COST-DRIVEN OPTIMIZATION OF CLOUD RESOURCE ALLOCATION FOR ELASTIC PROCESSES

Stefan Schulte[1], Dieter Schuller[2], Philipp Hoenisch[1], Ulrich Lampe[2], Ralf Steinmetz[2], Schahram Dustdar[1]
[1] Distributed System Group, Vienna University of Technology, Austria
Email: {s.schulte, p.hoenisch, dustdar}@infosys.tuwien.ac.at
[2] Multimedia Communications Lab (KOM), Technische Universität Darmstadt, Germany
Email: {firstName.lastName}@KOM.tu-darmstadt.de

Abstract

Today's extensive business process landscapes make it necessary to handle the execution of a large number of business processes and individual process steps. Especially if process steps require the invocation of resource-intensive applications or a large number of applications need to be executed concurrently, process owners may have to allocate extensive computational resources, leading to high fixed cost.

In the work at hand, we propose an alternative to the provision of fixed resources, based on automatic leasing and releasing of Cloud-based computational resources. For this, we present an integrated approach which addresses the cost-driven optimization of Cloud-based computational resources for business processes in order to realize so-called Elastic Processes. Through an evaluation, we show the practical applicability and benefits of our contributions. Specifically, we find that our approach substantially reduces the cost compared to an ad hoc approach.

Keywords: Elastic Processes, Cloud Computing, Business Process Execution

1. INTRODUCTION

Nowadays, IT-support for the execution of business processes is an essential prerequisite in many industries. For example, in the finance industry, trade settlement or execution control processes are executed automatically (Gewald, Dibbern, 2009). In the energy domain, computational resources are needed to carry out essential decision processes and a particular necessity is to support the processing of large amounts of data in so-called Smart Grids (Rohjans et al., 2012).

Especially in large companies, the number of different business process models available can become extensive (Breu et al. 2013; Jin et al., 2013). Correspondingly, a business process landscape may comprise a very large number of running business process instances, all of which are made up from single tasks (i.e., process steps) with differing computational resource demands. Over time, the invocation of new process instances and the completion of running process instances lead to ever-changing computational resource demands, which need to be met by a company. Apparently, computational resource demands during peak times (i.e., when many and/or resource-intensive tasks need to be carried out concurrently) will be much higher than in normal times – especially in volatile domains (Maurer et al., 2013).

On the one hand, permanently providing computational resources that can cover the demand during peak times leads not only to high fixed cost, but the resources will not be utilized most of the time (*overprovisioning*). On the other hand, providing computational resources which can cover only part of the processes' resource demand, will lead to lower fixed cost, but also to the risk that some processes cannot be carried out during peak times (*underprovisioning*) or will suffer from low Quality of Service (QoS).

To avoid the drawbacks arising due to over- and underprovisioning, computational resources should be scalable, i.e., the available resources should be in- or decreased based on the demands of the running and future business process instances. Applying Cloud technologies to provide the needed resources exactly allows this – (i) leasing and releasing computational resources in an *on-demand, utility-like fashion*, (ii) *rapid elasticity* through scaling the infrastructure up and down if necessary, and (iii) pay-per-use through *metered service* (Armbrust et al., 2010; Buyya et al., 2009).

So far, only few researchers have provided methods and solutions to facilitate *Elastic Processes*, i.e., processes which are carried out using elastic Cloud resources (Dustdar et al., 2011; Andrikopoulous et al., 2013). Current Business Process Management Systems (BPMS) do not only "lack the ability to learn, mine, and reason suitable resource allocation knowledge in business process execution" (Pesic and van der Aalst, 2007; Huang et al., 2011), but are also not able to make use of Cloud-based computational resources. In our former work (Schulte et al., 2013; Hoenisch et al., 2013; Hoenisch et al., 2013a), we have presented the *Vienna Platform for Elastic Processes (ViePEP)*, which combines the functionalities of a BPMS with that of a Cloud resource management system. ViePEP is able to schedule complete processes as well as the involved single tasks, and lease and release Cloud-based computational resources in terms of Virtual Machines (VMs) while taking into account Service Level Objectives (SLOs) defined by the process owners.

Within this paper, we extend our former work by addressing the problem of online Cloud resource allocation for

Elastic Processes based on process requests from various clients (process owners). In this scenario, it is necessary to schedule task executions and lease and release Cloud resources in order to carry out the single tasks under given SLOs. To encounter the complexity of this scenario, it is necessary to predict the resource demands of tasks, develop a cost model, predict the cost, and perform a cost/performance analysis. This has to be done continuously, as new process requests arrive, software services representing single process tasks do not behave as predicted, or a process instance is changed by the process owner. The goal is to provide cost-efficient process scheduling that takes into account the given SLOs and leases and releases Cloud-based computational resources (i.e., VMs) in order to optimize cost. Hence, the main contributions in this paper are:

- We formulate a model for scheduling and resource allocation for Elastic Processes.
- We design a heuristic based on the proposed model.
- We integrate this work into the Vienna Platform for Elastic Processes.

The remainder of this paper is organized as follows: In Section 2, we introduce the overall scenario, ViePEP, and some prerequisites for our research work. Afterwards, we present scheduling and resource allocation solution for Elastic Processes – for this, we define an integer linear optimization problem and a corresponding heuristic. We evaluate the scheduling and resource allocation algorithms through ViePEP-based testbed experiments (Section 4). In Section 5, we discuss the related work. The paper closes with a summary and an outlook on our future work.

2. PRELIMINARIES

2.1 GENERAL SCENARIO

Within this paper, we assume that a business process landscape is made up from a large number of business processes, which can be carried out automatically. The part of a business process which can be executed using machine-based computational resources is also known as a workflow (Ludäscher et al., 2009). Therefore, in the remainder of this paper, we will use the term workflow in order to identify an executable process. The automated processing of such workflows is a prominent field of research and has resulted in various concepts, methodologies, and frameworks (Mutschler et al., 2008). In recent years, the focus of this research was primarily on the composition of workflows from software services (Dustdar and Schreiner, 2005; Schuller et al., 2012).

While we also assume that workflows are composed of software services, we do not expect that companies are willing to outsource important services fully to external providers, since these services will then be outside the control domain of the process owner. In contrast, making use of private and public Cloud resources to host service instances, which are then invoked as workflow steps, leaves the control with the process owner: If particular service instances or VMs fail, the process owner or the admin of the business process landscape is able to deploy further service instances.

Making use of VMs to host particular services allows sharing resources among workflows, as the same service instance might be invoked within different workflows at the same time. Notably, it is important to distinguish between *services*, which are the basic building blocks of workflows, *service instances*, which are hosted by VMs, and *service invocations*, which denotes the unique execution of a service instance in order to serve a particular workflow request, i.e., a workflow instance.

In our scenario, workflows may be requested at any time by process owners and may be carried out in regular intervals or nonrecurring. It is the duty of a software framework (in our case, ViePEP), which combines the functionalities of a BPMS and a Cloud resource management system, to accept incoming workflow requests, schedule the workflow steps, and lease and release computational resources based on the workflow scheduling plan.

Process owners are able to define different QoS constraints as SLOs on the level of workflow instances. Without a doubt, the execution deadline is the most important SLO: Some business-uncritical workflows may have very loose deadlines, while other workflows need to be carried out immediately and finished as soon as possible. Usually, workflow instances will have a defined maximum execution time or a defined deadline. Process owners are able to define complex workflows which feature AND, XOR, or loop patterns. However, we assume that the next steps in a particular workflow instance are always known.

We assume that the business process landscape is volatile, i.e., ever-changing, since workflow requests may arrive anytime. Furthermore, changes may have to be necessary since services or VMs are not delivering the expected QoS, or the next steps in a workflow instance are not as planned.

For achieving an efficient scheduling and invocation of workflows and corresponding services, Cloud-based computational resources in terms of VMs required for invoking respective services have to be leased and released such that the total cost arising from leasing aforementioned Cloud resources is minimized. In addition, it has to be made sure that given QoS constraints such as deadlines, until which corresponding workflows have to be finished, are satisfied. The closer the deadline is for a certain workflow instance, the higher is the importance to schedule and execute corresponding services accomplishing its tasks. If not carefully considered and scheduled, further additional Cloud resources will have to be leased and paid in order to execute workflow instances that cannot be delayed any further. For avoiding such situations in which extra resources have to be leased due to an inefficient scheduling strategy, the execution of workflow instances along with the leasing and releasing of Cloud resources has to be optimized. This makes it necessary to facilitate self-adaptation and -optimization of the overall business process landscape through replanning of workflow scheduling and resource allocation.

