This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

Optimization of Complex Elastic Processes

Philipp Hoenisch, Dieter Schuller, Stefan Schulte, *Member, IEEE,* Christoph Hochreiner, Schahram Dustdar, *Senior Member, IEEE*

Abstract—Business Process Management is a matter of great importance in different industries and application areas. In many cases, it involves the execution of resource-intensive tasks in terms of computing power such as CPU and RAM. Due to the emergence of Cloud computing, theoretically unlimited resources can be used for the enactment of business processes. These Cloud resources render several challenges for Business Process Management Systems to ensure a predefined Quality of Service level during Cloud-based process enactment. Therefore, new solutions for process scheduling and resource allocation are required to tackle these challenges.

Within this paper, we present a novel approach to schedule business processes and optimize the used Cloud-based computational resources in a cost-efficient way, thus realizing so-called elastic processes. For that, we specify the Service Instance Placement Problem, i.e., an optimization model which defines the setting of how service instances are scheduled among resources. Through extensive evaluations we show the benefits of our contributions and compare the novel approach against a baseline which follows an ad hoc approach.

Index Terms—Cloud Computing, Elastic Processes, Optimization, Scheduling, Business Process Management

1 Introduction

Despite being a relatively new computing paradigm, Cloud computing has shown its potential to transform the way computing power is offered and consumed [1], [2]. With the market entrance of major players like Amazon or Google, this transformation is already pretty advanced [3]. Today, the worldwide public Cloud services market is worth 131 Billion US dollars, with an estimated compound annual growth rate of about 17% from 2011 to 2017 [4].

However, the impact of Cloud computing on Business Process Management (BPM) is still quite minor, both with regard to solutions offered by the software industry and research conducted in this area. One particular aspect where Cloud computing could provide several benefits for BPM is process enactment, i.e., the execution of business processes [5]: Process landscapes are ever-changing in terms of the number of process requests arriving. Therefore, the computational resources needed to enact process instances vary over time. These changes in workloads lead to peak resource demands as well as utilization gaps, if a fixed amount of computational resources is provided [6]. There is a risk of both computational resource *underprovisioning*, if the process landscape's demands cannot be fulfilled by the available resources, and *overprovisioning*, if sufficient resources are available but not utilized most of the time and therefore leading to unnecessary cost [2].

Leasing and releasing of Cloud-based resources is an obvious alternative to the provisioning of a fixed amount of computational resources. Business processes executed using Cloud resources are also known as *elastic processes* [7]. Elastic processes reflect three major benefits of Cloud computing [1], [2], [7]: (i) *Rapid elasticity* of single processes as well as the process landscape based on the actual demand for computational resources, (ii) leasing and releasing the needed computational resources in an *on-demand, utility-like fashion*, and (iii) pay-per-use of the resources through *metered service*.

In general, elastic processes are related to the idea of service composition, which is a prominent solution to enact business processes [8]. However, while in classic service composition these services are available on the Internet of Services, for elastic processes, these services are deployed on Cloud-based computational resources [5]. To realize elastic processes, it is necessary to enhance a Business Process Management System (BPMS) with the means to manage not only the process lifecycle, but also to act as a Cloud controller [9], i.e., to be able to lease and release Cloud resources, deploy services onto them (resulting in a service instance), and invoke the service instances following a process schedule. Process scheduling and resource allocation are based on the expected functionalities and non-functional demands of a process instance. Usually, constraints on non-functional attributes like Quality of Service (QoS) aspects, e.g., an expected deadline for process enactment, are defined in Service Level Agreements (SLAs). SLA violations may happen, which may lead to penalty cost [10]. There are several approaches to SLA enactment for single Cloud applications, e.g., [10], [11], [12], which optimize application executions with regard to cost minimization. However, there is a lack of optimization approaches for elastic processes [5].

In our former work, we have presented the *Vienna Platform for Elastic Processes* (ViePEP) [13], [14]. ViePEP is a research BPMS for elastic processes. In addition, we have provided optimization approaches for sequential elastic processes [15], [16]. However, more complex process patterns like XOR-blocks, AND-blocks or Repeat Loop orchestration components have not been covered yet, even though these

- *P. Hoenisch, S. Schulte, C. Hochreiner and S. Dustdar are with the Distributed Systems Group, Vienna University of Technology, Austria. Email: {p.hoenisch, s.schulte, c.hochreiner, dustdar}@infosys.tuwien.ac.at.*
- *D. Schuller is with the Multimedia Communications Lab (KOM), Technische Universität Darmstadt, Germany. Email: schuller@kom.tu-darmstadt.de.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

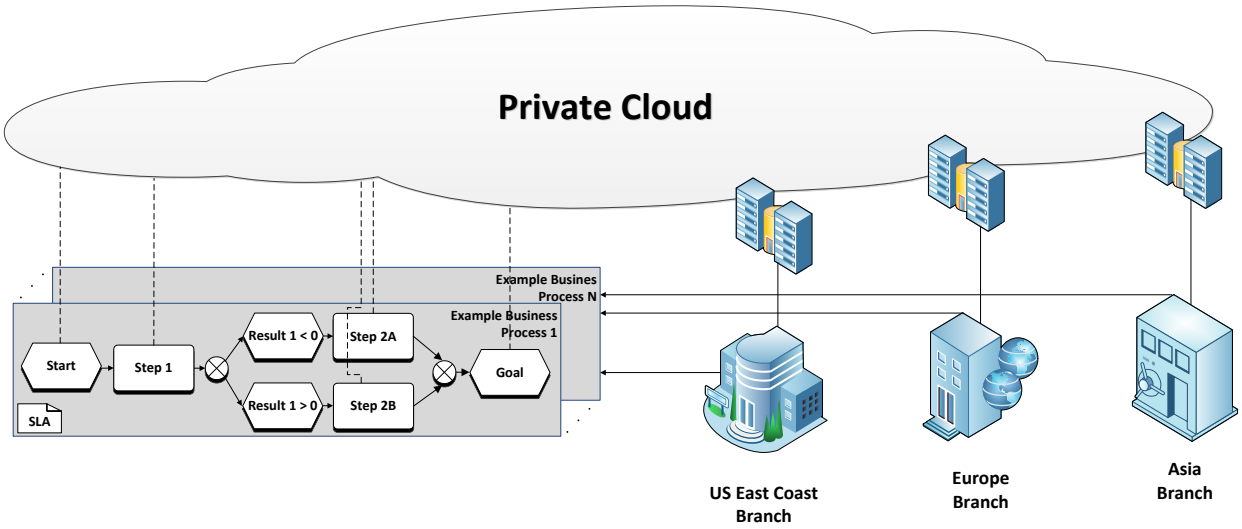


Fig. 1: Example Scenario

process patterns are very common in real-world business processes [17]. Also, despite being an important cost factor, penalty cost as well as the Billing Time Unit (BTU), which expresses the cost per leasing period, have not been regarded so far.

Hence, in this paper, we substantially extend our former work on elastic process scheduling and resource allocation. Our contributions can be summarized as follows:

- We define a system model for elastic process landscapes, taking into account complex process patterns, penalty cost, and BTUs.
- We present the *Service Instance Placement Problem* (SIPP), an optimization model which aims at minimizing the total cost arising from enacting an elastic process landscape. Solutions to the SIPP describe execution plans in terms of process scheduling and resource allocation. This includes the assignment of service instances to Virtual Machines (VMs), the scheduling of service invocations, and the leasing/releasing of VMs.
- We implement the SIPP and evaluate it extensively, showing its advantages compared to existing ad hoc resource allocation and process scheduling strategies.

The remainder of this paper is organized as following: While Section 2 introduces an example scenario motivating our work, Section 3 gives an overview of elastic processes and defines prerequisites for our approach. Afterwards, Section 4 presents the SIPP optimization problem. The results of the evaluation for the scheduling and resource allocation algorithm through testbed experiments are described in Section 5. While Section 6 discusses the related work, Section 7 concludes this paper and provides a short outlook on our future work.

2 Example Scenario

In the following paragraphs, we provide a basic example scenario to illustrate and motivate the work at hand. We consider a scenario from the financial industry, since in banks IT cost account for 15%-20% of the overall administrative expenses [18]. Hence, cost-efficient process enactment is an important goal in this domain. Notably, the presented example scenario is illustrative only and should neither be seen as complete nor exclusively for the applied banking domain. The work presented in this paper can be easily used in any domain which features an extensive process landscape and needs to be able to adjust efficiently to an ever-changing number of process requests, e.g., the manufacturing industry [19] or Smart Grids [20].

Figure 2 shows a graphical representation of our example scenario: We consider an international bank featuring several branches which are worldwide distributed from *US East Coast* over *Europe* to *Asia*. As each of the branches provides similar products to their customers, the bank maintains a private Cloud spanning all data centers of the distributed branches. This Cloud provides all business processes which are used within the bank's products and service.

Every branch of the bank has access to the business processes in the Cloud. Notably, the bank's branches have a large number of different business processes at their disposal. These may range from long-running data analytic processes to short-running trading processes. Especially in the latter kind of processes, time constraints are critical as even a short delay can lead to revenue loss or penalty payments. In order to ensure time constraints, a process is therefore equipped with a SLA which defines its deadline, i.e., the point of time at which the process enactment has to be finished.

