule and execute hundreds of workflows, which are each made up from several workflow steps. The BPMS is also responsible to assign needed computational resources to the single services representing the workflow steps [10]. Based on the functional diversity of these services, the requirements in computational resources do vary. Many of these services may be resource-intensive, especially if a large amount of data has to be processed in a short time [5]. Furthermore, some workflows need to be carried out immediately, other may have to be done regularly, and others may have to be done until a specific deadline. Naturally, the computational resources needed to execute workflows are subject to temporal variations – during peak times, the needed computational resources will be much higher than during non-peak times [11]. On the one hand, the permanent provision of fixed computing capacities that could handle peak loads in a resource-intensive process landscape would lead to very high costs based on the *overprovisioning* of resources in off-peak times, since resources will not be utilized most of the time. On the other hand, workflows may not be carried out at all or with unacceptable Quality of Service (QoS), if there is a shortage (*underprovisioning*) of resources.

With the advent of Cloud technologies, companies may apply a cost-efficient alternative that enables the use of computational resources in an on-demand, utility-like fashion while preventing the risk of over- and underprovisioning [12]. Especially for business process landscapes, the usage of Cloud-based computational resources is promising, as this approach offers *measured service*, ensures sufficient performance to serve the requested workflows, as resources can be acquired *on-demand* and in a *self-service* way, and the underlying Cloud infrastructure provides the possibility to scale up and down by adding or removing resources (*rapid elasticity*) [13].

To make use of Cloud resources, a BPMS has several responsibilities: First, it has to lease and release Cloud-based computational resources during system runtime. Second, it has to create a scheduling plan for the workflow executions, i.e., a detailed plan which defines when a particular workflow and its services have to be carried out. This plan needs to take into account the individual deadlines of all workflows in a process landscape. Third, it has to make sure that the scheduling plan is carried out and allocate according resources to the single services and service instances. Fourth, it needs to conduct countermeasures in the case of service or resource failures.

To the best of our knowledge, little research has been done so far in the field of business process execution using Cloud resources in order to realize *elastic processes* [11], [14]. In our former work [15], [16], we presented the *Vienna Platform for Elastic Processes (ViePEP)*, a BPMS capable to execute workflows in the Cloud. In the work at hand, we extend ViePEP by dynamic workflow scheduling and resource allocation algorithms which make use of Cloud-based resources in an optimal way.

The remainder of this paper is organized as follows: In

978-1-4799-2701-2/13 $31.00 © 2013 IEEE
DOI 10.1109/SOCA.2013.44

Section II, we briefly present our overall approach as well as ViePEP. Afterwards, we introduce the used scheduling (Section III) and reasoning algorithms (Section IV). In Section V, we evaluate the algorithms. In Section VI, we provide an overview of the related work. The paper closes with a summary and an outlook on our future work.

II. The Vienna Platform for Elastic Processes

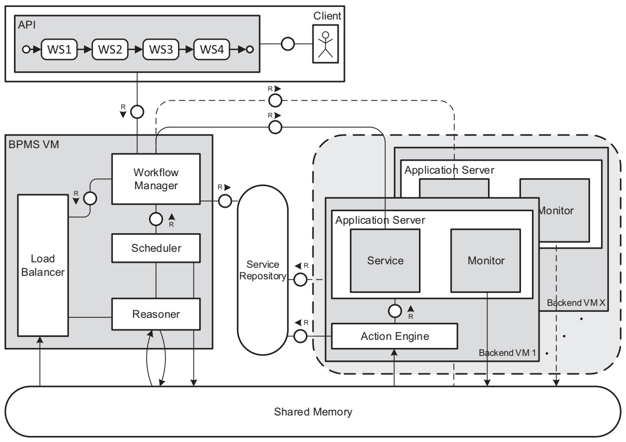


Figure 1: ViePEP – Architecture

ViePEP provides mechanisms and architectural components that are beyond those of traditional BPMS approaches [11], as it combines the functionalities of a Cloud resource management system and a BPMS. The platform has to manage a potentially highly volatile business process landscape, as workflow requests permanently arrive, leading to a changing demand of required resources. This makes it necessary to continuously align and optimize the system landscape, which is made up (i) from the business process landscape and (ii) the leased Cloud resources. ViePEP operates on the Platform as a Service (PaaS) level [17], i.e., is able to lease and release computational resources in terms of Virtual Machines (VMs) from a Cloud provider and deploy services on them. For this, ViePEP is able to schedule workflows and single workflow steps based on predefined rules, and allocate Cloud-based resources to execute them. ViePEP provides the capabilities to estimate the resource demand for carrying out single services and is able to lease and release Cloud resources based on this information.

As depicted in Figure 1, ViePEP consists out of five main top level entities: If a workflow is requested by the *Client*, it is forwarded to the *BPMS VM*, which hosts the core functionalities of ViePEP in terms of a BPMS and Cloud resource control solution. Most importantly, it includes the *Scheduler* and *Reasoner*, which are responsible for creating a detailed scheduling plan according to the workflow deadlines, and lease or release the required Cloud-based computational resources. For a detailed descriptions, please refer to Sections III and IV.