2.2 Self-Adaptation for Elastic Processes

Self-adaptation is a common concept from the field of Autonomic Computing and includes *self-healing*, i.e., the ability of a system to detect and recover from potential problems and continue to function smoothly, *self-configuration*, i.e., the ability of a system to configure and reconfigure itself under varying and unpredictable conditions, and *self-optimization*, i.e., the ability to detect suboptimal behavior and optimize itself to improve its execution (Kephart and Chess, 2003). The focus of this paper is on self-optimization.

In order to motivate the functionalities and components needed to provide self-optimization of a Cloud-based process landscape, we make use of the well-known MAPE-K cycle shown in Figure 1. As the scenario at hand is highly dynamic due to permanently arriving workflow requests and changing Cloud resource utilization, a continuous alignment to the new system status is necessary. Using the MAPE-K cycle, the process landscape is continuously monitored and optimized based on knowledge about the current system status. In the following, we will briefly discuss the four phases of this cycle:

- **Monitor**: In order to adapt a system, it is first of all necessary to monitor the system status. In the scenario at hand, this includes the monitoring of the status of single VMs in terms of CPU, RAM, and network bandwidth utilization, and the non-functional behavior of services in terms of response time and availability.
- **Analyze**: To achieve Elastic Process execution, it is necessary to analyze the monitored data and reason on the general knowledge about the system. In short, this analysis is done in order to find out if there is currently under- and overprovisioning regarding the computational resources (VMs) and to detect Service Level Agreement (SLA) violations in order to carry out corresponding countermeasures (e.g., provide further VMs or re-invoke a service instance).
- **Plan**: While the analysis of the monitored data and further knowledge about the system aims at the current system status, the planning also takes into account the future resource needs derived from the knowledge about future workflow steps, their SLOs, and the estimated resource requirements and runtimes. For this, a workflow scheduling and resource allocation plan needs to be generated.
- **Execute**: As soon as the plan is set up, each workflow step is executed corresponding to this plan.
- **Knowledge Base**: While not really a part of the cycle, the Knowledge Base stores information about the system configuration. In our case, this is the knowledge about which service instances are running on which VMs, how many VMs are currently part of the system, and of course the knowledge about requested workflow and services instances.

In order to provide self-adaptation, ViePEP will have to support all four phases of the cycle and provide a Knowledge Base.

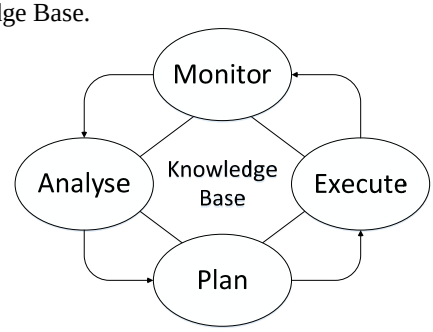

FIGURE 1: MAPE-K CYCLE (KEPHART AND CHESS, 2003)

2.3 The Vienna Platform for Elastic Processes

Figure 2 (using FMC notation[1]) depicts the high-level components of ViePEP and the message flow between them. The core functionalities of ViePEP are:

- Provisioning of an interface to the Cloud, allowing leasing and releasing Cloud-based computational resources (VMs) on demand.
- Execution of workflow steps by instantiating services on VMs and invoking the service instances in workflows instances.
- Dynamical scheduling of incoming requests for workflows based on their QoS requirements, i.e., timeliness in terms of maximum execution time or a deadline.
- Monitoring the deployed VMs in terms of resource utilization and monitor the QoS of service invocations.

ViePEP is made up from two major components and three helper components: The *BPMS VM* hosts the central functionalities of ViePEP, i.e., the functionalities for resource allocation and workflow scheduling. Resource allocation is done at the PaaS level, i.e., ViePEP is able to lease and release VMs from Cloud providers and allocate these resources to particular workflow steps. Workflows share resources as they are able to concurrently invoke the same service instance.

The *Workflow Manager* is the subcomponent both responsible for receiving workflow requests from the process owners via the *Client API* (see below) and for invoking single service instances running in a *Backend VM*. Furthermore, it monitors service invocations in order to control whether the service instance delivers the expected QoS and starts corresponding countermeasures if necessary.

The information which services need to be invoked at what point of time is generated by the *Scheduler* and the *Reasoner*. The former subcomponent is responsible to derive a detailed scheduling plan corresponding to service and workflow deadlines, while the latter subcomponent estimates the needed resources (VMs) based on the computed scheduling and sends corresponding requests to the Cloud

[1] http://www.fmc-modeling.org/

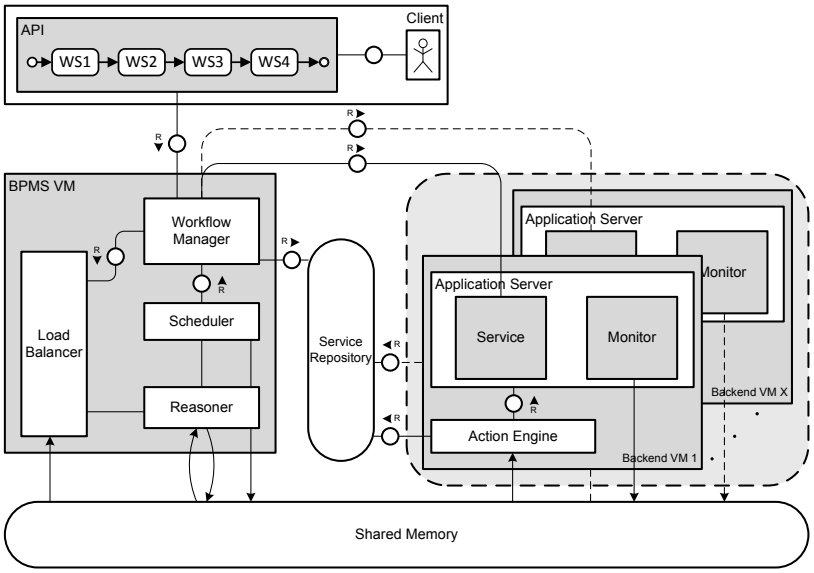

FIGURE 2: THE VIENNA PLATFORM FOR ELASTIC PROCESSES (VIEPEP)

providers. The functionality of these two subcomponents is defined through the optimization approach and heuristic presented in Section 3. The Reasoner interacts with the *Load Balancer* in order to estimate which Backend VM provides free resources for particular service instances and therefore could be used to invoke a service instance in the future.

The subcomponents of the ViePEP BPMS are placed in a VM to avoid that it becomes a bottleneck. Through vertical scaling, it is possible to provide ViePEP with additional resources if the business process and Cloud landscape becomes increasingly complex.

Whenever the Reasoner issues a request to the Cloud, either a *Backend VM* will be started and when its *Action Engine* is ready, a new service instance will be deployed, or (in case the Backend VM is already running) the Action Engine is directly able to execute the corresponding request. ViePEP allows the following requests:

- *Start* a new Backend VM, which includes deploying a new service instance on it.
- *Duplicate* a Backend VM including the service instance.
- *Terminate* a Backend VM, which marks the Backend VM as "phasing out" and will prevent the Load Balancer from requesting further service invocations from this Backend VM. Once all previously assured service invocations have been finished, the VM is terminated.
- *Exchange* the hosted service instance on the Backend VM. This will also prevent the Load Balancer from requesting any further invocations of this particular service instance. Once all assured service invocations have been finished, the instantiated service is replaced by another one.
- *Move* a service instance from one Backend VM to another one (with different computational resources).

Within the Backend VM, service instances are hosted in an *Application Server*, which also features a *Monitor* to observe the current load on the Backend VM and provide this information to the Load Balancer via the *Shared Memory*. This Monitor should not be confused with the monitoring capabilities of the Workflow Manager, which monitors the response time of service invocations.

The *Shared Memory* and the *Service Repository* are helper components. The latter hosts the service descriptions as well as their implementations as portable Web application ARchives (WAR). This repository allows searching for services and deploying them on a ViePEP Backend VM. The Shared Memory provides a distributed database which is used to send requests from the BPMS VM to the Backend VM and stores monitoring data. We chose MozartSpaces (Kühn et al., 2009) for this, as it allows easily deploying and accessing a peer-to-peer-based, distributed shared memory. In addition, MozartSpaces allows sending notifications with very low latency.

Finally, the *Client API* allows process owners to define workflows including SLOs. An owner may request many workflows consecutively or in parallel. When the request is submitted to the BPMS VM via the Workflow Manager, a new workflow instance is generated and taken into account in workflow scheduling and resource allocation.

2.4 PREREQUISITES

Before we formulate a model for workflow scheduling and resource allocation and define a corresponding heuristic, it is necessary to introduce some prerequisites in order to determine the scope of our work.