Assuming the discussed bank simultaneously serves several thousands of costumers, a very large number of business processes with different priorities may be requested at every point of time. This leads to high fluctuations of needed resources during peak- and off-peak times. Also, the bank's process landscape is ever-changing – new orders and therefore process requests arrive, while running process instances finish or have to be repeated. Thus, the usage of principles of elastic processes is an obvious approach [7]. Not surprisingly, elasticity and scalability have

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

been named as primary reasons for the usage of Cloud-based computational resources in the financial industry [21]. For this, approaches to cost-efficient process scheduling and resource allocation – as discussed in the following – are needed.

3 Background

3.1 Elastic Processes

As mentioned above, for companies which feature an extensive and varying process landscape, moving their process enactments into the Cloud is an obvious choice. This is especially the case if a very large number of processes with time-critical priorities have to be handled simultaneously to adhere to SLAs and further to prevent penalties. For these situations, the advantage of Cloud computing provides a solution as it allows a higher throughput and reduces the risk of delays. This can be achieved through *resource* elasticity, which allows scaling computational resources up or down based on the current demand.

Although elasticity is usually regarded from the resources' point of view, two other main dimensions do also play an important role, namely *cost* and *quality* elasticity [22]. The former describes that the same Cloud service might be offered at different price levels, e.g., since these services are offered on spot markets, based on the amount of resources leased by a particular consumer, or because Cloud providers adapt their prices in relation to the current overall demand. Quality elasticity refers to the trade-off between QoS and cost. For example, SLA violations might be avoided by adding further Cloud-based computational resources, leading to better QoS as well as higher cost. In the work at hand, we will focus on resource and quality elasticity. We determine that Cloud services are available at fixed prices and therefore cost elasticity is not given.

With regard to our example scenario for the achievement of elastic processes (see Section 2), we assume that such processes can be composed from single software services running on VMs in the Cloud, each service representing a particular process step. In our example, these elastic processes represent the services provided by the bank. To enact elastic processes, a middleware layer is need, which provides Platform-as-a-Service (PaaS) functionalities to process requesters. Notably, the middleware needs to provide the functionalities of both, a Cloud controller and a BPMS. Hence, the middleware enacts elastic processes by controlling the Infrastructure-as-a-Service (IaaS) and the Software-as-a-Service (SaaS) levels [23]: At the IaaS level, it acts as a *resource allocator*, taking into account the current and future resource demand. At the SaaS level, it schedules and enacts service invocations, which together enact a process.

Resource allocation and scheduling affect each other in a way that service invocations may be assigned to resources which are not leased during the allocation planning, and therefore have to be leased in the near future (i.e., right before the actual invocation) in order to ensure flawless service invocations. Therefore, resource allocation and scheduling have to be considered equally during the planning phase for elastic process execution [7]. In this paper, we focus on scheduling (and implicitly on resource allocation) and provide a solution based on a Mixed Integer Linear Programming (MILP) problem.

3.2 Preliminaries

Having an example scenario discussed and the idea of elastic processes introduced, we define further preliminaries before modeling the SIPP: We assume that a middleware is deployed in the Cloud and serves as a BPMS and Cloud controller in an intra- or inter-organizational process landscape. Process owners (e.g., employees of the bank, see Section 2) may define *process models* and request their enactment, resulting in single *process instances*. Process models are composed of *process steps*. To execute a process step, a *service* is deployed on a particular VM resulting in a certain *service instance* and will then be *invoked* (*service invocation*). In the process landscape, we distinguish between several different services, i.e., each service instance is of a particular *service type*. The *execution time* of a *service invocation*, i.e., the time span from starting the invocation until it is finished, may take from a few seconds to several minutes. For practical reasons, a VM may only run a single service instance. However, this service instance may be invoked several times simultaneously and a service type may be deployed several times on different VMs. Along with the process request, process owners define deadlines, which are part of a SLA and therefore an important constraint for the SIPP. Penalties accrue if a certain process instance does not meet its SLA [10]. It is the general goal to minimize the cost of process enactments, taking into account VM leasing cost and penalty cost.

There might be several process owners in the process landscape, requesting different process instances at different points of time. As a result, the process landscape is volatile and ever-changing, which needs to be taken into account when solving the SIPP. Process landscapes could become very large, since the Cloud offers theoretically unlimited resources [24].

During scheduling, the middleware needs to take into account process requests, running process instances, leased VMs, service instances, the current and planned workloads of the leased VMs, as well as the execution times and workloads of different service types on different VMs. Importantly, the middleware is aware of *future* service invocations, since it knows the next steps of requested process instances.

We do not only consider simple, sequential processes, but also address more complex process patterns. Referring to [17], our scheduling approach accounts for structured processes comprising parallel invocations, i.e., *AND-blocks* (AND-splits with corresponding joins), as well as exclusive invocations, i.e., *XOR-blocks* (XOR-splits with corresponding joins). Also *Repeat Loops* that allow a repeated execution of sub-processes are covered. Considering these complex process patterns aggravates the scheduling problem substantially, since the *next* step is not always straight forward in XOR-blocks or Repeat Loops. For these, and for AND-blocks, several different possibilities have to be considered during resource allocation and scheduling. Applying the recursive pattern interlacing approach presented in our former work in [25], the scheduling approach proposed in this work is also capable of considering interlaced structures. Depending on the openness to risk, a worst-, best-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

or average-case has to be performed prior scheduling, i.e., taking the longest, the shortest or the average path through a process structure. In the work at hand, we consider a worst-case scenario.

4 Complex Process Scheduling

For the enactment of elastic processes, Cloud-based computational resources in terms of VMs are used. In this respect, we aim at achieving an optimal scheduling and placing of service instances, realizing corresponding process instances. For this reason, leasing and releasing of Cloud resources has to be realized so that the cost for leasing aforementioned Cloud resources is minimized. Furthermore, we need to make sure that given constraints on QoS attributes such as deadlines for the process instances are satisfied. In the work at hand, we exclusively focus on execution time as QoS attribute. Other QoS attributes from the field of service composition, e.g., availability, reliability or throughput [26], are not explicitly covered. It should be noted that availability and reliability are partially regarded in our optimization model, since process requests are carried out as long as there are computational resources from a Cloud provider available. Since Clouds (in theory) offer unlimited resources, the limiting factor for throughput is the missing capability of the middleware to handle an unlimited number of processes at the same time. Scalability of the middleware is however a research topic on its own.

If a violation of QoS constraints and therefore a SLA breach takes place, penalties accrue. Thus, the closer the deadline for a certain process instance is, the higher is the importance for scheduling and invoking corresponding service instances to enact the particular process instance in time. If not carefully considered and scheduled, the middleware provider will either have to lease and pay for additional VMs to host service invocations that cannot be delayed any further, or to pay the aforementioned penalties. To avoid such situations where extra resources have to be leased or penalties have to be paid due to an inefficient scheduling strategy, the scheduling of service invocations along with the leasing and releasing of Cloud resources has to be optimized. Hence, we formulate the problem of scheduling and placing service instances on VMs for realizing process instances – the *SIPP* – as a MILP optimization problem.

The applied system model is described in the subsequent Section 4.1. Afterwards, whereas a formal specification of the corresponding optimization model is provided in Section 4.2, the model is extended in Section 4.3 to enable multiperiod scheduling.

4.1 System Model

It has to be noted that the optimization problem described in this paper per se refers to a certain time period. For considering different time periods, we use the parameter t as index which indicates the start of a period[1]. The concrete *time period* is indicated by the parameter τ_t and is given by a concrete *point in time* (e.g., Tuesday, May 4^{th}, 13:37 CET) that corresponds to the beginning of that (optimization) time period. In order to account for different types of process instances, we consider multiple process models. The set of process models is labeled with P, where $p \in P = \{1, ..., p^{\#}\}$ indicates a certain process model. The set of process instances that have to be considered during a certain period t according to a certain process model p is indicated by I_p, where $i_p \in I_p = \{1, ..., i_p^{\#}\}$ refers to a certain process instance. In this respect, it has to be noted that *considering* a certain process instance i_p in period t does not necessarily result in invoking corresponding service instances in this period. It rather ensures that respective service invocations are acknowledged as potential candidates for scheduling. Thus, they *may* be scheduled in period t or not.

Service instances that have to be invoked for accomplishing a process instance i_p are covered in the set J_{i_p}, where $j_{i_p} \in J_{i_p} = \{1, ..., j_{i_p}^{\#}\}$ refers to a certain service invocation for accomplishing a specific step in process instance i_p. Service invocations that have already been scheduled in previous optimization periods are referred to as ${j_{i_p}}^{run}$. They are covered in the set ${J_{i_p}}^{run}$. Since the steps within a certain process instance have to be invoked in a certain order, we may only schedule service invocations for the *next* step(s) in optimization period t. We label the corresponding service invocations accomplishing the *next* step(s) with $j_{i_p}^{*} \in J_{i_p}$. The *execution time* of service invocation j_{i_p}, i.e., the duration of the service invocation, is indicated by $e_{j_{i_p}}$.

In order to schedule j_{i_p}, the corresponding service type has to be instantiated on a VM. In the work at hand, we account for different VM types. The set of VM types is indicated by the parameter V, where $v \in V = \{1, ..., v^{\#}\}$ refers to VM type v. The corresponding resource supply of a VM of type v in terms of processing units (CPU) and main memory (RAM) is indicated by s_v^C and s_v^R. The unit of CPU resources is *percent*, i.e., a single core VM has 100% CPU, a dual core VM has 200% CPU, and so on, and the unit for RAM is *Mega Bytes* (MB). Analogously, the resource demand of a certain service invocation j_{i_p} with respect to CPU in percent and RAM in MB is indicated by $r^C_{(j_{i_p},k_v)}$ and $r^R_{(j_{i_p},k_v)}$. An instance k of a VM of type v is referred to as k_v. Although in theory unlimited, we assume the number $k_v^{\#}$ of leasable VMs of type v to be limited in a certain time period t and specify the set of VM instances of type v as $K_v = \{1, ..., k_v^{\#}\}$, where $k_v \in K_v$. The cost for leasing one VM instance of type v is indicated by c_v. The remaining, *free* resource capacity of VM instance k_v regarding CPU and RAM after scheduling and placing of service invocations is referred to as $f^C_{k_v}$ and $f^R_{k_v}$.