The BPMS VM controls a number of *Backend VMs*, each hosts a particular service on an *Application Server*. The *Action Engine* is able to deploy and undeploy services if requested by the Reasoner. For this, the Reasoner may send a request to *start*, *duplicate*, or *terminate* a Backend VM. Furthermore, it is possible *move* a service instance from one Backend VM to another, or to *exchange* the hosted service by another one.

Last but not least, the *Shared Memory* and *Service Repository* provide data storage capabilities for services and monitoring data. For a more detailed discussion of ViePEP, we refer to our previous work [15], [16].

So far, the workflow scheduling and reasoning algorithms of ViePEP are able to make sure that workflow deadlines are met. However, there has been no optimization of resource utilization. Within the next two sections, we present a more sophisticated scheduling and reasoning approach which is able to make sure that workflow scheduling is done in a resource-efficient way.

III. Scheduling

A. Problem Statement

A scheduling algorithm for elastic processes is responsible for finding a workflow execution plan which makes sure that all workflows are carried out under the given constraints. These constraints could be defined in a Service Level Agreement (SLA). Based on this scheduling plan, the Reasoner is able to allocate, lease, and release Cloud-based computational resources.

Scheduling has to be done continuously for an unknown duration of time, across a system landscape including the business process landscape as well as Cloud resources, so that:

- All SLAs defined for workflows are met.
- Resources are utilized in an efficient way, i.e., the costs for leasing VMs over the reckoned time span should be optimized.
- Scheduling and reasoning need to be redone once the system landscape changes, as new workflow requests arrive, or the predicted resource utilization of VMs does not apply.
- In addition, scheduling and reasoning are done in regular intervals. This interval can be set by a system administrator.

In order to find a scheduling, we make use of the following constraints:

- Each Backend VM hosts exactly one service instance, i.e., it is not possible that different service types are instantiated at the same Backend VM.
- All VMs offer the same capabilities in terms of computational resources and costs.

B. Workflow Descriptions and Monitoring Data

ViePEP accepts new workflow requests at any time, and maintains a queue of workflow requests which are ordered by their individual priorities, i.e., they are sorted by the user-defined deadline for the workflow execution. In order to accept new workflows and create a scheduling plan for the queued workflow steps, a workflow description language is needed. Formally, a workflow is defined as following:

$$W_i = \{sla_w, s_1, s_2, ..., s_n\}$$

$$s_i = \{sla_s, id\}$$

$$sla_i = \{slo_1, slo_2, ..., slo_n\}, i \in \{w, s\}$$

W_i defines a workflow with ID i which consists out of 1 to n single sequential service steps expressed by s_1, ... ,s_n. Each service step must provide a mandatory field id which is the unique identifier of the service representing this step. A workflow can have an optional SLA (sla_i) consisting out of several Service Level Objectives (SLO) expressed by slo_1, ... , slo_n on workflow (sla_w) or service level (sla_s). For now, we consider solely the deadline as SLO, i.e., the scheduling takes into account that workflows have to be finished before a particular point of time; deadlines for single steps are then derived from this SLO if they are not explicitly defined. For this, we define

$$sla_w = deadline_w$$

$$sla_s = deadline_s$$

In the following, we assume that SLAs and deadlines defined by the workflow requester (user) are valid and achievable. For example, regarding deadlines this means that the execution duration of the single steps in a workflow request does not exceed the set workflow execution deadline and that the deadline is in the future.

ViePEP monitors service invocations continuously regarding execution duration and needed computational resources in terms of RAM and CPU. These values are assigned to the corresponding services, thus monitoring data for a particular service can be described as follows:

$$m_{ID} = \{d, cpu, ram, |ID|\}$$

m_{ID} defines one monitoring data entry for the service with the id ID, i.e., the monitoring data for one particular service invocation of service ID. d defines the service execution duration, cpu defines the CPU usage, ram defines the RAM usage, and $|ID|$ defines the number of concurrent invocations of service ID at the time of monitoring at the particular VM it is running on. For now, the CPU value is used for the resource demand prediction (see Section IV) and the duration value is needed to create a proper scheduling plan.

Based on this data, we are able to derive the needed resources and duration of a single service invocation, as presented in [16].

C. Scheduling Algorithm

As written above, ViePEP maintains a queue of workflow requests, which are sorted by the user-defined workflow execution deadlines. A user (workflow requester) is able to specify an execution deadline, i.e., the latest date and time when workflow execution has to be finished.

If this is the case, we can calculate the overall workflow execution duration t_w as follows:

$$t_w = \sum_{i=1}^{n} t_{s_i} + \epsilon$$

As can be seen, t_w is the sum of each single service invocation duration (t_{s_i}) plus a safety margin ϵ. t_{s_i} has been calculated based on the monitoring data as described above. ϵ can be set independently for each single workflow request by the user. A large ϵ indicates a low risk to not meet the execution deadline, as there is a certain time buffer if a service invocation takes longer than estimated. Vice versa, if ϵ has been set to a small value or even 0, there is a high risk that the execution deadline cannot be met as there is no time buffer left.