First, each Backend VM hosts exactly one service instance; a particular service may be instantiated arbitrarily often at different VMs. Second, defined deadlines are realistic, i.e., the deadlines can be met if providing corresponding

resources. However, defined deadlines may be violated because of faults in the Cloud or in the network. In this case, the service invocation in question will be immediately carried out again. Third, it is possible to derive the total computation time and resource utilization of a particular service on a particular VM type from historical data: Instances of the same service behave similarly with regard to resource consumption and runtime of single invocations. Fourth, computational resources in terms of additional VMs can be leased from different Cloud providers; it is also possible to combine resources from public and private Clouds. Indefinite resources are available. Hence, all resource demands of the business process landscape can be met by the available Cloud-based computational resources (i.e., the workflow scheduling and resource allocation are *effective*). Fifth, different types of VMs with different computational resources (CPUs, RAM, bandwidth, …) are available from different Cloud providers. Following Amazon's EC2 pricing scheme, the cost of these VMs are proportional, e.g., the cost of a VM with 2 cores are half of the cost of a VM with 4 cores with the same specifications. Sixth, it is the goal of our model to minimize the overall cost arising from leasing VMs (i.e., scheduling and resource allocation are *efficient*).

3. SOLUTION APPROACH

We formulate the problem of scheduling workflow instances and their individual services, respectively, as optimization problem. The general approach proposed in the work at hand for achieving an optimized scheduling and resource allocation is presented in Section 3.1. A formal specification of the corresponding optimization problem is provided in Section 3.2. Finally, in Section 3.3, we describe a heuristic solution method for efficiently solving the optimization problem.

3.1 GENERAL APPROACH

For achieving the necessary workflow scheduling and resource allocation, the deadlines indicating the time when the corresponding workflow instances have to be finished are considered. In addition, in order to minimize the total cost of (Backend) VMs leased for executing workflow requests and corresponding services, respectively, the leased VMs should be utilized as much as possible, i.e., leased but unused resource capacities should be minimized. Thus, in order to reduce the necessity of leasing additional VMs, we try to invoke services that cannot afford further delays firstly on already leased VMs before leasing additional VMs if the remaining resource capacities of already leased VMs are sufficient. Only if the resource requirements of workflows cannot be covered by already leased VMs, additional VMs will be leased. For instance, if the deadlines for certain workflow instances allow delaying their service invocations to another period, it can be beneficial to release leased resources and delay such services invocations. However, it has to be considered that those service invocations have to be scheduled for one of the subsequent optimization periods to ensure that corresponding deadlines are not violated.

In addition, since VMs are leased only for a certain time period and the execution of already scheduled invocations will be finished at a future point in time, the scheduling strategy to be developed cannot be static, i.e., the optimization approach for scheduling service invocations should not be applied only once. It rather has to be applied multiple times at different optimization points in time. In this respect, it has to be noted that potentially further requested workflow instances may have to be served, which additionally have to be considered when carrying out an optimization step.

Thus, an efficient scheduling strategy has to review allocated service instances and scheduled service invocations periodically and carry out further optimization steps for considering dynamically changing requirements and keeping the amount of leased but unused resource capacities low. Furthermore, the resource demand and average runtime for single service instances needs to be known in advance – for this, an approach based on linear regression – as presented in our former work (Hoenisch et al., 2013) – will be applied.

3.2 OPTIMIZATION PROBLEM

In this section, we model the scheduling and allocation of workflows and services as an optimization problem. Since Cloud resources are only leased for a limited time period, it has to be decided whether those resources or even additional resources have to be leased for another period, or whether leased resources can be released again. In this respect, we aim at utilizing leased Cloud resources, i.e., VMs in the context of this paper, to their full capacities. Further, additional resources will only be leased if capacities of already leased VMs are not sufficient to cover and carry out further service invocations that cannot be delayed.

For considering the different (optimization) time periods, the index t will be used. Depending on these periods, the parameter τ_t refers to the actual *time* and point in time, respectively, indicated by a time period t. For scheduling different workflows, multiple workflow templates are considered. The set of workflow templates is labeled with W, where $w \in W = \{1, \dots, w^{\#}\}$ refers to a certain workflow template. The set of workflow instances that have to be considered during a certain period t corresponding to a certain workflow template w is indicated by I_w, where $i_w \in I_w = \{1, \dots, i_w^{\#}\}$ refers to a certain workflow instance. The total number of workflow instances that have to be considered in period t is indicated by $i_w^{\#}$. Please note that *considering* a certain instance i_w in period t does not necessarily result in invoking corresponding service instances in this period. It rather makes sure that it is considered for the optimization step conducted in period t, which may lead to the decision that this instance is further delayed – until another optimization step is carried out in a subsequent period. The *remaining* execution time for executing a certain instance i_w, which might involve invoking multiple service

instances for accomplishing the different tasks of a certain workflow instance, is indicated by the parameter e_{i_w}. Thus, by specifying the parameter e_{i_w} as the *remaining* execution time of a certain workflow instance, we account for the fact that certain tasks of this instance might have already been accomplished by having invoked corresponding service instances in previous periods.

It has to be noted that executing workflow instances refers to invoking a service j_{i_w} that accomplishes a task of those workflow instances. The service, executing the *next* task of a workflow instance, is labeled with $j^*_{i_w}$. If, for instance, a workflow instance consists of five tasks, its *remaining* execution time e_{i_w} is determined by the sum of the execution times $e_{j_{i_w}}$ the services j_{i_w} require for accomplishing the different tasks of the workflow instance, i.e., $e_{i_w} = \sum_{j_{i_w} \in J_{i_w}} e_{j_{i_w}}$. The set J_{i_w} thereby represents the set of services that have to be invoked for accomplishing all tasks of workflow instance i_w. Allocating and executing this instance refers to invoking the *next* service $j^*_{i_w}$ that accomplishes the next task, i.e., the first task in this example. Thus, four tasks of this workflow instance still remain unaccomplished. After having invoked service $j^*_{i_w}$, the remaining execution time e_{i_w} for this workflow instance is reduced by the execution time $e^*_{j_{i_w}}$ of the invoked service. Thus, e_{i_w} can be determined by adding up the services' execution times for the remaining four tasks of this workflow instance (Hoenisch et al., 2013a). The corresponding workflow instance needs to be executed again, i.e., a service instance accomplishing the second task has to be invoked. For accomplishing this second task, the corresponding service becomes the *next* service $j^*_{i_w}$.

The deadline at which the execution of a workflow instance has to be finished is indicated by the parameter d_{i_w}. Assuming a continuous flow of time, d_{i_w} refers to an actual *point in time* as, for instance, "23.10.2013, 14:10:00". For executing a certain workflow instance i_w, i.e., for invoking the *next* service $j^*_{i_w}$, of a workflow instance, a certain amount of computational resources from a VM are required. The resource requirement for the corresponding next services is indicated by r_{i_w}.

Regarding the leasing of Cloud resources, we assume different types v of VMs. The set of VM types is indicated by the parameter V, where $v \in V = \{1, \dots, v^{\#}\}$ refers to VM type v. The corresponding resource supply of a VM (in terms of CPU, RAM, bandwidth) of type v is indicated by the parameter s_v. For counting and indexing leased VM instances of type v, the variable k_v is used. Although in theory unlimited, we assume the number of leasable VM instances of type v in a time period t to be restricted by $k_v^{\#}$ for modeling reasons, i.e., in order to make the idea of indefinite resources, provided by Clouds, tangible.

Model 1 Optimization Problem

Objective function

(1) Minimize $\sum_{v \in V} c_v \cdot \gamma_{\{v,t\}} + \sum_{v \in V} \sum_{k_v \in K_v} y_{k_v,t} \cdot f_{k_v}$

so that

(2) $\tau_{t+1} + e_{i_w} - e^*_{j_{i_w}} \cdot x_{i_w,v_k,t} \leq d_{i_w} \; \forall w \in W, i_w \in I_w$

(3) $\tau_{t+1} \leq \tau_t + e^*_{j_{i_w}} \cdot x_{i_w,v_k,t} + (1 - x_{i_w,v_k,t}) \cdot M \quad \forall w \in W, i_w \in I_w$

(4) $\tau_{t+1} \geq \tau_t + \epsilon$

(5) $\sum_{w \in W} \sum_{i_w \in I_w} r_{i_w} \cdot x_{i_w,v_k,t} \leq s_{k_v} \cdot y_{k_v,t} \; \forall v \in V, k_v \in K_v$

(6) $\sum_{k_v \in K_v} y_{k_v,t} \leq \gamma_{v,t} \; \forall v \in V$

(7) $y_{k_v,t} \cdot s_{k_v} - \sum_{w \in W} \sum_{i_w \in I_w} r_i \cdot x_{i_w,k_v,t} \leq f_{k_v} \; \forall v \in V, k_v \in K_v$

(8) $x_{i_w,k_v,t} = 1 \; \forall w \in W, i_w \in I_w, v \in V, k_v \in K_v \mid i_w$ runs

Thus, we assume a maximum number $k_v^{\#}$ of leasable VMs. The set of leasable VM instances of type v is indicated by K_v, where $k_v \in K_v = \{1, \dots, k_v^{\#}\}$. The cost for leasing one VM instance of type v is indicated by c_v.