As previously stated, a service has to be deployed on a VM in order to be invoked (resulting in a particular service instance). The corresponding *deployment time* depends on the type of the service, i.e., the *service type* which is referred to as st_j. The service deployment time is indicated by Δ_{st_j} and expressed in milliseconds. Parameter $z_{(st_j,k_v,t)} \in \{0, 1\}$ indicates if a service of type st_j is already deployed ($z_{(st_j,k_v,t)} = 1$) at VM instance k_v in the time period starting at t. In addition, the respective VM needs to be up and running in order to enable service deployment and invocation. The corresponding VM *start-up time* (in milliseconds) for a VM instance of type v is labeled with Δ_v,

1. For a concise overview of the parameters used in this paper, see the provided supplemental material.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

TABLE 1: Worst-Case Aggregation Specifications

Pattern	Execution time (e)
Sequence	$e^{seq} = \sum_{j \in J^{seq}} (e_j + \Delta_{st_j} + \Delta)$
AND-block	$e^{AND} = \max_{l \in L_a}(\sum_{j \in J^l} (e_j + \Delta_{st_j} + \Delta))$
XOR-block	$e^{XOR} = \max_{l \in L_x}(\sum_{j \in J^l} (e_j + \Delta_{st_j} + \Delta))$
Repeat Loop	$e^{RL} = re \cdot e^{seq}$

whereas $\Delta = \max_{v \in V}(\Delta_v)$ refers to the maximum start-up time. Whether the VM instance k_v is already up and running in period t is indicated by the parameter $\beta_{(k_v,t)} \in \{0,1\}$ – similar to $z_{(st_j,k_v,t)}$.

The *remaining* execution time in milliseconds for a process instance i_p is indicated by e_{i_p}. It can be computed by aggregating the execution times $e_{j_{i_p}}$ of service invocations j_{i_p} according to the structure of process instance i_p. As previously stated, we account for XOR-blocks, AND-blocks, and Repeat Loops in addition to Sequences. Corresponding aggregation specifications accounting for a worst-case analysis are provided in Table 1 – without an index for a specific process instance i_p. For the sake of simplicity, we assume the services' execution times not to be dependent on the concrete VM instance k_v. Accounting for VM-dependent service execution times could be achieved straightforwardly, since it does not affect our approach for computing optimal solutions to the SIPP. Only the way of computing the execution times would have to be adapted. Instead, we account for VM resources in terms of CPU and RAM as previously stated.

For a Sequence, the execution times of the respective service invocations have to be added up. In order to account for a worst-case analysis, we need to additionally consider the corresponding service deployment times Δ_{st_j} and the worst VM start-up time Δ, each expressed in milliseconds. With respect to an AND-block, it is necessary to account for the different *paths* l within the AND-block. The set of all paths is indicated by $L = \{1, ..., l^{\#}\}$, where l refers to a certain path. In order to separate the set of paths for AND-blocks from the set of paths for XOR-blocks, we use additional indices *a* (for AND) and *x* (for XOR). In order to obtain the execution time for an AND-block, we need to compute the execution times for each of the paths $l \in L_a$ separately and then take the maximum, which is indicated in Table 1. We thereby implicitly assume the steps within a path of the AND-block to be arranged sequentially, so that the corresponding execution time can be computed according to the aggregation specification for a Sequence. In order to account for interlaced structures, we apply a technique for recursively combining the provided aggregation specifications, which is described in our former work [25].

The aggregation specification for computing the execution time for an XOR-block is basically the same as for an AND-block – at least in terms of a worst-case analysis since we need to consider the *worst* path in terms of execution time. But in contrast to an AND-block, only one of the paths within an XOR-block will finally be invoked. With respect to Repeat Loops, the structure that has to be repeated is invoked multiple times. For a worst-case analysis, we assume a maximum number re of repeated invocations. With respect to Table 1, we consider a Sequence, indicated by e^{seq}, as the structure to be repeatedly invoked. However, also single process steps or whole processes comprising AND-/XOR-blocks could be subject for repeated invocations and, thus, for Repeat Loops.

The deadline DL_{i_p} indicates at which point in time the enactment of process instance i_p has to be finished. If DL_{i_p} is violated, penalty cost will accrue. The penalty cost depends on time and duration, respectively, the invocation of process instance i_p took longer than restricted by the deadline, indicated by $e^p_{i_p}$, and on the penalty cost per time unit, referred to as $c^p_{i_p}$.

For finally deciding whether to schedule a certain service invocation j_{i_p} in optimization period t, we use binary decision variables $x_{(j_{i_p},k_v,t)} \in \{0,1\}$. A value $x_{(j_{i_p},k_v,t)} = 1$ indicates that the service invocation j_{i_p} of process instance i_p should be scheduled in period t on the k-th VM of type v (and invoked afterwards), whereas a value $x_{(j_{i_p},k_v,t)} = 0$ indicates that the scheduling and invocations can be delayed. For indicating if we need to lease a certain VM instance k_v of type v in period t, the decision variable $y_{(k_v,t)} \in \mathbb{N}_0$ is used. In contrast to $x_{(j_{i_p},k_v,t)}$, $y_{(k_v,t)}$ may take values greater than 1 indicating that VM instance k_v should be leased in period t for $y_{(k_v,t)}$ BTUs. The BTU is the minimum leasing duration. Thus, releasing a VM before the end of the BTU corresponds to wasting paid resources. The actual leasing duration of a particular VM is always a multiple of the BTU. The *remaining* leasing duration for a specific VM instance k_v in period t is labeled with $d_{(k_v,t)}$. The total number of VMs of type v to lease in period t is indicated by $\gamma_{(v,t)}$.

Having described the underlying system model in this section, the subsequent sections present the optimization problem. As described, the optimization problem takes as input a set of process instances I_p including their process steps and corresponding service invocations J_{i_p}. In addition to that, a set of computational resources is needed (V).

4.2 Optimization Problem

In this section, we model the scheduling and placement of service invocations for enacting corresponding process instances – the SIPP – as an optimization problem. Using the system model provided in Section 4.1, we gradually develop the corresponding optimization model for the current optimization period t in this section. A multiperiod extension of the optimization problem is discussed in Section 4.3.

$$
\begin{aligned}
\min \; & \sum_{v \in V} c_v \cdot \gamma_{(v,t)} + \sum_{p \in P} \sum_{i_p \in I_p} c^p_{i_p} \cdot e^p_{i_p} \\
& + \sum_{v \in V} \sum_{k_v \in K_v} (\omega^C_f \cdot f^C_{k_v} + \omega^R_f \cdot f^R_{k_v}) \\
& - \sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in J^*_{i_p}} \frac{1}{DL_{i_p} - \tau_t} x_{(j_{i_p},k_v,t)}
\end{aligned} \tag{1}
$$

In (1), the objective function, which is subject for minimization, is depicted. It comprises four terms. In the first term, i.e., $\sum_{v \in V} c_v \cdot \gamma_{(v,t)}$, we compute the total cost accruing due to leasing $\gamma_{(v,t)}$ VM instances of type v at a cost of c_v per VM instance in period t. The second term, i.e., $\sum_{p \in P} \sum_{i_p \in I_p} c^p_{i_p} \cdot e^p_{i_p}$, accounts for penalties arising due to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

violating deadlines. For computing these penalties, a linear penalty function is applied [10]: We multiply the durations $e^p_{i_p}$, i.e., the time units the invocations of process instances i_p took longer than restricted by the corresponding deadline, with the cost per time unit, represented by $c^p_{i_p}$. The third term, i.e., $\sum_{v \in V} \sum_{k_v \in K_v} (\omega^C_f \cdot f^C_{k_v} + \omega^R_f \cdot f^R_{k_v})$, regards the sum of *free* resource capacities $f^C_{k_v}$ (CPU) and $f^R_{k_v}$ (RAM) for all (leased) VM instances – weighted with corresponding weights ω^C_f and ω^R_f. Finally, in the fourth term, we calculate the difference between the deadlines DL_{i_p} for all process instances and the current period τ_t and compute the corresponding reciprocal value. This way, we deduce a measure for the *urgency* and *importance*, respectively, for each process instance, since the closer the deadline is for a process instance i_p, the larger is $\frac{1}{DL_{i_p} - \tau_t}$.

With the first term, we aim at minimizing the total cost. With the second term, we aim at minimizing penalty cost. With the third term, we aim at minimizing leased but unused Cloud resource capacities. With the fourth term, we aim at *maximizing* (note the minus in front of the term) the relative importance of the scheduled service invocations. Since the minimization of the leasing cost and the penalty cost should have highest priority, we set the weights ω^C_f and ω^R_f to a very low value such as 0.000001. Since the actual values for the deadlines DL_{i_p} and the current period τ_t are quite large (as they are represented as the time elapsed since $01/01/1970$ in milliseconds), their difference remains large, so that the reciprocal value becomes rather small. Thus, the minimization of leasing cost and penalty cost receives the highest (relative) weights and thereby priorities within the objective function.