Based on t_w, the latest starting time $start_w$ can be calculated:

$$start_w = deadline_w - t_w$$

where $deadline_w$ is the given SLO which defines the workflow's execution deadline and t_w is the workflow execution duration calculated in the last step.

In order to compute at which point of time a particular workflow step has to be executed, thus the given SLO is not violated, we have to compute the latest starting time for each single workflow step s_i accordingly:

$$start_{s_i} = start_{s_{i+i}} - t_{s_i}$$

where $start_{s_{i+i}}$ defines the latest starting time of the next step in the workflow, and t_{s_i} defines the execution duration of step s_i. As can be seen, this is a recursive function, as it is necessary to derive the latest starting time of the next step. However, for the last step s_n in a workflow, the execution deadline is identical with the workflow execution deadline:

$$deadline_{s_n} = deadline_w$$

Hence, the latest starting time for the last step in a workflow is:

$$start_{s_n} = deadline_w - t_{s_n}$$

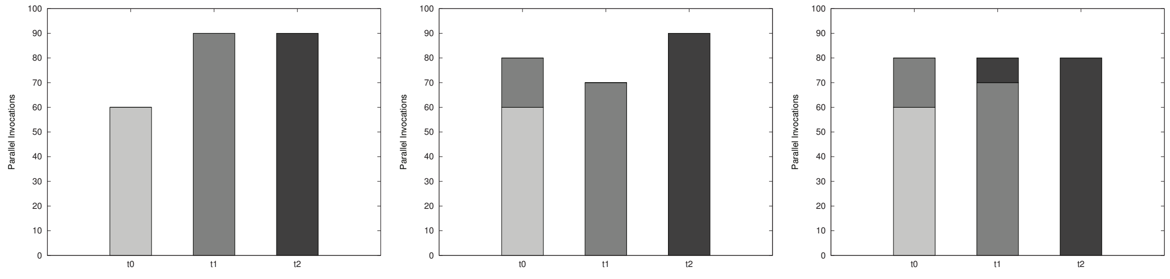
In case deadlines have been set for single services, the starting time for step s_i is set by:

$$start_{s_i} = deadline_{s_i} - t_{s_i}$$

After the starting time is calculated for each single workflow step in the queue, a full scheduling plan for the workflow executions is available, i.e., ViePEP has now the information at which point of time how many service invocations have to be performed. This information is forwarded to the Reasoner (Section IV) which is responsible for reasoning about the required demand of resources.

IV. Reasoning

In the previous section, we described how deadlines of individual workflow steps and the corresponding starting times are assigned in order to fulfill a given SLA. The result of this step is a detailed plan of which workflow step should be



(a) Initial State (b) First Round Optimized (c) Final State

Figure 2: Timeslot Scheduling

Algorithm 1 Method 'reasoning()'

```
 1: //Variable Initialization
 2: for ($i = 1; i \leq timeSlotsList$.size()$-1; i = i + 1$) do
 3:     $timeSlot_{t0}$ = $timeSlotsList$.get($i$);
 4:     $timeSlot_{t1}$ = $timeSlotsList$.get($i + 1$);
 5:     $demand_{t0}$ = computeResourceDemand();
 6:     if $demand_{t0}.isFullyUtilized()$ then
 7:         $continue()$;
 8:     end if
 9:     while ($demand_{t0}$.isNotFullyUtilized()) do
10:         moveInvocations($timeSlot_{t0}$, $timeSlot_{t1}$);
11:         $demand_t0$ = computeResourceDemand();
12:     end while
13:     reasoning();
14: end for
```

executed at which point of time. Based on this information, it is now possible for the Reasoner to derive the resource demand. To compute the needed amount of computational resources, each different service is considered separately, as only one service type per Backend VM is allowed to be instantiated.

Figure 2 represents a scheduling plan for a specific service type. On the horizontal axes, the time periods are shown, while the vertical axes show the number of planned parallel service invocations at a specific timeslot t_i. Figure 2a shows the initial state as it would be provided by the Scheduler: In t_0 there would be a total of 60 parallel invocations, in t_1 90 and in t_2 also 90. By using Ordinary Least Square (OLS) Linear Regression, we can compute the total amount of required resources for one timeslot based on historical monitoring data (see Section III-B) [18]. This process is based on the notion that a dependent variable – corresponding to the resource utilization of a Backend VM for a particular service type – can be derived through the linear combination of a set of independent variables, corresponding to the individual monitoring results for this service type. In the following, we will describe the outcome of the OLS-based regression analysis based on the example from Figure 2. For further details on the application of OLS to derive the resource demand for a particular service type, please refer to our previous work [16], where we presented this approach in detail.