For finally deciding at period t which service to instantiate and invoke, binary decision variables $x_{i_w,k_v,t} \in \{0,1\}$ are used. A value $x_{i_w,k_v,t} = 1$ indicates that the next service $j^*_{i_w}$ of workflow instance i_w should be allocated and invoked in period t at VM k_v, whereas a value $x_{i_w,k_v,t} = 0$ indicates that the invocation of the corresponding service should be delayed, i.e., no service of workflow instance i_w should be invoked in period t. For indicating, whether a certain instance k_v of VM type v should be leased in period t, another decision variable $y_{k_v,t} \in \{0,1\}$ is used. Similar to $x_{i_w,k_v,t}$, a value $y_{k_v,t} = 1$ indicates that instance k_v of VM type v is leased. The total number of VMs of type v to lease in period t is labeled with $\gamma_{v,t}$.

Using these parameters and variables, we formulate the optimization problem in Model 1 for deciding which workflow instances i_w and corresponding services j_{i_w}, respectively, should be allocated and invoked in period t.

The constraints in (2) make sure that the deadlines d_{i_w} for workflow instances i_w will not be violated. For this, the sum of the remaining execution time e_{i_w} and the next optimization point in time τ_{t+1} has to be lower or equal to the deadline. By scheduling a service invocation, the corresponding remaining execution time e_{i_w} is reduced, because the execution time for the next service $e_{j^*_{i_w}}$ will be subtracted in this case.

The constraints in (3) and (4) determine the next optimization point τ_{t+1}. In order to avoid optimization deadlocks, which would result from not advancing the next optimiza-

tion time τ_{t+1}, τ_{t+1} is restricted in (4) to be greater or equal to τ_t plus a small value $\epsilon > 0$. In order to replan the scheduling and invocation of workflows instances as soon as a service invocation has been finished, τ_{t+1} should be lower or equal to τ_{t+1} plus the minimum execution time of the services invoked in period t. Using an additional parameter $M \geq \max\left(e_{j_{i_w}}\right)$, the last term in (3) thereby makes sure that services that are not invoked in this period do not restrict τ_{t+1}, to be lower or equal to τ_t. However, a replanning will anyhow be triggered if events such as the request of further workflow instances or the accomplishment of a certain invoked service occur.

The constraints in (5) make sure that the resource capacities required by the *next* services for workflow instances i_w assigned to instance k_v of VM type v are lower or equal to the capacities these VM instances can offer. If processes are assigned to a certain instance k_v of type v, the corresponding decision variable $y_{k_v,t}$ will assume a value of 1. The sum of all decision variables $y_{k_v,t}$ determines the total number $\gamma_{v,t}$ of instances for VMs of type v, as indicated in (6). In (7), the amount of *unused capacities* for VM instance k_v, which is indicated by the variable f_{k_v}, is determined. In order to account for running service instances invoked in previous periods, corresponding decision variables are set to 1, which is indicated in (8).

The objective function, which is specified in (1), aims at minimizing the total cost for leasing VMs. In addition, by adding the amount of unused capacities f_{k_v} of leased VMs to the total cost, the objective function also aims at minimizing unused capacities of leased VM instances.

3.3 Heuristic Solution Approach

Due to its formulation as integer linear program, the solution approach from the previous section features relatively high computational complexity, which renders its applicability to realistic, large-scale scenarios difficult. Hence, for efficiently solving the optimization problem presented in the last subsection, we develop a heuristic solution method.

This heuristic basically examines which workflow instances i_w and services j_{i_w}, respectively, have to be scheduled and invoked in the current optimization period t in order to avoid violating corresponding deadlines d_{i_w}. For this, the heuristic initially determines the point in time τ_{t+1}, where the next optimization step has to be carried out – at the latest. Corresponding to this (latest) subsequent optimization time, a virtual *time buffer* is calculated for each workflow instance, which will be referred to as *slack*, indicating the time, the corresponding instance i_w may be delayed at maximum, before a violation of the corresponding deadline d_w takes place. Those instances, for which the slack is lower than *0*, i.e., those instances, for which a further delay would result in violating a workflow deadline, are considered as *critical*. Thus, at first, we try to invoke the critical

ALGORITHM 1: HEURISTIC SOLUTION APPROACH

```
1: //Initialize variables
2: d[w,i]; //Deadline for instance i of
   workflow w
3: e[w,i]; //Remaining execution time for
   instance i of workflow w
4: s[v]; //Resource supply for VM of type v
5: k[v]; //Number of already leased VMs of
   type v
6: leasedVM[v,k]; //k^th VM instance of type
   v
7: unusedRes[v,k]; //Unused resources for
   k^th VM of type v
8: sl[w,i] := new Double[w#,i#]; //Array for
   slack
9: r_crit := 0; //Aggregated resource demand
   for critical instances
10: vmTypList; //List of available VM types,
   sorted by size ascending
11: sortList := new List(); //Sorted List
   corresponding to slack
12: critList := new List(); //List
   containing critical instances
13: τ_t+1 = d[1,1]; //Initialize next
   optimization point in time
14: //Compute τ_t+1
15: for (w=1;w≤w#;w=w+1) do
16:   for (i=1;i≤i#;i=i+1) do
17:     if d[w,i]-e[w,i]-τ_t+1 then
18:       τ_t+1=d[w,i]-e[w,i];//Get minimum τ_t+1
19:     end if
20:     if τ_t+1≤τ_t then
21:       τ_t+1=τ_t+ε;//Avoid deadlocks
22:     end if
23:   end for
24: end for
25: //Compute slack sl_iw
26: for (w=1;w≤w#;w=w+1) do
27:   for (i=1;i≤i#;i=i+1) do
28:     sl[w,i]=d[w,i]-e[w,i]-τ_t+1;//Get slack
29:     if sl[w,i]<0 then
30:       critList.add(getInst(w,i));
31:       r_crit=r_crit+getInst(w,i).
              resNextService();
32:     else
33:       sortList.
              insert(getInst(w,i),sl[w,i]);
34:     end if
35:   end for
36: end for
37: //Invoke critical instances on leased
   but unused resources
38: usedRes=placeOnUnusedRes(critList);
39: r_crit= r_crit-usedRes;
40: //lease new VMs until r_crit is satisfied
41: leaseNewVMs(r_crit);
42: //Place further non-critical instances
   on unused resources
43: placeOnUnusedRes(sortList);
```

services instances on such VM instances that are already leased and running. Afterwards, we lease new VM instances such that all remaining critical service instances are allocated and invoked in the current period. Finally, in order to minimize unused resources of leased VM instances, we invoke service instances on the leased VM instances corresponding to their slack.

This heuristic solution method is provided in Algorithm 1 using pseudocode. Corresponding methods used in Algorithm 1 are indicated in Algorithm 2, Algorithm 3 and Algorithm 4. In lines 1-13 of Algorithm 1, the required parameters and variables are initialized. For instance, the deadlines d_{i_w} for workflow instances i_w of workflow template w are initialized in line 2. In this respect, it has to be noted that the corresponding parameter $d[w,i]$ represents an array containing all deadlines d_{i_w} of workflow instances i_w that are considered at optimization period t. Analogously, $e[w,i]$ represents an array for the *remaining* execution times of workflow instances i_w, which is initialized in line 3. The resource supply s_v for a VM of type v is initialized in line 4, whereas the number of instances k_v of leased VMs of type v is initialized in line 5. Note that VM instances potentially have been leased in previous periods. Thus, the number of instances k_v is not necessarily 0 when carrying out an optimization step.

Correspondingly, leased VMs as well as unused resources of already leased VMs, which are indicated by the arrays $leasedVM[v,k]$ and $unusedRes[v,k]$, have to be considered (cf. lines 6-7). For computing the slack of all workflow instances and aggregating the resource demands of the critical instances, the array $sl[w,i]$ and the variable r_{crit} is used (cf. lines 8-9). Since in this heuristic, different sizes of VMs are considered, i.e., they differ in the amount of available resources, a list of the available VM types is stored in $vmTypList$ in line 10. This list is sorted in ascending order by the VMs' sizes, i.e., the smallest VM comes first. In lines 11-12, empty lists are created for storing the critical instances as well as the non-critical instances, which are sorted in the list $sortList$ corresponding to their slack. Finally, the point in time τ_{t+1}, where the next optimization step has to be carried out at latest, is initialized with an arbitrary deadline, as, e.g., d_{1_1}.

Having initialized required parameters and variables, τ_{t+1} is determined in lines 15-24 by computing the minimum difference between deadlines d_{i_w} and the remaining execution times e_{i_w} for all workflow instances. For this point in time, the difference between remaining execution time and deadline, i.e., the slack, will be 0 for at least one workflow instance. In order to avoid deadlocks that will result if the subsequent optimization time τ_{t+1} is equal to the current time τ_{t+1}, a small value $\epsilon > 0$ is added (cf. lines 20-21).