The constraints in (2) demand the deadlines DL_{i_p} not to be violated for all $p \in P$, $i_p \in I_p$, $j_{i_p} \in J^{run}_{i_p}$. For this, the sum of the *remaining* execution time e_{i_p} and the next optimization period starting at τ_{t+1} has to be lower or equal with respect to the deadline. The length of the current optimization period, consequently, the start of the next optimization is defined by τ_{t+1} and is restricted in (3). τ_{t+1} has to be greater or equal to the current optimization period at τ_t plus a small value $\epsilon > 0$. ϵ is needed to avoid optimization deadlocks resulting from a too small or negative value for τ_{t+1}.

$$\tau_{t+1} + e_{i_p} + e^{run}_{j_{i_p}} \leq DL_{i_p} + e^p_{i_p} \tag{2}$$

$$\tau_{t+1} \geq \tau_t + \epsilon \tag{3}$$

The remaining execution time e_{i_p} is computed in (4) for all $p \in P$, $i_p \in I_p$. Depending on the structures of the process instances, different remaining execution times apply, i.e., for sequences using (5), for AND-blocks (6), for XOR-blocks (7) and for Repeat Loops (8). Notably, a process instance may consist of a combination of different process patterns, which have to be summed up (see (4)).

As indicated in (5)-(8), the remaining execution time e_{i_p} can be reduced if a service invocation j_{i_p} is scheduled on a VM instance k_v. The helper variables $e^s_{i_p}$ and $e^l_{i_p}$ are defined in (10) and (11). As described in (9), in this case, the corresponding execution time $e_{j^*_{i_p}}$ along with the service deployment time $\Delta_{j^*_{i_p}}$ and VM start-up time Δ will be added up, resulting in the variable $ex_{j^*_{i_p}}$ and subtracted from the remaining execution time e_{i_p}. Note that execution times $e_{j_{i_p}}$ for service invocations that have already been scheduled in previous optimization periods will be set to zero. Further, since steps within AND-blocks may be invoked in parallel, it is possible to schedule multiple *next* service invocations – one *next* service invocation per branch of the AND-block.

$$e_{i_p} = e^{seq}_{i_p} + e^{L_a}_{i_p} + e^{L_x}_{i_p} + e^{RL}_{i_p} \tag{4}$$

$$e^{seq}_{i_p} = \begin{cases} \hat{e}^s_{i_p} - ex_{j^*_{i_p}} & \text{, if } x_{(j^*_{i_p}, k_v, t)} = 1 \\ \hat{e}^s_{i_p} & \text{, else} \end{cases} \tag{5}$$

$$e^{L_a}_{i_p} = \begin{cases} \max_{l \in L_a}(\hat{e}^l_{i_p} - ex_{j^*_{i_p}}) & \text{, if } x_{(j^*_{i_p}, k_v, t)} \\ \max_{l \in L_a}(\hat{e}^l_{i_p}) & \text{, else} \end{cases} \tag{6}$$

$$e^{L_x}_{i_p} = \begin{cases} \max_{l \in L_x}(\hat{e}^l_{i_p} - ex_{j^*_{i_p}}) & \text{, if } x_{(j^*_{i_p}, k_v, t)} \\ \max_{l \in L_x}(\hat{e}^l_{i_p}) & \text{, else} \end{cases} \tag{7}$$

$$e^{RL}_{i_p} = \begin{cases} re \cdot \hat{e}^s_{i_p} - ex_{j^*_{i_p}} & \text{, if } x_{(j^*_{i_p}, k_v, t)} \\ re \cdot \hat{e}^s_{i_p} & \text{, else} \end{cases} \tag{8}$$

$$ex_{j^*_{i_p}} = \sum_{v \in V} \sum_{k \in K_v} ((e_{j^*_{i_p}} + \Delta_{j^*_{i_p}} + \Delta) x_{(j^*_{i_p}, k_v, t)}) \tag{9}$$

$$\hat{e}^s_{i_p} = \sum_{j_{i_p} \in J^{seq}_{i_p}} (e_{j_{i_p}} + \Delta_{j_{i_p}} + \Delta) \tag{10}$$

$$\hat{e}^l_{i_p} = \sum_{j_{i_p} \in J^l_{i_p}} (e_{j_{i_p}} + \Delta_{j_{i_p}} + \Delta) \tag{11}$$

Since it might be the case that in an optimization period t certain service invocations j_{i_p} are currently running, we need to add the corresponding remaining execution times, indicated by $e^{run}_{j_{i_p}}$, in (2). If the deadlines DL_{i_p} were violated, the corresponding durations $e^p_{i_p}$ would increase which in turn would lead to higher penalty cost (see (1)).

The constraints in (12) make sure that for all $v \in V$, $k \in K_v$, VM instances k_v will be leased if service invocations j_{i_p} are to be scheduled on them. This is indicated by a value of 1 for the corresponding decision variables, i.e., $x_{(j_{i_p}, k_v, t)} = 1$. The corresponding VM instance k_v has already been leased (and paid for) in a previous optimization period, which is indicated by a value of 1 for the parameter β_{k_v}, i.e., $\beta_{(k_v,t)} = 1$. Or we need to set the corresponding decision variable $y_{(k_v,t)}$ accordingly, i.e., $y_{(k_v,t)} \geq 1$. Since it may be the case that multiple service invocations are intended to be scheduled on VM instance k_v, the sum of the left-hand side of (12) might exceed a value of 1. Thus, in order to satisfy these constraints, we multiply $(\beta_{(k_v,t)} + y_{(k_v,t)})$ with a sufficiently large value M, i.e., 1,000,000. We chose this value since it is unlikely that there is a VM which is able to host more than 1,000,000 parallel service invocations.

$$\sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in J_{i_p}} x_{(j_{i_p}, k_v, t)} \leq (\beta_{(k_v,t)} + y_{(k_v,t)}) \cdot M \tag{12}$$

With respect to scheduling service invocations on a VM instance k_v, we demand the corresponding service types st_j in (13) to be the same. This way, we aim at achieving that a VM instance only invokes service instances of the same type. Differently stated, service instances with different service types may not be invoked at the same VM instance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

$$x_{(j^1_{i^1_{p^1}},k_v,t)} + x_{(j^2_{i^2_{p^2}},k_v,t)} \leq 1 \tag{13}$$

The constraints in (14) and (16) make sure that for all $v \in V$, $k \in K_v$ the resources (with respect to CPU and RAM) required by the service invocations that either already run on VM instance k_v or are scheduled to run on it, do not exceed the respective capacity of a VM of the type v. The remaining *free* capacities $f^C_{k_v}$ and $f^R_{k_v}$ are determined in (15) and (17). Note that we consider *free* capacities $f^C_{k_v}$ and $f^R_{k_v}$ for VMs that are either already running or leased in period t. For this, we use an additional variable $g_{(k_v,t)} \in \{0,1\}$, which takes a value of 1 only if VM k_v is either running ($\beta_{k_v} = 1$) or leased ($y_{(k_v,t)} \geq 1$) in period t. This is indicated in (18)-(20) for all $v \in V$, $k \in K_v$.

$$\sum_{p\in P}\sum_{i_p\in I_p}\sum_{j_{i_p}\in(J^*_{i_p}\cup J^{run}_{i_p})} r^C_{(j_{i_p},k_v)} x_{(j_{i_p},k_v,t)} \leq s^C_v \tag{14}$$

$$g_{(k_v,t)} \cdot s^C_v - \sum_{p\in P}\sum_{i_p\in I_p}\sum_{j_{i_p}\in(J^*_{i_p}\cup J^{run}_{i_p})} r^C_{(j_{i_p},k_v)} x_{(j_{i_p},k_v,t)} \leq f^C_{k_v} \tag{15}$$

$$\sum_{p\in P}\sum_{i_p\in I_p}\sum_{j_{i_p}\in(J^*_{i_p}\cup J^{run}_{i_p})} r^R_{(j_{i_p},k_v)} x_{(j_{i_p},k_v,t)} \leq s^R_v \tag{16}$$

$$g_{(k_v,t)} \cdot s^R_v - \sum_{p\in P}\sum_{i_p\in I_p}\sum_{j_{i_p}\in(J^*_{i_p}\cup J^{run}_{i_p})} r^R_{(j_{i_p},k_v)} x_{(j_{i_p},k_v,t)} \leq f^R_{k_v} \tag{17}$$

$$g_{(k_v,t)} \geq \beta_{(k_v,t)} \tag{18}$$

$$g_{(k_v,t)} \geq y_{(k_v,t)} \tag{19}$$

$$g_{(k_v,t)} \leq \beta_{(k_v,t)} + y_{(k_v,t)} \tag{20}$$

The constraints in (21) demand that the remaining leasing duration d_{k_v} plus the number $y_{(k_v,t)}$ of BTUs for VM k_v (for all $v \in V$, $k \in K_v$) is larger or equal to the sum of remaining execution time $e_{j_{i_p}}$, the deployment time $\Delta_{j_{i_p}}$ (if the corresponding service type is not yet deployed, i.e., $z_{(j_{i_p},k_v,t)} = 0$), and the VM start-up time Δ, provided the VM is not yet running ($\beta_{(k_v,t)} = 0$). As we conduct a worst-case analysis, this remaining leasing duration has to be greater or equal to the service execution times of all *next* service invocations (for all $p \in P$, $i_p \in I_p$, $j_{i_p} \in J^*_{i_p}$) that should be scheduled on this VM including the corresponding service deployment and VM start-up time.