In the example, the regression analysis has led to the outcome that a single Backend VM is able to handle 80 invocations of a particular service type in one single timeslot. Please note that this analysis would be done for all service types in parallel. Furthermore, the degree of maximum utilization of a Backend VM can be set as a parameter by the system admin. A high maximum utilization would lead to a higher saturation of the computational resources of a VM, but also yield a higher risk of underprovisioning, as there is only a small resource buffer.

Taking the original scheduling plan into account, in the first timeslot t_0, we have 60 parallel service invocations scheduled. The Reasoner would calculate the demand of resources for these 60 invocations and would derive that it is possible to do even more invocations in t_0, as there are still resources available and not exploiting them would lead to a waste of resources. Therefore, we present a reasoning algorithm which is able to move the scheduled invocations from one timeslot to another in order to fully utilize the acquired resources. The algorithm is presented in form of pseudocode in Algorithm 1.

The input for this algorithm is the list of queued workflow steps (services) which is sorted by their execution deadline ($timeSlotList$). Each timeslot is exactly as long as one service invocation lasts; however, as reasoning is done for each service separately, different services in a workflow will have different timeslot lengths. From line 2–14, we iterate over these time slots and consider each timeslot separately. Since the Scheduler has assigned a number of workflow steps to each timeslot, we can compute the amount of required resources for this timeslot (see line 5 in the Algorithm 1). This is done by calling the method $computeResourceDemand()$ which is based on the abovementioned OLS-based regression [16].

In the next step, we calculate if the required resources are fully utilized (line 6). If this is the case, we continue with the next timeslot (line 7), if not, service invocations from the next timeslot are preponed to the current timeslot in order to try to fully utilize the acquired resources (lines 9–12). Naturally, this leads to a lesser number of services in the next timeslot. Hence, it is necessary to replan the scheduling. Therefore, the $reasoning()$ method is called recursively (see line 13).

Table I: Evaluation Results

	Constant Arrival		Linear Arrival		Pyramid Arrival	
	Pure Scheduling	Scheduling + Reasoner	Pure Scheduling	Scheduling + Reasoner	Pure Scheduling	Scheduling + Reasoner
Number of Workflow Requests	20,000					
Interval between two Request Bursts (in Seconds)	10		10		20	
Number of Requests in one Burst	$y = 200$		$y = 10 * \lfloor \frac{x}{3} \rfloor + 10$		Based on Random Number	
Duration in Minutes (Standard Deviation)	25.67 (σ=2.09)	23.93 (σ=0.42)	44.52 (σ= 3.01)	42.04 (σ= 6.91)	40.22 (σ=0.67)	38.5 (σ=1.2)
Costs in VM-Minutes (Standard Deviation)	302.5 (σ=13.60)	277.44 (σ=7.18)	545.05 (σ=20.62)	537.72 (σ=14.03)	555.16 (σ=25.98)	545.56 (σ=17.30)

For the example, Figure 2b presents the result of the first round of scheduling: 20 invocations were shifted from t_1 to t_0. In the next step, the second time slot will be considered. After the first round of replanning the scheduling, only 70 invocations are scheduled for timeslot t_1. Since the available resources are able to handle more invocations in parallel, 10 further invocations are shifted from the t_2 to t_1. The result is presented in Figure 2c. At this point, we want to state that the presented numbers in this example are only for illustration and do not echo the result of a real evaluation. Evaluation results will be presented in the next section.

Based on the outcome of the replanning of the original scheduling, the Reasoner is now able to deduct the required resources and will correspondingly trigger the start and termination of Backend VMs, duplicate a Backend VM, move a running service to another Backend VM, or exchange the service hosted by a Backend VM by another service.

V. Evaluation

In the following, we will first shortly describe the evaluation scenarios and secondly we will present the results of the evaluation runs.

ViePEP and the scheduling and reasoning algorithms were implemented using Java 1.6. The evaluation was done in an OpenStack-based Cloud testbed. The single services have been deployed on an Apache Tomcat 6 as Application Server. The invocations were monitored using PSI-Probe. The reasoner made use of the Java Library Apache Commons Math to solve the OLS problem.

A. Evaluation Scenario

For the evaluation, we chose a particular workflow which has to be processed 20,000 times. This workflow consists out of 5 individual steps represented by RESTful services, i.e., a *Dataload Service* which loads data from a sensor, a *Pre-Processing Service* which performs some simple computations with the loaded data, a *Calculation Service* which processes the given data, a *Reporting Service* which generates a report out of the results, and a *Mailing Service* which delivers a mail containing the report. Each individual service needs a different amount of computational resources in terms of CPU and RAM. The maximum utilization parameter for the Backend VMs is set to 80%, i.e., scheduling and resource allocation will only utilize max. 80% of the offered resources of a Backend VM.