In lines 26-36, the slack for each workflow instance is computed (cf. line 28) and the corresponding workflow instances are either added to the list of critical instances (cf. line 30) or inserted into a sorted list of (non-critical) instances (cf. line 33) corresponding to their slack. In addition, the resource requirements for the next services of the critical workflow instances are aggregated (cf. line 31).

The corresponding critical service instances need to be allocated and invoked in the current period – either on already leased and running VM instances or on further VM instances that have to be additionally leased in this period. Invoking critical service instances on already leased VM instances is accounted for in line 38 by calling the method $placeOnUnusedRes$, which is provided in Algorithm 2. Within Algorithm 2, the method $placeInst$ is called, which is provided in Algorithm 3.

ALGORITHM 2: METHOD PLACEONUNUSEDRES(LIST)

```
1: //Variable Initialization
2: usedRes = 0;
3: removeList = new List();
4: for
   (iter=1;iter≤list.size();iter=iter+1) do
5:   inst = list.get(iter);
6:   r = inst.resNextService();
7:   placed = false;
8:   for (v=1;v≤v#;v=v+1) do
9:     if (!placed) then
10:      for (k=k[v];k≥1;k=k-1) do
11:        if (r≤unusedRes[v,k]) then
12:          placeInst(inst,leasedVM[v,k]);
13:          placed = true;
14:          unusedRes[v,k]=unused[v,k]-r;
15:          usedRes = unused[v,k]-r;
16:          removeList.add(inst);
17:          break;
18:        end if
19:      end for
20:    end if
21:  end for
22:end for
23:list.remove(removeList);
24:return usedRes;
```

ALGORITHM 3: METHOD PLACEINST(LIST,VM,K)

```
1: //Variable Initialization
2: v = VM.getType();
3: removeList = new List();
4: unusedRes = supply[v];
5: for (iter=1; iter≤list.size();
   iter=iter+1) do
6:   inst = list.get(iter);
7:   r = inst.resNextService();
8:   if (unusedRes≥r) then
9:     placeInst(inst,leasedVM[v,k]);
10:    unusedRes=unusedRes-r;
11:    removeList.add(inst);
12:  end if
13:end for
14:list.remove(removeList);
15:return supply[v]-unusedRes;
```

ALGORITHM 4: METHOD LEASENEWVMS(RES)

```
1: //Variable Initialization
2: vm; //Temp variable for new VM
3: while (r_crit>0) do
4:   for (iter=1; iter≤vmTypList.size();
   iter=iter+1) do
5:     v := vmTypList.get(iter);
6:     if (supply[v]≥r_crit OR
   iter==vmTypList.size()) then
7:       //start a new VM of type v
8:       vm = leaseVM(v);
9:       k[v] = k[v]+1;
10:      leasedVM[v,k[v]-1] = vm;
11:      unusedRes[vm,k[v]-1] = supply[v];
12:      usedRes = placeInst(critList,vm,
   k[v]-1);
13:      r_crit = r_crit-usedRes;
14:      break;
15:    end if
16:  end for
17: end while
```

In line 39, the resource requirements of the successfully invoked critical instances are subtracted from the aggregated resource demand of the remaining critical service instances. In line 41, additionally required resources are acquired. For this, the method *leaseNewVMs* (Algorithm 4) is called.

The types of the new VMs are chosen according to the following procedure: In general we successively try to acquire VMs with rather high resource supply aiming at reducing unused resource capacities due to having a large number of rather small-sized VMs. In addition, the cost of VMs in our scenario is proportional (see Section 2.4), i.e., larger VMs will provide a proportionately lower basic load that is not available to service instances. Therefore, as long as more resources are required, i.e., $r_{crit} > 0$ (cf. line 3 in Algorithm 4), a new VM having the least amount of provided resources, but still bigger or equal to the required resource demand r_{crit} (cf. line 6), will be leased. If no VM type fulfills this requirement, the biggest available VM type will be leased (cf. line 7, i.e., if the end of vmTypList is reached). Subsequent to that, a new VM having this type will be leased (cf. line 8). If a new VM is leased, the resource demand r_{crit} will be reduced by the amount of actually used resources, i.e., after having placed critical instances on this VM (cf. line 12-13). Before this, we update in lines 9-11 the number of leased VM instances of type v, i.e., k_v, store the corresponding VM instance, and set the value of its unused resource supply to the (maximum) supply of a VM of type v. Using the method *placeInst*, which is provided in Algorithm 3, we allocate critical service instances on the newly leased VM such that no further critical service instances can be placed on it due to its limited resource capacity. Subsequently, a "break" statement will stop the *for* loop and the procedure will be repeated until enough resources are available.

Corresponding to the presented algorithm for leasing additional resources (Algorithm 4), it is possible that resources are still available on the leased VMs , i.e., they are not fully utilized. Therefore, we invoke further scheduled service instances on (already) leased VM instances in order to reduce the amount of unused VM resources. This is realized in line 43 of Algorithm 1 by calling the method *placeOnUnusedRes* (cf. Algorithm 2) another time.

Based on the results of the algorithms, the Reasoner is able to lease/release resources corresponding to the calculated resource demand. In addition, the Workflow Manager gets the information at which point in time to invoke which service instance as part of which workflow instance.

4. EVALUATION

The Vienna Platform for Elastic Processes is a purely Java-based framework and was developed and tested in a Unix-based environment. The evaluation was done in a private Cloud running the OpenStack operating system. The individual services are deployed in an Apache Tomcat-based Application Server. In the following, we present our General Evaluation Approach (Section 4.1), i.e., the evaluation scenario including the evaluation criteria. The experiment's results are presented in Section 4.2.

4.1 GENERAL EVALUATION APPROACH

While ViePEP and the presented reasoning approach are applicable in arbitrary process landscapes and industries, we evaluate the heuristic using a data analysis process from the finance industry. Choosing this particular process does not restrict the portability of our approach. We apply a testbed-driven evaluation approach, i.e., real Cloud resources are used. For the individual services, we simulate differing workloads regarding CPU and RAM utilization and service execution time (see below). However, real services are deployed and invoked during workflow executions.

To simplify an interpretation of the chosen evaluation settings, we decided to make use of one single workflow which will be processed 20,000 times. This sequential workflow consists out of five individual service steps: The lightweight *Dataloader Service* simulates the loading of data from an arbitrary source; afterwards, the more resource-intensive *Pre-Processing Service* is invoked; next, the *Calculation Service* simulates data processing, which leads to high CPU load; then, the *Reporting Service* generates a simple report – it generates a load similar to the one of the Pre-Processing Service. Last, the *Mailing Service* sends the report to different recipients – this is a lightweight service comparable to the Dataloader Service. The user-defined maximum execution time for workflow instances has been set to 5 minutes, commencing with the request.

In order to test our optimization approach against a baseline, we have implemented a very basic ad hoc approach. As the name implies, this approach is only able to take into account currently incoming workflow requests in an ad hoc way. While this includes the scheduling of workflow requests, the baseline approach does not take into account

future resource demands. Instead, whenever a Backend VM is utilized more than 80%, an additional single-core Backend VM for the according service is leased. When the utilization is below 20%, the VM is released again. Notably, the baseline approach will only lease single-core VMs, as it is not able to take into account future resource demands.

Arrival Patterns We make use of two distinct workflow request arrival patterns: In the *Constant Arrival* pattern, the workflow requests arrive in a constant manner. This means, the same amount of workflows arrives in a regular interval. In our evaluation, the number of simultaneously executed workflows is set to 200 and is send to ViePEP every 20 seconds. In the *Linear Arrival* pattern, the workflows are executed following a linear rising function, i.e., $y = k * \left\lfloor \frac{x}{3} \right\rfloor + 40$, where y is the amount of concurrent requests and 40 is the start value. This value is increased by $k = 40$ at an interval of 60 seconds.

Metrics In order to get reliable numbers, we executed each arrival pattern three times and evaluated the results against three quantitative metrics. First, we measure the overall execution duration which is needed to process all 20,000 workflow requests (*Duration in Minutes*). This is the timespan from the arrival of the first workflow request until the last step of the last workflow instance has been processed successfully. The second metric is the amount of concurrently leased number of cores, i.e., the sum of (CPU) cores of the leased VMs (*Active Cores).* The combination of the first two metrics, results in *Cost in Core-Minutes*, i.e., these tell us the resulting cost of the overall evaluation. The Core-Minutes are calculated following a similar pricing schema as Amazons EC2, i.e., the VMs cost increase proportionally with the number of provided resources. Our evaluation environment, i.e., the private Cloud we are running ViePEP in, provides four different VM types with 1-4 cores respectively In order to get the resulting cost, we sum up the active cores over time and get the overall Core-Minutes.