This way, we make sure that service invocations scheduled on a VM instance will not be *moved* to another VM instance during their invocation. The same restriction is provided by the constraints in (22) for all currently running service invocations. In order to make sure that the running service invocations can be finished on the concrete VM k_v where they have previously been assigned to, we explicitly consider this VM k_v by using an additional index k_v for $e^{run_{k_v}}_{j_{i_p}}$ in (22). Since these service invocations are already running, we do not need to consider service deployment and VM start-up times.

$$\begin{aligned}&(e_{j_{i_p}} + \Delta_{j_{i_p}} \cdot (1 - z_{(j_{i_p},k_v,t)}) \\ &+ \Delta \cdot (1 - \beta_{(k_v,t)}))x_{(j_{i_p},k_v,t)} \\ &\leq d_{(k_v,t)} + y_{(k_v,t)} \cdot BTU\end{aligned} \tag{21}$$

$$e^{run_{k_v}}_{j_{i_p}} \leq d_{(k_v,t)} + y_{(k_v,t)} \cdot BTU \tag{22}$$

The sum of the $y_{(k_v,t)} \geq 1$ (for $v \in V$) values indicates the total number of VM instances and for how many $BTUs$ they have to be leased. This sum determines $\gamma_{(v,t)}$ in (23). In this respect, it has to be noted that $y_{(k_v,t)}$ includes both, the decision, which concrete VM instance k_v to lease and for how many $BTUs$.

$$\sum_{k\in K_v} y_{(k_v,t)} \leq \gamma_{(v,t)} \tag{23}$$

Finally, in (24), we make sure that for all $p \in P$, $i_p \in I_p$, $j_{i_p} \in J^*_{i_p}$ each service invocation can be scheduled only on one VM instance. In (25), we set the decision variables for all service invocations already running (for all $p \in P$, $i_p \in I_p$, $j_{i_p} \in J^{run}_{i_p}$, $v \in V, k \in K_v$) in period t to 1. The constraints in (26)-(29) restrict the decision variables $x_{(j_{i_p},k_v,t)}$ (for all $p \in P$, $i_p \in I_p$, $j_{i_p} \in J^*_{i_p}$, $v \in V, k \in K_v$), $y_{(k_v,t)}$ (for all $v \in V$, $k \in K_v$), and $e^p_{i_p}$ (for all $p \in P$, $i_p \in I_p$) to take values from $\{0,1\}$, $\mathbb{N}_0$, and $\mathbb{R}^+$, respectively.

$$\sum_{v\in V}\sum_{k\in K_v} x_{(j_{i_p},k_v,t)} \leq 1 \tag{24}$$

$$x_{(j_{i_p},k_v,t)} = 1 \tag{25}$$

$$x_{(j_{i_p},k_v,t)} \in \{0,1\} \tag{26}$$

$$g_{(k_v,t)} \in \{0,1\} \tag{27}$$

$$y_{(k_v,t)} \in \mathbb{N}_0 \tag{28}$$

$$e^p_{i_p} \in \mathbb{R}^+ \tag{29}$$

The optimization model for the current optimization period t is obtained by assembling (1)-(29).

Having defined the optimization model used for computing an optimal solution to the SIPP in period t, we are now able to realize a multiperiod scheduling as described in the next section.

4.3 Multiperiod Scheduling Approach

Applying the approach for computing an optimal solution to the SIPP as presented in the previous Section 4.2, we obtain a scheduling plan, i.e., which service invocation to schedule in period τ_t. But as new requests for the invocation of further process instances i_p may arise during the scheduled service invocations, the previously computed *optimal* solution might no longer be optimal. Thus, we may not only account for one single optimization period but need to conduct multiple optimization steps.

In each optimization step, we only schedule the *next* service invocations for the process instances i_p which cannot be delayed any further according to (2). For this, we extract

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

the decision variables $x_{(j_{i_p},k_v,t)}$, lease and start corresponding VMs k_v (if they have not been leased and started yet), deploy respective service invocations j_{i_p} on aforementioned VMs (if they have not been deployed yet), and initiate and monitor their invocation in terms of success and execution time.

The next optimization period in time, i.e., where the next optimization step will be carried out, is indicated by the variable τ_{t+1} – as a result of the optimization. If the invocation of a process instance i_p is finished until τ_{t+1}, we determine whether a QoS violation occurred, i.e., we compute $e^p_{i_p}$. In case of a QoS violation, i.e., if $e^p_{i_p} > 0$, corresponding penalties accrue.

Immediately prior to the next optimization step, i.e., for the next optimization period at τ_{t+1}, which then *becomes* the new current optimization period at τ_t in (3), we need to update certain parameters as described subsequently. We set:

- $\beta_{(k_v,t)} = 1$ if VM k_v runs in τ_t; 0 otherwise.
- $z_{(j_{i_p},k_v,t)} = 1$ if service invocation j_{i_p} with the same service type $st_j = st_{j_{i_p}}$ is already deployed at VM k_v in τ_t; 0 otherwise.
- $d_{(k_v,t)} = d_{(k_v,t-1)} + y_{(k_v,t)} \cdot BTU - (\tau_t - \tau_{t-1})$ to account for the time elapsed between the previous (τ_{t-1}) and the current (τ_t) optimization period when computing the remaining leasing duration of VM k_v in optimization period τ_t.

Service invocations that have already been scheduled in previous periods are referred to as $j^{run}_{i_p}$. For such service invocations, we determine the remaining execution times $e^{run}_{j_{i_p}}$ as follows: We subtract the time elapsed since the point in time where the corresponding service invocations have been scheduled (τ_{t_s}) from the sum of the services' execution times, the VM start-up times, and the services' deployment times. This is shown in (31) whereas $\hat{e}_{j_{i_p}}$ is defined in (30). Optimization period τ_{t_s} refers to the period where service invocation j_{i_p} has been scheduled.

$$\hat{e}_{j_{i_p}} = e_{j_{i_p}} + \Delta_{j_{i_p}} + \Delta \tag{30}$$

$$e^{run}_{j_{i_p}} = \begin{cases} 0 & \text{, if } j_{i_p} \text{ is finished} \\ \max\left(0, \hat{e}_{j_{i_p}} - (\tau_t - \tau_{t_s})\right) & \text{, else} \end{cases} \tag{31}$$

Referring to (31), the remaining execution times are considered as 0 if the corresponding service instances (j_{i_p}) have already been invoked, i.e., the service invocation is finished. Further, by taking the maximum in (31), we make sure not to consider negative remaining execution times. This means, if an invocation lasts longer than expected, the SIPP has to include this information in the next optimization period, i.e., the corresponding process instance will have a higher priority as it may get delayed otherwise.

Having set up and updated the necessary parameters for the next period, we conduct another optimization step. According to the results of this newly conducted optimization step, i.e., according to the values of the decision variables $x_{(j_{i_p},k_v,t)}$, we schedule and invoke corresponding service instances. Continuing this optimization and scheduling procedure results in an efficient scheduling strategy, which takes optimal scheduling decisions (at the optimization periods τ_t) into account and minimizes unused, free VM capacities (see the usage of $f^C_{k_v}$ and $f^R_{k_v}$ in (1)).

5 Evaluation

As a proof of concept, the proposed SIPP model has been thoroughly evaluated. We apply our prototype and testbed framework ViePEP as presented in [13], [16]. ViePEP is a BPMS and Cloud controller which acts as a middleware between process owners and the Cloud-based computational resources. Clients can model processes and request their enactments by ViePEP. The core ViePEP functionalities are provided in a BPMS VM which is capable of receiving requests and leasing as well as releasing computational Cloud resources. Further, the BPMS VM is responsible for computing how many resources are really needed, and where to deploy the single service instances. For this, an optimization model like the SIPP is needed.

The leased VMs are called Backend VMs and host the actual software services, which have to be invoked in order to process a single process step. Next to the services, a monitoring component is deployed which monitors the underlying VM in terms of CPU load and RAM usage. Backend VMs are connected by a message queue to the BPMS VM thus the monitored data can be transferred. In addition, a Service Registry is provided, which hosts the actual services in form of deployable Web ARchives (WAR-) files.

The Backend VMs are based on an OSGi 4.3-based framework[2], i.e., Apache Karaf 2.3.2[3]. As a communication engine between the Backend VMs and the BPMS VM we apply a JMS-based message queue, i.e., Apache ActiveMQ 5.7.0[4]. ViePEP itself is implemented using Java. Both the BPMS VM and the Backend VMs are running in a private Cloud testbed running OpenStack (OpenStack Folsom[5]). To solve the optimization problem, CPLEX[6] is applied.

In the following subsections, we present the evaluation setting (Section 5.1), the applied metrics (Section 5.2), and discuss the quantitative evaluation results (Section 5.3).

5.1 Setting

5.1.1 Test Collection

To evaluate the proposed optimization approach, we choose a subset of the SAP reference model [27], [28]. The SAP reference model has been analyzed and exploited in various scientific papers and provides a well-known and widely accepted foundation for our evaluation [29]. Out of the 604 process models in the SAP reference model, we choose 10 exemplary process models which feature different degrees of complexity in terms of process patterns.