B. Arrival Pattern

We make use of three distinct workflow request arrival patterns:

- *Constant Arrival Scenario*: In this scenario, the workflow requests arrive in a constant manner. An automated client sends 200 requests simultaneously to the BPMS every 10 seconds, i.e., $y = 200$.
- *Linear Arrival Scenario*: In this scenario, the workflow requests arrive in a linear rising function, i.e., $y = k * \lfloor \frac{x}{3} \rfloor + d$. The number of constantly rising parallel requests is increased by 10 every 10 seconds.
- *Pyramid Arrival Scenario*: In this scenario, the workflow requests are send to the server according to a pyramid-like function. We start at 0 parallel requests and increase it to a randomly chosen peak. As soon as this peak is reached we decrease the number again to 0 and repeat this procedure until the maximum number of workflow requests has been covered (here: 20,000).

The different scenarios have been chosen in order to show how the algorithms perform under different degrees of volatility. While the number of workflow requests is quite predictable in the Constant and Linear Arrival scenarios, the Pyramid-based follows a more unpredictable arrival pattern. The arrival patterns are shown as "Workflow Requests" in Figure 3.

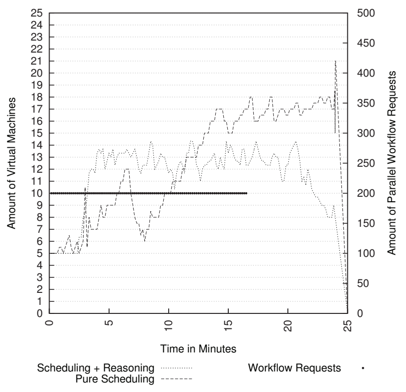
C. Metrics

In order to get reliable numbers, we executed each arrival pattern three times and evaluated the results against the following three quantitative metrics:

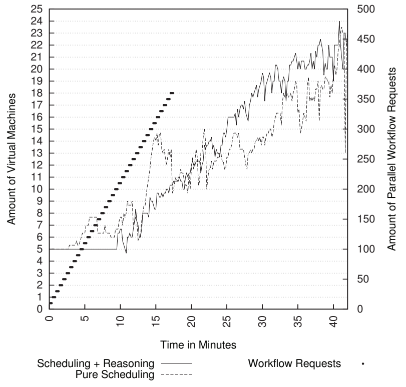
- *Duration in Minutes:* This metric defines the overall execution duration of all 20,000 workflows, i.e., this is the timespan from the first workflow request to the moment when the last workflow step has been processed.
- *Costs:* This metric defines the costs emerging from the acquired resources. The costs are defined in VM-Minutes, i.e., one VM-Minute is defined as one VM running for one minute. VM-Minutes can be directly mapped to common pricing schemes, e.g., from Amazon's EC2.

In the evaluation runs, we compare the "Scheduling + Reasoning" approach against a baseline which makes use of an ad hoc approach ("Pure Scheduling"), i.e., applies scheduling and resource allocation, but does not apply the reasoning. This means, that in the baseline scenario additional VMs are leased in an ad hoc approach, i.e., whenever the threshold is exceeded additional VMs are acquired, or if the load falls short of the lower threshold (20%), unneeded VMs are released.

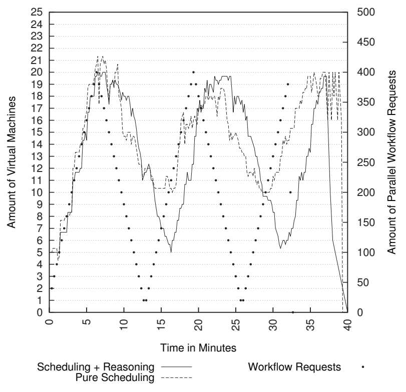
D. Results



(a) Constant Arrival Scenario



(b) Linear Arrival Scenario



(c) Pyramid Arrival Scenario

Figure 3: Evaluation Results

The results of our evaluation in terms of the average number of all evaluation runs can be found as numbers in Table I and as graphical presentation in Figure 3. "Pure Scheduling" denotes the ad hoc approach without reasoning, while "Scheduling + Reasoning" marks the application of the complete approach as presented in Sections III-IV.

Table I presents the observed metrics as presented in Section V-C for all three arrival scenarios. The table states the average of the evaluation runs including the standard deviation (σ). Figure 3 completes the presentation of the average evaluation results by depicting the arrival patterns ("Workflow Requests") over time and the number of running VMs until all workflow requests have been served. Again, evaluation results for "Pure Scheduling" and "Scheduling + Reasoning" are shown. In some cases it may happen that the "Workflow Requests" are ending before the whole scenario is finished. This can be reduced to the fact, that each workflow has a deadline in the future and some workflows are still waiting in the queue which have to be processed.