4.2 Results and Discussion

Table 1 and Figures 3-4 present our evaluation results in terms of the average numbers from the conducted evaluation runs. Table 1 presents the observed metrics as discussed in the last section for both arrival patterns. For each pattern, the numbers for evaluation runs are given for the baseline algorithm as well as the deployed optimization approach. The table also states the standard deviation for each metric. In general, the observed standard deviation is low, and therefore indicates a low dispersion in the results of the evaluation runs. Figures 3-4 complete the presentation of the average evaluation results by depicting the arrival patterns over time and the number of active cores until all workflow requests have been served. To combine numbers from different evaluation runs, we apply nearest-neighbor interpolation to the next full minute.

The numbers in Table 1 indicate a substantial performance difference between the baseline and the optimization approach. Most importantly, the cost in terms of Core-Minutes is lower in both cases, leading to almost 16.5% cost savings for the Constant Arrival pattern and 22.6% for the Linear Arrival pattern. Hence, we can deduce that the optimization approach helps to achieve a significantly better utilization of VMs, thus preventing additional cost arising from overprovisioning of Cloud-based computational resources. Also, the optimization approach is faster in absolute numbers, as it needs 25% less time to execute all workflow requests in the Constant Arrival pattern and 22.3% in the Linear Arrival pattern.

For both arrival patterns, the baseline approach is in many cases not able to comply with the workflow deadlines (5 minutes), as can be seen from the backlog after all workflow requests have arrived. This can be traced back to the applied ad hoc approach, i.e., it takes the baseline approach too long to react to new workflow requests and adjust the number of leased VMs correspondingly.

TABLE 1: EVALUATION RESULTS

	Constant Arrival		Linear Arrival	
	Baseline	Reasoner + Scheduler	Baseline	Reasoner + Scheduler
Number of Workflow Requests	20,000			
Interval between two Request Bursts (in Seconds)	20		20	
Number of Requests in one Burst	200		$y = 40 * \left\lfloor \frac{x}{3} \right\rfloor + 40$	
Duration in Minutes (Standard Deviation)	52 (σ = 2.16)	39 (σ = 0.81)	28.67 (σ = 1.25)	22 (σ = 0.82)
Max. Active Cores (Standard Deviation)	11 (σ = 0)	10 (σ = 0)	18 (σ = 0)	16.66 (σ = 0.47)
Cost in Core-Minutes (Standard Deviation)	443.67 (σ = 7.72)	370.33 (σ = 5.90)	314.71 (σ = 25.36)	243.59 (σ = 6.70)

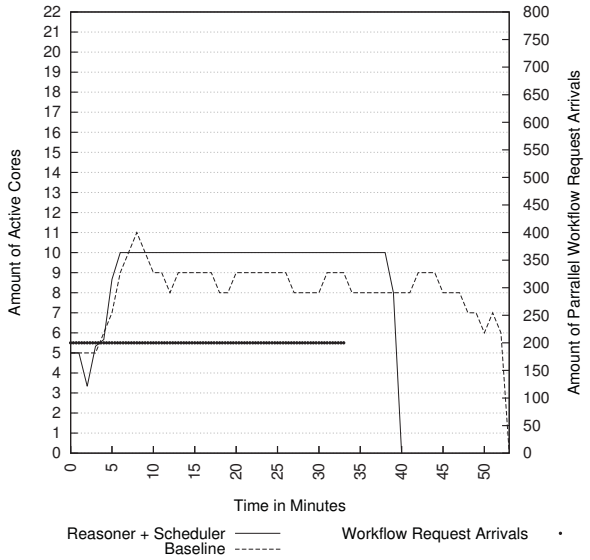

FIGURE 3: CONSTANT ARRIVAL RESULTS

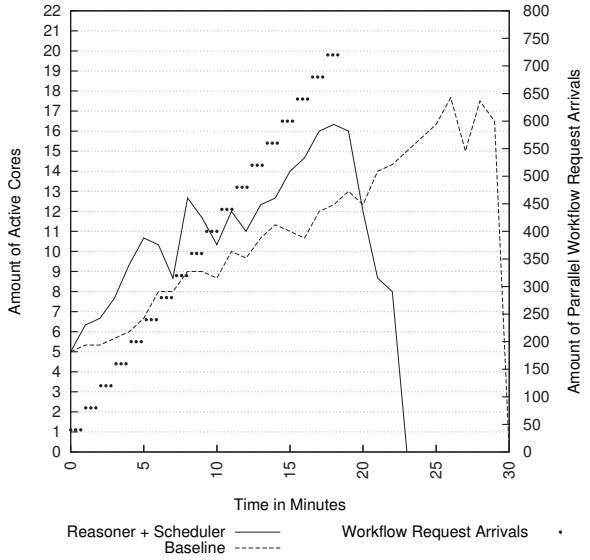

FIGURE 4: LINEAR ARRIVAL RESULTS

Interestingly, for the Constant Arrival pattern, Figure 3 shows clearly that ViePEP was able to optimize the system's landscape almost perfectly, i.e., the number of active VMs vary only a few times during the experiment. It can be perfectly seen that the optimization approach (i.e., "Reasoner + Scheduler") is not only faster than the baseline, but also acquired less overall computing resources, i.e., VMs.

For the Linear Arrival pattern, the number of active cores increases quite similarly for the optimization and the baseline. However, the biggest difference is that in the "Reasoner + Scheduler" approach, VMs with more than one core are acquired while in the baseline approach only single-core VMs are acquired. This results in a slower processing of the whole workflow queue since the overhead of the operating system is comparable higher in a single-core VM than in a quad-core VM.

To summarize, the evaluation results show that the proposed optimization approach indeed leads to a more efficient allocation of computational resources. As a result, ViePEP is able to provide a higher cost-efficiency than approaches that do not take the process perspective into account – in our evaluation, such approaches where represented by the baseline. In addition, ViePEP is able to decrease the risk of under- and overprovisioning and therefore adds an important functionality to BPMS.

5. RELATED WORK

Research on the utilization of Cloud-based computational resources for the execution of business processes is still at its beginning (Dustdar et al., 2011; Andrikopoulos et al., 2013). To the best of our knowledge, the number of approaches is still very small, but nevertheless, there is related work from other fields of research which should be taken into account, i.e., resource allocation and service provisioning for single tasks (Section 5.1), for Scientific Workflows (Section 5.2), and for business processes (Section 5.3).

5.1 SINGLE TASKS

In the field of Cloud Computing, resource allocation and automated service provisioning is a major research challenge (Buyya et al., 2009), and many methods and algorithms to allocate or schedule single service requests in an ad hoc manner have been proposed in recent years. These approaches focus on different aspects, with cost optimization and resource utilization naturally being the most obvious ones. For instance, Lampe et al. (2011) define the Software Service Distribution Problem in order to appoint services on the Software as a Service (SaaS) level to particular VMs on the Infrastructure as a Service (IaaS) level. The authors make use of a Knapsack-based heuristic approach in order to solve the problem. Li and Venugopal (2011) provide mechanisms to automatically scale applications up and down on the IaaS level. For this, a reinforcement learning approach is followed, which learns the best server and application actions. QoS and SLA enforcement are also taken into account, e.g., by Buyya et al. (2010), who propose the federation of independent Cloud resources in order to deliver the needed QoS in a cost-efficient way, or by Cardellini et al. (2011), who model resource management in terms of VM allocation for services as a mixed integer linear optimization problem and propose heuristics to solve them. Wu et al. (2011) discuss dynamic resource allocation from the perspective of a SaaS provider, aiming at profit maximization. Scheduling of service requests is based on defined SLAs between the provider and its customers.

All approaches discussed so far lack a process perspective across utilized resources, but focus on the ad hoc allocation of Cloud resources for individual services and tasks.

5.2 Scientific Workflows

There have been several approaches to utilize Cloud resources for the execution of Scientific Workflows (SWFs), e.g., by Hoffa et al. (2008) or Juve and Deelman (2010). Pandey et al. (2010) propose the usage of Particle Swarm Optimization for scheduling SWFs on Cloud resources. The authors especially take into account the cost of data transmissions and storage cost and focus on the minimization of total cost. SLAs or QoS aspects are not taken into account. Szabo and Kroeger (2012) apply evolutionary algorithms in order to solve scheduling for data-intensive SWFs on a fixed number of VMs. Deadlines are not explicitly regarded and only one workflow is considered at a time. The latter constraint also applies to the works by Byun et al. (2011) and Abrishami et al. (2013), who both present resource allocation and scheduling approaches to optimize cost under a user-defined deadline. While these approaches offer interesting ideas and insights, there are certain differences between business processes and SWFs that prevent a direct adaptation of such approaches (Ludäscher et al., 2009).