Table 2 shows the basic characteristics of the 10 different models, i.e., if a model includes AND-blocks, XOR-blocks, Repeat Loops or a combination of them. Notably, each

2. http://www.osgi.org/Main/HomePage
3. http://karaf.apache.org/
4. http://activemq.apache.org
5. http://www.openstack.org/software/folsom/
6. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

TABLE 2: Evaluation Process Models

Name	|Steps|	$\|XOR\|$	$\|AND\|$	$\|loops\|$
1	3	0	0	0
2	2	1	0	0
3	3	0	1	0
4	8	0	2	0
5	3	0	1	0
6	9	1	1	0
7	8	0	0	0
8	3	0	1	0
9	4	1	1	1
10	20	0	4	0

split (AND, XOR) also includes a join. Figure 2 shows two example process models, namely No. 5, which contains an XOR-block, and No. 9, which contains an AND leading to two new branches. No. 9 also contains a Repeat Loop.

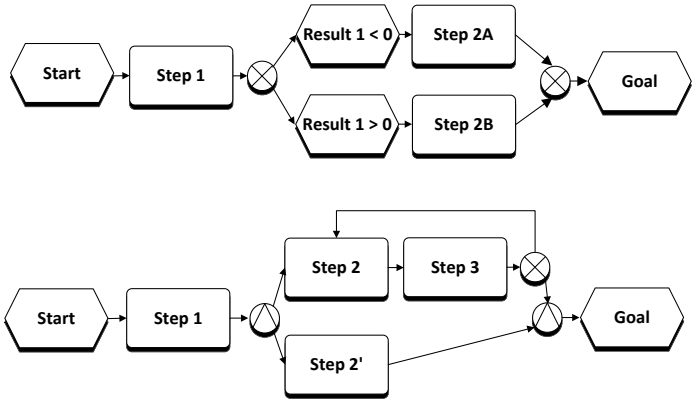


Fig. 2: Process No. 5 (top), Process No. 9 (bottom)

While process models in the SAP reference model usually contain some human-provided services, we are focusing on software services. Hence, for all services in the process models, we deploy services with differing degrees of computational complexity, resource demands, and duration. For this, we apply the *lookbusy* load generator[7], which is a configurable tool able to generate a particular CPU load for a particular time span.

Following our assumption that services are shared among processes, we generate 10 different software services as described in Table 3. Duration and CPU Load are mean values μ_{cpu} and μ_{dur} of normal distributions as described in (32) and (33). We assume that the services' actual resource consumption varies to some extent for each service invocation. We assume $\sigma_{cpu} = \mu_{cpu}/10$ respectively $\sigma_{dur} = \mu_{dur}/10$ and we only select values between 95% and 105% of the provided mean value to conduct a reproducible evaluation.

$$f(x, \mu_{cpu}, \sigma_{cpu}) = \frac{1}{\sigma_{cpu}} \phi \left(\frac{x - \mu_{cpu}}{\sigma_{cpu}} \right) \tag{32}$$

$$f(x, \mu_{dur}, \sigma_{dur}) = \frac{1}{\sigma_{dur}} \phi \left(\frac{x - \mu_{dur}}{\sigma_{dur}} \right) \tag{33}$$

To estimate the need for computational resources for a service invocation on different VMs, we assume that each service invocation can be fully parallelized among the available CPUs. This means, while the mean execution time stays the same, the amount of required CPU load is divided by the number of available cores. E.g., if a service invocation needs 100% of a single core VM, only a quarter of it will be used on a quad core VM, i.e., 25% for each core. This enables 4 times more simultaneous invocations of that particular service instance on a quad core VM compared to a single core VM.

TABLE 3: Evaluation Services

Service No.	CPU Load in% (μ_{cpu})	Service Makespan in sec. (μ_{dur})
1	5	30
2	10	80
3	15	120
4	30	100
5	45	10
6	55	20
7	70	40
8	125	20
9	125	60
10	190	30

5.1.2 Applied SLAs

SLAs are defined on process level and contain the deadline for the complete process enactment. To evaluate our optimization approach under different settings, two different SLAs are linked to each process model and evaluated in separate runs. The first set of SLAs provides rather "lenient" values, i.e., the SLAs are defined with significant leeway for delays, while the second set provides more strict values. For the lenient SLAs, the average violation threshold is set to $2.5 * ED_{I_p}$, while for the strict SLAs, the threshold is $1.5 * ED_{I_p}$, where ED_{I_p} is the mean execution time of a process model. Those values were chosen with having in mind the start-up time for VMs Δ and the time which is needed to deploy the respective service onto them, i.e., the service deployment time Δ_{st_j}.

5.1.3 Process Request Arrival Patterns

We apply different process request arrival patterns. The first scenario follows a *Constant* arrival pattern, i.e., in regular intervals (120 seconds), we choose 5 instances of different process models, i.e., in the first round we request process model No. 1 to 5, in the second round No. 6 to 10, the third round again No. 1 to 5 etc. This is repeated until a total of 50 process instance requests are sent to ViePEP.

The second arrival pattern follows a *Pyramid*-like function and the process instances requested at particular points of time are randomly chosen: A total of 100 process instances are put in a randomly shuffled queue and a different amount of process instance requests is sent simultaneously. We start with a low number, e.g., with 1 instance a time, increase it to a peak, and decrease it again to 1 instance request. After a few iterations, the amount increases again slowly to a peak until all 100 instance requests are sent to ViePEP. The detailed function can be found in (34).

$$f(n) = a \begin{cases} (n+1)/(n+1) & \text{if } 0 \leq n \leq 3 \\ \lceil (n+1)/4 \rceil & \text{if } 5 \leq n \leq 17 \\ 0 & \text{if } 18 \leq n \leq 19 \\ (n/n) & \text{if } 20 \leq n \leq 35 \\ \lceil (n-9)/20 \rceil & \text{if } 36 \leq n \leq 51 \end{cases} \tag{34}$$

7. http://devin.com/lookbusy/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

TABLE 4: Evaluation Results

Arrival Pattern	Constant Arrival				Pyramid Arrival			
	SIPP		Baseline		SIPP		Baseline	
SLA Level	Strict	Lenient	Strict	Lenient	Strict	Lenient	Strict	Lenient
Number of total Process Requests	50				100			
Interval between the Process Request	120 seconds				60 seconds			
Number of parallel Process Requests	$y = 5$				$f(n)$ (see (34))			
SLA Adherence in % (Standard Deviation)	94.0 (σ=2.0)	96.0 (σ=2.0)	68.66 (σ=18.33)	92.66 (σ=1.55)	100.0 (σ=0.0)	99.67 (σ=0.58)	67.33 (σ=3.06)	96.33 (σ=0.58)
Total Makespan in Minutes (Standard Deviation)	23.33 (σ=0.58)	31.33 (σ=4.51)	22.67 (σ=1.15)	25.67 (σ=2.89)	58.67 (σ=2.52)	57.67 (σ=0.58)	57.0 (σ=0.0)	53.33 (σ=1.15)
Leasing Cost (Standard Deviation)	1136.0 (σ=80.88)	780.0 (σ=78.0)	1738.33 (σ=20.21)	1493.33 (σ=40.41)	2339.33 (σ=226.89)	2176.33 (σ=268.59)	3628.33 (σ=40.41)	3430.0 (σ=121.24)
Penalty Cost (Standard Deviation)	10.0 (σ=3.61)	5.33 (σ=4.93)	42.67 (σ=15.31)	7.0 (σ=4.36)	0.0 (σ=0.0)	0.33 (σ=0.58)	88.0 (σ=18.36)	7.33 (σ=3.21)
Total Cost (Standard Deviation)	1146.0 (σ=82.66)	785.33 (σ=81.13)	1781.0 (σ=18.74)	1500.33 (σ=38.89)	2339.33 (σ=226.89)	2176.67 (σ=268.78)	3716.33 (σ=37.55)	3437.33 (σ=124.42)

In this function, n represents the time in minutes, which is used to calculate a and a represents the amount of process instance requests which will be sent to ViePEP. Between each bunch of a requests we assign a waiting period of 60 seconds. It remains to mention that for repeated evaluation runs, the same order of process models in the queue is applied in order to generate reproducible results.

The arrival patterns are depicted in Table 4 and Figures 3 and 4.

5.1.4 Baseline

To compare our optimization approach against a baseline, we apply a basic strategy for resource provisioning and scheduling which is based on existing work on process scheduling [10], [30], [31]. It should be noted that we adapted the strategy slightly to fit our basic assumption that service instances may be shared simultaneously among process instances and that Backend VMs host only one particular service instance but there might be several concurrent service invocations.

Applying the baseline strategy (*X_VM for Each*), ViePEP leases a new quad-core Backend VM for a particular service type once the workload on a particular VM is above an upper threshold of 80%. The VM is released again once the workload is below a lower threshold of 20%. These values have been chosen based on experiences collected in our former work [14], [15], [16]: The 20% lower threshold was chosen due the fact that the operating system needs up to 15%, which means, a overall system load of less than 20% means that there are only few service invocations running, and the VM is not needed anymore. The 80% upper threshold has been chosen to be able to handle unexpected deviations of needed resources for running service invocations. However, this does not mean that a VM will not be used 100%.

Hence, there might be several VMs for the same service type. This approach applies a basic scheduling of process instances and in addition takes into account the near future, i.e., VMs are leased for a fixed period which allows to prepone future steps in order to use leased resources more efficient. Further, the baseline also considers the deadline, i.e., processes with an earlier deadline have to be enacted before processes with a later deadline.