As presented in Table I, for all three observed arrival patterns, the application of "Scheduling + Reasoning" leads to lower costs if compared with the "Pure Scheduling": The Constant Arrival scenario is seen as the best case scenario. Both the "Pure Scheduling" and the combined approach lead to very low costs, i.e., 302.5 VM-Minutes vs. 277.44 VM-Minutes. The combined approach leads to cost savings of ~8.3%. This shows, that in this scenario, our approach was able to reduce the overall execution duration while allocating less resources. In contrast to that, the savings from the combined approach in the Linear Arrival scenario (~1.4%) and the Pyramid Arrival scenario (~1.7%) are comparably small. Obviously, this can be traced back to the fact that in these scenarios, there is not as much information available about future resource demands, as the number of workflow requests changes permanently and therefore it is more difficult to prepone service invocations to earlier timeslots. Furthermore, these scenarios require more changes in the numbers of Backend VMs per service, which also leads to higher costs.

Regarding the time savings, the Constant Arrival scenario features the largest saving for the "Scheduling + Reasoning" approach (~6.8%), while the Linear Arrival scenario (~5.6%) and the Pyramid Arrival scenario (~4.3%) feature smaller numbers. Taking the cost and time savings into account, it can be deduced that the combined approach speeds up the total processing time, but the Reasoner is not able to fill up the Backend VMs to the same degree for all arrival scenarios. Once again, this can be traced back that in the Linear Arrival and Pyramid Arrival scenario, information about workflow requests arrives in a more unpredictable manner and the number of Backend VMs per service is more volatile. As we can achieve the smallest cost savings in the Pyramid Arrival scenario, it can be deduced that this scenario is the worst case scenario for our approach. Hence, it will be interesting to investigate such a scenario in more detail in our future work.

Last but not least, it should not be forgotten, that the good results for the "Pure Scheduling" approach are due to the fact that our approach already takes into account the single deadlines for individual workflow steps. In general, both the "Pure Scheduling" and the "Scheduling + Reasoning" approaches led to a quite efficient resource allocation of Cloud-based computational resources. Therefore, through the application of these algorithms, ViePEP is able to decrease the risk of unwanted spendings produced by overprovisioning. Furthermore, the system also decreases the risk of workflow execution errors by preventing underprovisioning of resources.

VI. Related Work

To the best of our knowledge, so far, surprisingly little effort has been put into the application of Cloud-based computational resources for the execution of business processes [11], [14].

To start with, a number of researchers have done work in the field of dynamic resource allocation for BPM, e.g., [10], who apply reinforcement learning to solve this problem. Tsai et al. [19] provide a distributed BPMS, which mirrors our idea of placing the BPMS in a VM to scale it up if the load increases. However, these approaches do not apply Cloud resources.

In Cloud computing research, resource allocation for single services and applications has been regarded in a number of approaches, with a focus on cost-efficient resource allocation under given QoS or resource constraints, e.g., [20], [21], and the enforcement of SLAs, e.g., [22], [23]. The goal of other approaches is the achievement of a higher resource utilization [24]–[27] or maximizing the profit for an infrastructure broker [28]. Usually, rule-based thresholds are applied to identify the necessary actions (e.g., to start/stop VMs), but Li and Venugopal make use of machine learning to automatically scale an application up or down [26]. All these approaches do not take into account the process perspective, but focus on the ad hoc allocation of Cloud resources for single applications.

Taking into account a process perspective, the usage of Cloud resources for the execution of data- and computing-intensive tasks has been regarded for Scientific Workflows, e.g., [29]–[31]. Approaches for business processes/workflows are still scarce, but recently, some researchers have started to work on this topic: Andrikopoulos et al. present a migration model to execute workflows in the Cloud. The authors also state that current approaches to optimization for business processes do not take into account the technical level [14]. Gambi and Pautasso define design principles for RESTful BPM in the Cloud [32]. In contrast to our work, the authors do not divide process instances into single tasks, but place complete processes on a particular VM. Wang et al. make use of the same premise in their work on process runtime governance in the Cloud [33]. Again, benefits from resource sharing cannot be realized. Janiesch et al. [34] define a meta-model for elastic processes and therefore provide a conceptual framework for our work. However, the authors do not present algorithms for workflow scheduling and resource allocation.

Wei and Blake [35] and Bessai et al. [36], use the workflow model which is also applied in our work, i.e., workflows are composed from single software services, which can be run in the Cloud. In both works, resource allocation for workflows is taken into account. However, only one workflow is regarded at a time, i.e., cost benefits from invoking the same service as part of different workflows cannot be realized.

In a later work, Wei and Blake extend their model in order to be able to take into account different workflows, which share services and therefore Cloud-based resources [37]. Notably, the authors assume that different service instances will most likely consume a different amount of resources due to differing workloads, and resource allocation therefore needs to take this into account. While this is a valid assumption, it hinders the deduction of future resource demands from historical data. Hence, the authors apply an ad hoc approach to resource allocation instead of the workflow queue-based one we apply. In our opinion, the approach presented in our work and the work by Wei and Blake complement each other – if the resource demand of services is volatile, it could indeed be useful to integrate ideas from their work into ViePEP. However, if the resource demand is predictable, this is not necessary and the benefits from taking knowledge from the workflow queue into account will lead to more efficient resource allocation.