5.3 Business Processes

Approaches which directly address business processes are still scarce, but recently, a number of researchers have started to present corresponding work: Xu et al. (2009) provide some basic assumptions for the work at hand, most importantly, that workflows are interdependent and share services. Optimization of scheduling is done with respect to cost and time, but SLAs are not taken into account. While not explicitly regarding business processes, Lee et al. (2010) allow the execution of applications composed from interdependent services running on different machines. The authors focus on maximizing the profit for an IaaS broker, who leases resources and provides VMs to service consumers.

Juhnke et al. (2011) provide an extension to a standard BPEL workflow engine, which allows making use of Cloud resources to execute business processes. As BPEL is applied, workflows are composed from services, which mirrors our approach. It is possible to execute several workflows in parallel and optimize their scheduling and the resource allocation with respect to cost and overall execution time; apart from the cost for VMs, data transfer cost are also taken into account. A genetic algorithm is applied to solve the optimization problem. However, workflow deadlines are not regarded. Hence, this approach makes use of a similar resource model as done by the SWF approaches discussed above. The same applies to the work by Bessai et al. (2013), who also assume that workflows are composed from single software services. The authors propose different methods to optimize resource allocation and scheduling, aiming at cost or time optimization or to find a pareto-optimal solution covering both cost and time. Tasks may be shared among concurrent workflows, but in contrast to our work, tasks will not share the same VM (and service instance) concurrently. Deadlines are also not regarded. As the discussed approaches do not regard deadlines, they are not able to optimize resource allocation through postponing particular workflows steps to future timeslots.

Wei and Blake (2013) and Wei et al. (2013) propose a similar approach – again, workflows are built from single services and the authors focus on resource allocation. While service instances may be part of different workflows (Wei et al., 2013), the authors do not allow for parallel service invocations in different workflows, i.e., one service instance can only be invoked by a particular workflow at a time. In contrast, we follow the "classic" service composition model, which allows exactly this. The authors do not take into account SLAs or workflow deadlines, but a workflow owner may define some generic QoS constraints (Wei et al., 2013). Since deadlines are not taken into account, the authors do not provide scheduling mechanisms. Cost areCosts are also not regarded explicitly, but mechanisms are presented which aim at saving cost. Because workflows are not able to concurrently share service instances, the potential for optimization of resource allocation is not completely exploited. Similar to Bessai et al. (2013), Wei et al. also do not implement a testbed to test their algorithms, but use simulation in their evaluation. Despite the differences between our work and the work by Wei et al., there are also some commonalities, e.g., to allow different sizes of VMs at proportional cost. Furthermore, Wei and Blake (2013) also discuss the usage of resource demand prediction as a prerequisite for resource allocation.

Janiesch et al. (2014) provide an extensive conceptual model for Elastic Processes and implement an corresponding testbed which makes use of Amazon Web Services. The authors take into account SLAs (including workflow deadlines) and cost optimization, but do not provide automatic scheduling and resource allocation methods yet. In contrast to our work, the authors do not make use of workflow monitoring data to derive resource demands for upcoming services, but assume that there is a correlation between the resource demands of different tasks in a workflow. Applying a complementary scenario to the work presented within this paper, Gambi and Pautasso (2013) define design principles for RESTful business processes executed using Cloud resources. However, the authors propose to place complete processes on the same VM instead of allowing distributing services which belong to different workflows on different VMs. Hence, it is not possible to share resources between workflows. Finally, Frincu et al. (2013) analyze the application of resource provisioning and scheduling approaches for Grid workflows to Cloud-based workflows.

6. Conclusion

Resource-intensive processes and their execution using workflow and service technologies play an increasingly important role in many industries. The usage of Cloud resources to allow the execution of such processes in an elastic way seems to be an obvious choice, but so far, BPMS do

lack the ability to lease and release Cloud resources and allocate them in order to execute workflows.

In this paper, we have presented the Vienna Platform for Elastic Processes, which combines the functionalities of a BPMS with that of a Cloud resource management system. We have also presented an extended optimization model and heuristic for workflow scheduling and resource allocation for Elastic Process execution. As has been shown in our evaluation, the optimization approach leads to significant cost saving and time savings

Research on Elastic Processes is just at the beginning. There are several research directions that should be pursued in the future. First of all, we would like to extend the basic model of our workflow scheduling and resource allocation approach by allowing several different service instances per VM, vertical and horizontal scaling of VMs, a more complex VM model (e.g., non-proportional cost for VMs, minimum lease periods for VMs from public Clouds), include data transfer cost when scheduling workflows, and explicitly taking into account more complex workflow patterns. Second, while ViePEP was conceptualized for usage in hybrid Clouds, we are currently running it within a private Cloud environment. In the future, we will extend it by making it possible to combine public and private Cloud resources. Third, while the evaluation provides important results, we consider it as preliminary. In our future work, we want to make use of a more realistic Elastic Process test collection; we will also provide this test collection to interested researchers. Last but not least, we are currently reengineering ViePEP in order to make it ready for distribution as Open Source software.

7. ACKNOWLEDGMENTS

This work is partially supported by the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201) and by the E-Finance Lab e.V., Frankfurt am Main, Germany (www.efinancelab.de).

This paper is an extended version of Hoenisch et al. (2013).

8. REFERENCES

Abrishami, S., Naghibzadeh, M., Epema, D.H.J. (2013). Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds, *Future Generation Computer Systems*, 29(1), 158-169.

Andrikopoulos, V., Binz, T., Leymann, F., Strauch, S. (2013). How to adapt applications for the Cloud environment – Challenges and solutions in migrating applications to the Cloud, *Computing*, 95(6), 493-535.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M. (2010). A View of Cloud Computing, *Communications of the ACM*, 53(4), 50-58.

Bessai, K., Youcef, S., Oulamara, A., Godart, C. (2013). Bi-criteria Strategies for Business Processes Scheduling in Cloud Environments with Fairness Metrics, *Proc. of IEEE 7th Intern. Conf. on Research Challenges in Information Science (RCIS 2013)*, Paris, France, 1-10.

Breu, R., Dustdar, S., Eder, J., Huemer, C., Kappel, G., Köpke, J., Langer, P., Mangler, J., Mendling, J., Neumann, G., Rinderle-Ma, S., Schulte, S., Sobernig, S., Weber. B. (2013). Towards Living Inter-Organizational Processes, *Proc. of the 15th IEEE Conf. on Business Informatics (CBI 2013)*, Vienna, Austria, 363-366.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, *Future Generation Computing Systems*, 25(6), 599-616.

Buyya, R., Ranjan, R., Calheiros, R. N. (2010). InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, *Proc. of 10th Intern. Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP 2010)*, Busan, Korea, 13-31.

Byun, E.-K., Kee, Y.-S., Kim, J.-S., Maeng, S. (2011). Cost optimized provisioning of elastic resources for application workflows, *Future Generation Computer Systems*, 27(8), 1011-1026.

Cardellini, V., Casalicchio, E., Lo Presti, F., Silvestri, L. (2011). SLA-aware Resource Management for Application Service Providers in the Cloud, *Proc. of the First Intern. Symposium on Network Cloud Computing and Applications (NCCA '11)*, Toulouse, France, 20-27.

Dustdar, S., Guo, Y., Satzger, B., Truong, H.-L. (2011). Principles of Elastic Processes, *IEEE Internet Computing*, 15(5), 66-71.

Dustdar., S., Schreiner, W. (2005). A survey on web services composition, *Intern. J. of Web and Grid Services*, 1(1), 1-30.

Frincu, M. E., Genaud, S., Gossa, J. (2013). Comparing Provisioning and Scheduling Strategies for Workflows on Clouds, *Proc. of the 2013 IEEE 27th Intern. Symposium on Parallel and Distributed Processing (IPDPS 2013) Works. and PhD Forum – 2nd Intern. Works. on Workflow Models, Systems, Services and Applications in the Cloud (CloudFlow 2013)*, Boston, MA, USA, 2101-2110.

Gambi, A., Pautasso, C. (2010). RESTful Business Process Management in the Cloud, *Proc. of the 5th Intern. Works. on Principles of Engineering Service-Oriented Systems (PESOS 2013) in conjunction with the 35th Intern. Conf. on Software Engineering (ICSE 2013)*, San Francisco, CA, USA, 1-10.

Gewald, H., Dibbern, J. (2009). Risks and benefits of business process outsourcing: A study of transaction services in the German banking industry, *Information & Management*, 46(4), 249-257.

Hoenisch, P., Schulte, S., Dustdar, S., Venugopal, S. (2013). Self-Adaptive Resource Allocation for Elastic Process Execution, *Proc. of IEEE 6th Intern. Conf. on Cloud Computing (CLOUD 2013)*, Santa Clara, CA, USA, 220-227.

Hoenisch, P., Schulte, S., Dustdar, S. (2013a). Workflow Scheduling and Resource Allocation for Cloud-based Execution of Elastic Processes (forthcoming), *Proc. of 6th IEEE Intern. Conf. on Service Oriented Computing and Applications (SOCA)*, Kauai, HI, USA; NN-NN.

Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J. (2008). On the Use of Cloud Computing for Scientific Workflows, *Proc. of IEEE Fourth Intern. Conf. on e-Science (e-Science'08)*, Indianapolis, IN, USA, 640-645.

Huang, Z, van der Aalst, W. M. P., Lu, X., Duan, H. (2011). Reinforcement learning based resource allocation in business process management, *Data & Knowledge Engineering*, 70(1), 127-145.

Janiesch, C., Weber, I., Menzel, M., Kuhlenkamp. J. (2014). Optimizing the Performance of Automated Business Processes Executed on Virtualized Resources (forthcoming), *Proc. of Hawaii Intern. Conf. on System Sciences (HICSS-47)*, Hawaii, USA, NN-NN.

Jin, T., Wang. J., Rosa, M. L., ter Hofstede, A. H. M., Wen, L. (2013). Efficient Querying of Large Process Model Repositories, *Computers in Industry*, 64(1), 41-49.

Juhnke, E., Dörnemann, T., Bock, D., Freisleben, B. (2011). Multi-objetive Scheduling of BPEL Workflows in Geographically Distributed Clouds, *Proc. of IEEE 4th Intern. Conf. on Cloud Computing (CLOUD 2011)*, Washington DC, USA, 412-419.

Juve, G., Deelman, E. (2010). *Scientific Workflows and Clouds*, ACM Crossroads, 16(3), 14-18.

Kephart, J. O., Chess, D. M. (2003). The Vision of Autonomic Computing, *Computer*, 36(1), 41-50.

Kühn, E., Mordinyi, R., Lang, M., Selimovic, A. (2009). Towards Zero-Delay Recovery of Agents in Production Automation Systems, *Proc. of 2009 IEEE/WIC/ACM Conf. on Intelligent Agent Technology (IAT 2009)*, Milano, Italy, 307-310.

Lampe, U., Mayer, T., Hiemer, J., Schuller, D., Steinmetz, R. (2011). Enabling Cost-Efficient Software Service Distribution in Infrastructure

Clouds at Run Time, *Proc. of 4th IEEE Intern. Conf. on Service-Oriented Computing and Applications (SOCA 2011)*, Irvine, CA, USA, 1-8.
Lee, Y.C., Wang, C., Zomaya, A. Y., Zhou, B. B. (2010). Profit-Driven Service Request Scheduling in Clouds, *Proc. of 10th IEEE/ACM Intern. Conf. on Cluster, Cloud and Grid Computing (CCGrid 2010)*, Melbourne, Australia, 15-24.
Li, H., Venugopal, S. (2011). Using Reinforcement Learning for Controlling an Elastic Web Application Hosting Platform, *Proc. of 8th Intern. Conf. on Autonomic Computing (ICAC 2011)*, Karlsruhe, Germany, 205-208.
Ludäscher, B., Weske, M., McPhilipps, T. M., Bowers, S. (2009). Scientific Workflows: Business as Usual? *Proc. of 7th Intern. Conf. on Business Process Management (BPM 2009)*, Ulm, Germany, 31-47.
Maurer, M., Brandic, I., Sakellariou, R. (2013). Adaptive resource configuration for Cloud infrastructure management, *Future Generation Computing Systems*, 29(2), 472-487.
Mutschler, B., Reichert, M., Bumiller, J. (2008). Unleashing the Effectiveness of Process-Oriented Information Systems: Problem Analysis, Critical Success Factors, and Implications, *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(3), 280-291.
Pandey, S., Wu, L, Guru, M, Buyya, R. (2010). A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments, *Proc. of 24th IEEE Intern. Conf. on Advanced Information Networking and Applications (AINA 2010)*, Perth, Australia, 400-407.
Pesic, M, van der Aalst, W. M. P. (2007). Modeling work distribution mechanisms using Colored Petri Nets, *Intern. J. on Software Tools for Technology Transfer*, 9(3-4), 327-352.
Rohjans, S., Dänekas, C., Uslar, M. (2012). Requirements for Smart Grid ICT Architectures, *Proc. of Third IEEE PES Innovative Smart Grid Technologies (ISGT) Europe Conf.*, Berlin, Germany, 1-8.
Schuller, D., Lampe, U., Eckert, J., Steinmetz, R., Schulte, S. (2012). Cost-driven Optimization of Complex Service-based Workflows for Stochastic QoS Parameters, *Proc. of 10th IEEE Intern. Conf. on Web Services (ICWS 2012)*, Honolulu, HI, USA, 66-74.
Schulte, S., Hoenisch, P., Venugopal, S., Dustdar, S. (2013). Introducing the Vienna Platform for Elastic Processes, *Proc. of Performance Assessment and Auditing in Service Computing Works. (PAASC 2012) at 10th Intern. Conf. on Service Oriented Computing (ICSOC 2012)*, Shanghai, China, 179-190.
Szabo, C., Kroeger, T. (2012). Evolving Multi-objective Strategies for Task Allocation of Scientific Workflows on Public Clouds, *Proc. of IEEE Congress on Evolutionary Computation (CEC 2012)*, Brisbane, Australia, 1-8.
Wei, Y., Blake, M. B. (2013). Decentralized Resource Coordination across Service Workflows in a Cloud Environment, *Proc. of 22nd IEEE Intern. Conf. on Collaboration Technologies and Infrastructures (WETICE 2013)*, Hammamet, Tunisia, 15-20.
Wei, Y., Blake, M. B., Saleh, I. (2013). Adaptive Resource Management for Service Workflows in Cloud Environments, *Proc. of the 2013 IEEE 27th Intern. Symposium on Parallel and Distributed Processing (IPDPS 2013) Works. and PhD Forum – 2nd Intern. Works. on Workflow Models, Systems, Services and Applications in the Cloud (CloudFlow 2013)*, Boston, MA, USA, NN-NN.
Wu, L., Garg, S. K., Buyya, R. (2011). SLA-based Resource Allocation for a Software-as-a-Service (SaaS) Provider in Cloud Computing Environments, *Proc. of 11th IEEE/ACM Intern. Symposium on Cluster, Cloud and Grid Computing (CCGRID 2011)*, Newport Beach, CA, USA, 195-204.
Xu, M., Cui, L., Wang, H., Bi, Y. (2009). A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing, *Proc. of 2009 IEEE Intern. Symposium on Parallel and Distributed Processing with Applications (ISPA 2009)*, Chengdu, China, 629-634.

Authors

Dr.-Ing. Stefan Schulte is a Postdoctoral Researcher at the Distributed Systems Group at Vienna University of Technology and the project manager of the ongoing EU FP7 project SIMPLI-CITY - The Road User Information System of the Future (http://www.simpli-city.eu). His research interests span the areas of SOA and Cloud Computing, with a special focus on QoS aspects.

Dr.-Ing. Dieter Schuller is a Postdoctoral Researcher at the Multimedia Communications Lab of Technische Universität Darmstadt, Germany. Conjointly with Ulrich Lampe, he leads the research area on "Service-oriented Computing". Dieter's research interests are in the areas of Service-oriented Computing, specifically on QoS and efficient service selection.

Dipl. Ing. Philipp Hoenisch is a first year PhD student at the Distributed Systems Group at Vienna University of Technology. Before starting his PhD, Philipp collected hands-on software developing experiences in several Open Source projects. His research interests cover the whole spectrum of Cloud computing, with the main focus on cost-efficient automatic scaling in order to provide a high QoS.

Dr.-Ing. Ulrich Lampe is a Postdoctoral Researcher at the Multimedia Communications Lab of Technische Universität Darmstadt, Germany. Conjointly with Dieter Schuller, he leads the research area on "Service-oriented Computing". Ulrich's research interests are in the areas of Service-oriented Computing and Cloud Computing, specifically on efficient software service distribution, auction-based capacity allocation, and Cloud-based multimedia services.

Prof. Dr.-Ing. Ralf Steinmetz is a professor in the Department of Electrical Engineering and Information Technology as well as in the Department of Computer Science at Technische Universität Darmstadt, Germany. Since 1996, he is managing director of the "Multimedia Communications Lab". He is the author and co-author of more than 750 publications. He has served as editor of various IEEE, ACM and other journals. He was awarded as Fellow of both the IEEE and the ACM.

Schahram Dustdar is a full professor of computer science with a focus on Internet technologies and heads the Distributed Systems Group at the Vienna University of Technology. He is an ACM Distinguished Scientist (2009) and recipient of the IBM Faculty award (2012). He is an Associate Editor of IEEE Transactions on Services Computing, ACM Transactions on the Web, and ACM Transactions on Internet Technology and on the editorial board of IEEE Internet Computing. He is the Editor-in-Chief of Computing (an SCI-ranked journal of Springer).