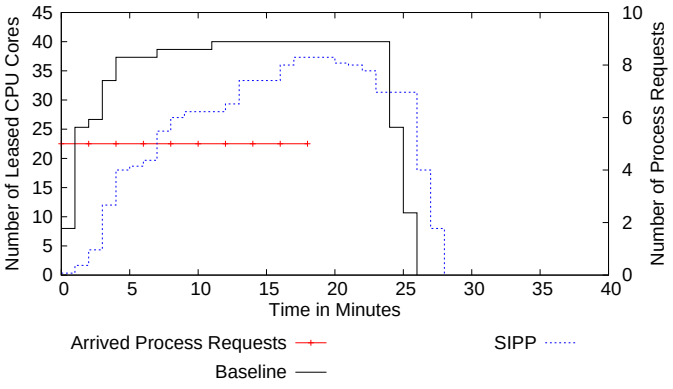
5.2 Metrics

To assess the quality of our optimization approach, we use different metrics. For all numbers, we also calculate the standard deviation σ. Since our optimization problem aims at cost minimization, comparing the *total cost* arising from VM leasing *and* penalties is an obvious option. For this, we apply the following cost model: We assume that it is cheaper to lease a quad-core VM than 4 single-core VMs (respectively a dual-core VM is cheaper than 2 single-cores, etc.). We also apply a linear penalty cost model based on [10]: We assign 1 unit of *penalty cost* per 10% of time units of delay. Penalty cost and VM leasing cost over the runtime of all process instances result in the *Total Cost* metric. Second, we measure the *SLA Adherence*, i.e., the percentage of process requests which have been fulfilled on time. Third, we measure the overall duration to process all requests (*Total Makespan*), starting with the point of time when the first request arrives and stopping once all service invocations have been finished.

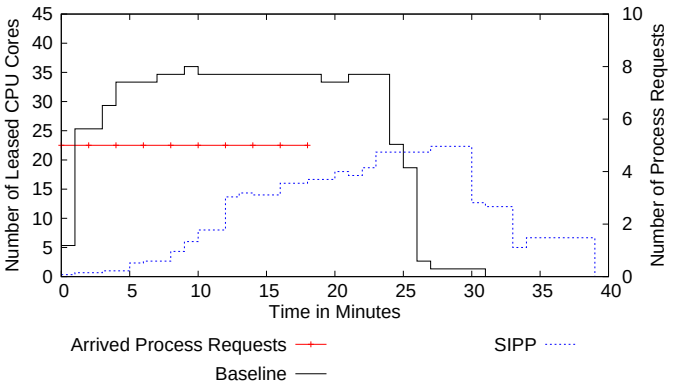
5.3 Results and Discussion

In order to get representative numbers, each scenario (with the SIPP model and the baseline, with lenient and strict SLAs) was executed 3 times over a time span of 7 days. This has been done to avoid corruption of results due to differing base loads in the OpenStack-based Cloud testbed. While Table 4 presents the average of the conducted evaluation runs including their standard deviation as numbers, Figures 3 and 4 show the results as charts. For each chart, on the horizontal axis, the time in minutes is presented, on the *left* vertical axis the number of leased CPU cores is shown and on the *right* vertical axis, the number of parallel process requests is presented. It remains to mention that while the metric *Total Makespan in Minutes* in Table 4 represents the time of executing all process requests, the *Time in Minutes* in Figures 3 and 4 shows the time span of leasing the first VM until releasing the last one.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

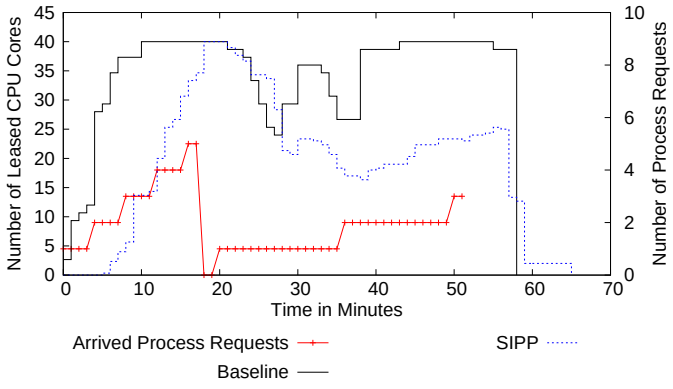


(a) Constant Scenario – Strict SLA

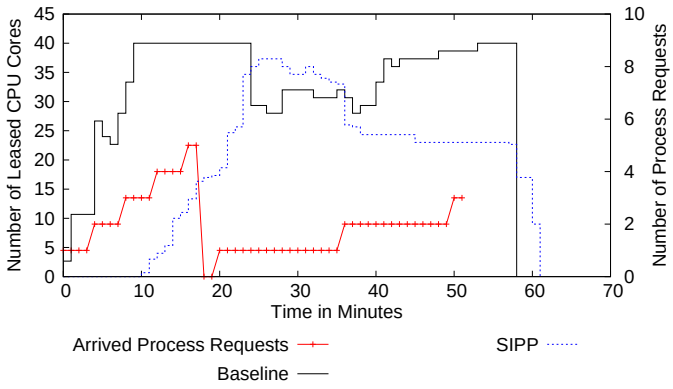


(b) Constant Scenario – Lenient SLA

Fig. 3: Evaluation Results – Constant Scenario



(a) Pyramid Scenario – Strict SLA



(b) Pyramid Scenario – Lenient SLA

Fig. 4: Evaluation Results – Pyramid Scenario

First, we discuss the Constant Arrival scenario for both SLA levels, lenient and strict. As it can be seen in Table 4, the SLA adherence using our optimization approach is always above 94% (with a rather small standard deviation of 2.00%). This means, our optimization approach was capable of detecting potential shortcomings in time, and rescheduled the service invocations. However, the few SLA violations can be reduced to the fact that it may be cheaper in some situations to accept a short delay than leasing additional VMs.

The SLA adherence for the baseline is much lower: As the numbers show, almost a third of the process requests were delayed resulting in four times higher penalty cost (42.67) than using our SIPP model (10.0) for the strict SLA level. The low amount of penalty cost for SIPP can be explained by our leasing and releasing policy, i.e., we try to lease exactly as many resources as needed, while the baseline leases additional resources ad hoc, i.e., when a specific threshold has been reached.

The fact that by applying the baseline each time a new quad-core VM was leased, resulted in much higher cost, namely almost 55% higher than applying the SIPP model. Comparing the overall makespans, we see that the difference between lenient and strict SLA levels for the baseline can be neglected, i.e., 22.67 vs. 25.67 minutes. Comparing these numbers with SIPP, we see that the baseline was even faster than our approach. This is due to the fact that leasing a quad-core VM allows much more service invocations at a certain point of time. However, due to the limited amount of requested process instances, the leased resources were not fully utilized, but have to be paid for the full BTU. This can be seen in Figures 3a and 3b around minute 25, i.e., the amount of leased CPUs literally falls from 40 to 0, compared to our approach, where the releasing follows more a step-like function.

Second, we discuss the Pyramid Arrival scenario. Again, using our approach, we achieve a close to 100% SLA adherence, thus we can assume that the scheduling plan created by the SIPP model considered all given SLAs and assigned the service invocations accordingly. The fact that the baseline with strict SLA level ended up in more than 30% SLA violations, can be linked to its leasing and releasing policy of Cloud resources, i.e., it leases and releases the resources based on a threshold. Comparing the baseline with our approach with regard to time, the overall makespan for our approach is slower in every case, however, less resources are needed as our SIPP is able to distinguish what kind of VMs are needed, i.e., whether a single-, dual-, triple-, or quad-core Backend VM should be leased.

Notably, the SLA adherence for the strict and lenient SLA levels differ for the SIPP. The evaluation shows that applying stricter SLAs requires more resources in less time. This can perfectly be seen in Figures 4a and 4b. Since a higher-cored VM is comparably cheaper than several lower-cored ones, they are more likely to be leased when a strict SLA is applied (see Figure 4a) as more invocations have

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

to be processed in shorter time. Since this enables a faster invocation of the process instances, we observe a smaller number of SLA violations. Comparing this to the lenient SLAs (see Figure 4b), we see that less resources are acquired at the beginning. Due to that, and since our services mirror real-world services where the CPU load and execution time is not deterministic, we experience a few SLA violations ($\sim$1%), which is acceptable by the definition of our SIPP model.

As the numbers reveal, by using the SIPP approach, we were able to reduce the overall cost by 41.66% for the constant arrival pattern and 38.86% for the pyramid arrival pattern. Further, we achieved a shorter total makespan compared to the baseline. Therefore, it can be deducted that leasing of additional resources in an ad hoc manner may end up in faster process enactments, but will also result in higher cost, since resources are always leased for a defined BTU. Thus, the resources have to be paid, but may not be needed later on.

Our evaluations have shown that using an optimization model for creating a scheduling plan for complex process patterns will create much lower cost than using an ad hoc approach. Although a MILP solver is used to find an optimal solution to such problems, having real-world factors in the evaluation, such as a fluctuation of resource usage, the overall outcome will always be affected. However, the fact that we experienced less than 5% of SLA violations and an average cost improvement of almost 40% definitely shows that using an optimization model for scheduling service instances and their invocations among Cloud-based computational resources should always be preferred over an ad hoc approach.

6 Related Work

Resource allocation and service scheduling in an automated way is a major research challenge in the field of Cloud computing [1], [11], and several approaches have observed the problem with regard to single services or Scientific Workflows (SWFs) [5]. Resource allocation approaches for single services do not take into account the process perspective, i.e., resources are scaled rather in an ad hoc manner. While resource allocation and scheduling for SWFs offers interesting insights, there are certain differences between business processes and SWFs in terms of QoS requirements, dataflow- vs. control flow-orientation, or process instantiation. These prevent a direct adaptation for the use in the work at hand [32].