VII. Conclusion & Future Work

In this paper, we presented scheduling and reasoning algorithms for elastic processes, i.e., for the execution of workflows using Cloud resources. The decision at what point of time a particular workflow should be processed is based on user-defined non-functional requirements such as execution deadlines. Depending on a computed scheduling plan, a Reasoner is able to deduce the resource demand and allocate resources accordingly. Furthermore, the Reasoner changes the scheduling plan based on the current resource situation. As we have shown in the evaluation, our "Pure Scheduling" approach already leads to good results in terms of costs and total execution time; the "Scheduling + Reasoning" approach leads to further cost and time savings.

As discussed above, research on elastic process landscapes in the Cloud, including dynamically leasing and releasing of computational resources, is just at the beginning and we are sure, that more and more approaches addressing this topic will follow in the next years. In our own work, we want to take into account more complex workflow patterns – currently, we focus on sequential workflows, but in the future, we want to take into account loops, branches and further patterns as these are more realistic and closer to real-world scenarios. Another aspect will be the consideration of probabilistic QoS values based on historical data. Further, we are planning to accept additional SLOs such as the maximum of costs a user is willing to spend or other service-specific QoS attributes.

So far, we apply one particular VM type in our scenario, but we have already started to take into account different types of VMs with different resource settings and costs. Further, it could be interesting to take into account scaling *up* (i.e., change the size of a particular Backend VM) instead of the currently applied scaling *out* approach.

Last but not least, as we have shown in our evaluation, scheduling and reasoning results (in terms of cost and time savings) differ substantially if taking into account a best or worst case scenario. Hence, it could be helpful to make use of different scheduling and reasoning approaches based on the volatility of a particular domain or scenario.

ACKNOWLEDGMENTS

This work is partially supported by the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201).

REFERENCES

[1] W. M. P. van der Aalst, A. H. M. T. Hofstede, and M. Weske, "Business process management: A Survey," in *International Conference on Business Process Management (BPM 2003)*, ser. LNCS, vol. 2678. Springer, 2003, pp. 1–12.

[2] S. Dustdar and H. L. Truong, "Virtualizing Software and Humans for Elastic Processes in Multiple Clouds – a Service Management Perspective," *Inter. J. of Next-Generation Computing*, vol. 3, no. 2, pp. 109–126, 2012.

[3] H. Gewald and J. Dibbern, "Risks and benefits of business process outsourcing: A study of transaction services in the German banking industry," *Information & Management*, vol. 46, no. 4, pp. 249–257, 2009.

[4] E. Santacana, G. Rackliffe, L. Tang, and X. Feng, "Get Smart," *IEEE Power and Energy Magazine*, vol. 8, no. 2, pp. 41–48, 2010.

[5] S. Rohjans, C. Dänekas, and M. Uslar, "Requirements for Smart Grid ICT Architectures," in *Third IEEE PES Innovative Smart Grid Technologies (ISGT) Europe Conference*. IEEE, 2012, pp. 1–8.

[6] B. Ludäscher, M. Weske, T. M. McPhillips, and S. Bowers, "Scientific Workflows: Business as Usual?" in *7th International Conference on Business Process Management (BPM 2009)*, ser. LNCS, vol. 5701. Springer, 2009, pp. 31–47.

[7] S. Dustdar and W. Schreiner, "A survey on web services composition," *Inter. J. of Web and Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

[8] B. Mutschler, M. Reichert, and J. Bumiller, "Unleashing the Effectiveness of Process-Oriented Information Systems: Problem Analysis, Critical Success Factors, and Implications," *IEEE Trans. on Systems, Man, and Cybernetics, Part C*, vol. 38, no. 3, pp. 280–291, 2008.

[9] A. Kumar, W. M. P. van der Aalst, and H. M. W. E. Verbeek, "Dynamic Work Distribution in Workflow Management Systems: How to Balance Quality and Performance," *J. of Management Information Systems*, vol. 18, no. 3, pp. 157–194, 2002.

[10] Z. Huang, W. M. P. van der Aalst, X. Lu, and H. Duan, "Reinforcement learning based resource allocation in business process management," *Data Knowl. Eng.*, vol. 70, no. 1, pp. 127–145, 2011.

[11] S. Dustdar, Y. Guo, B. Satzger, and H. L. Truong, "Principles of Elastic Processes," *IEEE Internet Computing*, vol. 15, no. 5, pp. 66–71, 2011.

[12] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computing Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[13] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*. Recommendations of the National Institute of Standards and Technology, 2011.

[14] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to Adapt Applications for the Cloud environment – Challenges and Solutions in Migrating Applications to the Cloud," *Computing*, vol. 95, no. 6, pp. 493–535, 2013.

[15] S. Schulte, P. Hoenisch, S. Venugopal, and S. Dustdar, "Introducing the Vienna Platform for Elastic Processes," in *Performance Assessment and Auditing in Service Computing Workshop (PAASC 2012) at 10th International Conference on Service Oriented Computing (ICSOC 2012)*, ser. LNCS, vol. 7759. Springer, 2013, pp. 179–190.

[16] P. Hoenisch, S. Schulte, S. Dustdar, and S. Venugopal, "Self-Adaptive Resource Allocation for Elastic Process Execution," in *IEEE 6th International Conference on Cloud Computing (CLOUD 2013)*. IEEE, 2013, pp. 220–227.

[17] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, pp. 50–58, 2010.

[18] M. Verbeek, *A Guide to Modern Econometrics*, 3rd ed. John Wiley & Sons, Hoboken, NJ, 2008.

[19] C.-H. Tsai, K.-C. Huang, F.-J. Wang, and C.-H. Chen, "A distributed server architecture supporting dynamic resource provisioning for BPM-oriented workflow management systems," *J. of Systems and Software*, vol. 83, no. 8, pp. 1538–1552, 2010.

[20] Q. Cao, Z.-B. Wei, and W.-M. Gong, "An Optimized Algorithm for Task Scheduling Based on Activity Based Costing in Cloud Computing," in *3rd International Conference on Bioinformatics and Biomedical Engineering (ICBBE 2009)*. IEEE, 2009, pp. 1–3.

[21] U. Lampe, T. Mayer, J. Hiemer, D. Schuller, and R. Steinmetz, "Enabling Cost-Efficient Software Service Distribution in Infrastructure Clouds at Run Time," in *4th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2011)*. IEEE, 2011, pp. 1–8.

[22] R. Buyya, R. Ranjan, and R. N. Calheiros, "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," in *10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010)*, ser. LNCS, vol. 6081. Springer, 2010, pp. 13–31.

[23] V. Cardellini, E. Casalicchio, F. Lo Presti, and L. Silvestri, "SLA-aware Resource Management for Application Service Providers in the Cloud," in *First International Symposium on Network Cloud Computing and Applications (NCCA '11)*. IEEE, 2011, pp. 20–27.

[24] V. C. Emeakaroha, I. Brandic, M. Maurer, and I. Breskovic, "SLA-Aware Application Deployment and Resource Allocation in Clouds," in *COMPSAC Workshops 2011*. IEEE, 2011, pp. 298–303.

[25] A. Kertesz, G. Kecskemeti, and I. Brandic, "An Interoperable and Self-adaptive Approach for SLA-based Service Virtualization in Heterogeneous Cloud Environments (forthcoming)," *Future Generation Computer Systems*, vol. NN, no. NN, 2013.

[26] H. Li and S. Venugopal, "Using Reinforcement Learning for Controlling an Elastic Web Application Hosting Platform," in *8th International Conference on Autonomic Computing (ICAC 2011)*. ACM, 2011, pp. 205–208.

[27] E. Juhnke, T. Dörnemann, D. Bock, and B. Freisleben, "Multi-objective Scheduling of BPEL Workflows in Geographically Distributed Clouds," in *The 4th International Conference on Cloud Computing (IEEE CLOUD 2011)*. IEEE, 2011, pp. 412–419.

[28] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-Driven Service Request Scheduling in Clouds," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010)*. IEEE, 2010, pp. 15–24.

[29] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the Use of Cloud Computing for Scientific Workflows," in *IEEE Fourth International Conference on e-Science (eScience'08)*. IEEE, 2008, pp. 640–645.

[30] G. Juve and E. Deelman, "Scientific Workflows and Clouds," *ACM Crossroads*, vol. 16, no. 3, pp. 14–18, 2010.

[31] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010)*. IEEE, 2010, pp. 400–407.

[32] A. Gambi and C. Pautasso, "RESTful Business Process Management in the Cloud," in *5th International Workshop on Principles of Engineering Service-Oriented Systems (PESOS 2013) in conjuction with the 35th International Conference on Software Engineering (ICSE 2013)*. IEEE, 2013.

[33] M. Wang, K. Y. Bandara, and C. Pahl, "Process as a Service – Distributed Multi-tenant Policy-based Process Runtime Governance," in *2010 IEEE International Conference on Services Computing (SCC 2010)*. IEEE, 2010, pp. 578–585.

[34] C. Janiesch, I. Weber, J. Kuhlenkamp, and M. Menzel, "Optimizing the Performance of Automated Business Processes Executed on Virtualized Infrastructure (forthcoming)," in *47th Hawaii International Conference on System Sciences (HICSS 2014)*. IEEE, 2014.

[35] Y. Wei and M. B. Blake, "Adaptive Service Workflow Configuration and Agent-Based Virtual Resource Management in the Cloud," in *2013 IEEE International Conference on Cloud Engineering (IC2E 2013)*. IEEE, 2013, pp. 279–284.

[36] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, "Resources allocation and scheduling approaches for business process applications in Cloud contexts," in *4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012)*. IEEE, 2012, pp. 496–503.

[37] Y. Wei and M. B. Blake, "Decentralized Resource Coordination across Service Workflows in a Cloud Environment," in *22nd IEEE International Conference on Collaboration Technologies and Infrastructures (WETICE 2013)*. IEEE, 2013, pp. 15–20.