To the best of our knowledge, relatively little effort has been invested into resource optimization and scheduling for elastic processes. In general, resource constraints for business processes are a relatively neglected topic [33]. Xu et al. propose business processes scheduling considering time and resource availability constraints. Doing so, a greedy algorithm and two holistic algorithms are proposed [34]. Pla et al. propose an approach to assign resources to process tasks during runtime using an auction-based mechanism [35]. However, in both cases only one task is assigned to a particular resource. Cloud resources are not regarded and scalability is also not taken into account.

In the field of elastic processes, Juhnke et al. provide an extension to a standard BPEL engine which allows the usage of Cloud-based computational resources leased in an on-demand fashion in order to execute business processes [36]. Within their scheduling, the authors consider cost for VMs as well as data transfer cost. Bessai et al. aim at cost or time optimization or a pareto-optimal solution covering both aspects [37]. Wei et al. allow generic QoS constraints, but do not explicitly discuss deadline-constrained scheduling [38]. Instead, the goal is to optimize overall resource utilization. Amziani et al. discuss the modeling of single elastic processes using Petri nets and simulate elasticity strategies, but do not take into account scheduling [39]. In contrast to our work, none of these approaches takes into account any user-specific SLAs, e.g., a deadline which defines the latest point of time at which the process enactment has to be finished.

Euting et al. propose an approach for BPM-aware Cloud computing which exploits the knowledge about scheduled processes ensuring timeliness and improves the quality of resource allocations [40]. In order to do so, they introduce an IaaS resource controller based on fuzzy theory. The solution monitors process executions and is able to predict and control resource requirements for future steps. Other researchers provide resource allocation approaches which lease further public Cloud resources if a process cannot be carried out in time in a private Cloud [41], [42]. The goal is to enact a process cost-effectively given the deadline. In [41], only single process instances are regarded. Contrary to that, [42] allows resource sharing between processes. In contrast to our work, both approaches do not regard service-based processes, where services may be shared between different process instances.

Wu et al. present a scheduling algorithm for service-based processes [43]. Similar to our work, the goal is to minimize the cost for a Cloud-based process landscape. In the first stage, services which could fulfill a process step are located. Services which are not available are deployed in a private Cloud, while existing services could be located anywhere in the public Cloud. This scheduling is only carried out for individual process instances. In the second step, a local scheduler optimizes the task-to-VM assignment in the local Cloud. For this, different metaheuristics are applied. While this approach applies heuristics, it nevertheless comes closest to our work.

Cai et al. also model the scheduling problem as a MILP problem [44]. Similar to our work, their goal is to minimize the cost under given time constraints. However, only single processes are taken into account and consequently, resources may not be shared among processes. The authors propose a heuristic which is based on critical path optimization to solve the problem. An optimal solution is also provided.

Klein et al. extend the "classic" service composition problem to the Cloud [45] . The authors do not regard service instantiation and invocations, but analyze QoS of services distributed in the Cloud. While this topic is only indirectly related to the work at hand, the approach could be an interesting aspect for future research regarding elastic process enactment in public Clouds.

None of the abovementioned approaches provides optimal scheduling and resource allocation for complex process patterns on a global level. Also, penalties are not taken

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2428246, IEEE Transactions on Services Computing

into account during scheduling. Apart from Cai et al. [44] and our own former work [16], none of the discussed related work models the problem using MILP. However, as mentioned in Section 1, our former work is restricted in terms of penalties, recognition of BTUs, and complex process patterns. Also, we have only applied a heuristic solution to the optimization problem, while in the work at hand, we facilitate an optimal solution.

7 Conclusion & Future Work

Using Cloud-based computational resources to enact business processes seems to be an obvious choice. However, there is still a lack of BPMS frameworks which are able to lease and release resources for process enactment in a cost- and time-efficient manner. Within this paper, we presented a novel solution for scheduling complex elastic processes. We defined the SIPP by considering a worst-case scenario of the process structures. We used MILP to solve the optimization model. Our solution has been thoroughly evaluated and compared against an ad hoc baseline solution to scheduling. The evaluation has shown that using such an optimization model can heavily reduce leasing and penalty cost compared to the baseline: In our evaluations the BPMS issued on average 41.66% less cost for the constant arrival pattern and 36.86% less cost for the pyramid arrival pattern.

In our future research, we want to extend the optimization model and combine several Cloud providers resulting in a hybrid Cloud environment. While this allows accessing more Cloud-based computational resources and enables us to enact more processes simultaneously, it also leads to more complex research topics such as *privacy aspects*, i.e., some services may only be invoked in private Clouds, *data transfer cost*, i.e., transferring service instances and data from and to different Cloud providers, and *data storage cost*, which arise if a large amount of data has to be stored for a certain amount of time.

Acknowledgements

This work is partially supported by the Commission of the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201) and the ADVENTURE project (Grant agreement no. 231396).

The authors thank IBM for providing an academic license for CPLEX.

References

[1] 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.

[2] 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.

[3] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing – The business perspective," *Decision Support Systems*, vol. 51, pp. 176–189, 2011.

[4] Gartner, Inc., *Forecast: Public Cloud Services, Worldwide, 2011-2017, 4Q13 Update*, 2013.

[5] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch, "Elastic Business Process Management: State of the Art and Open Challenges for BPM in the Cloud," *Future Generation Computer Systems*, vol. 46, pp. 36–50, 2015.

[6] M. Maurer, I. Brandic, and R. Sakellariou, "Adaptive resource configuration for Cloud infrastructure management," *Future Generation Computer Systems*, vol. 29, pp. 472–487, 2013.

[7] 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.

[8] F. Leymann, D. Roller, and M.-T. Schmidt, "Web services and business process management," *IBM Systems Journal*, vol. 42, no. 2, pp. 198–211, 2002.

[9] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated Control in Cloud Computing: Challenges and Opportunities," in *1st Works. on Automated Control for Datacenters and Clouds (ACDC'09)*. ACM, 2009, pp. 13–18.

[10] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar, "Cost-Efficient and Application SLA-Aware Client Side Request Scheduling in an Infrastructure-as-a-Service Cloud," in *5th Intern. Conf. on Cloud Computing (CLOUD 2012)*. IEEE, 2012, pp. 213–220.

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

[12] L. Wu, S. K. Garg, and R. Buyya, "SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments," in *11th IEEE/ACM Intern. Symp. on Cluster, Cloud and Grid Computing (CCGrid 2011)*. IEEE, 2011, pp. 195–204.

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

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

[15] P. Hoenisch, S. Schulte, and S. Dustdar, "Workflow Scheduling and Resource Allocation for Cloud-based Execution of Elastic Processes," in *6th IEEE Intern. Conf. on Service Oriented Computing and Applications (SOCA 2013)*. IEEE, 2013, pp. 1–8.

[16] S. Schulte, D. Schuller, P. Hoenisch, U. Lampe, S. Dustdar, and R. Steinmetz, "Cost-Driven Optimization of Cloud Resource Allocation for Elastic Processes," *Intern. Journal of Cloud Computing*, vol. 1, no. 2, pp. 1–14, 2013.

[17] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[18] U. Lampe, O. Wenge, A. Müller, and R. Schaarschmidt, "On the Relevance of Security Risks for Cloud Adoption in the Financial Industry," in *19th Americas Conf. on Information Systems (AMCIS 2013)*. AIS, 2013.

[19] S. Schulte, P. Hoenisch, C. Hochreiner, S. Dustdar, M. Klusch, and D. Schuller, "Towards Process Support for Cloud Manufacturing," in *18th IEEE Intern. Enterprise Distributed Object Computing Conf. (EDOC 2014)*. IEEE, 2014, pp. 142–149.

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

[21] A. Q. Gill, D. Bunker, and P. Seltsikas, "An Empirical Analysis of Cloud, Mobile, Social and Green Computing – Financial Services IT Strategy and Enterprise Architecture," in *IEEE Ninth Intern. Conf. on Dependable, Autonomic and Secure Computing (DASC 2011)*. IEEE, 2011, pp. 697–704.

[22] G. Copil, D. Moldovan, H. L. Truong, and S. Dustdar, "Multi-level elasticity control of cloud services," in *11th Intern. Conf. on Service-Oriented Computing (ICSOC 2013)*, ser. LNCS, vol. 8274. Springer, 2013, pp. 429–436.

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

[24] D. Kossmann, T. Kraska, and S. Loesing, "An evaluation of alternative architectures for transaction processing in the cloud," in *2010 ACM SIGMOD Intern. Conf. on Management of Data (SIGMOD '10)*. ACM, 2010, pp. 579–590.

[25] D. Schuller, A. Miede, J. Eckert, U. Lampe, A. Papageorgiou, and R. Steinmetz, "QoS-based Optimization of Service Compositions for Complex Workflows," in *Intern. Conf. on Service Oriented Computing (ICSOC 2010)*, 2010, pp. 641–648.

[26] A. Strunk, "QoS-Aware Service Composition: A Survey," in *8th IEEE European Conf. on Web Services (ECOWS 2010)*. IEEE, 2010, pp. 67–74.

[27] T. A. Curran and G. Keller, *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Prentice Hall PTR, Upper Saddle River, 1997.

[28] G. Keller and T. Teufel, *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[29] J. Mendling, H. M. W. Verbeek, B. F. van Dongen, W. M. P. van der Aalst, and G. Neumann, "Detection and prediction of errors in EPCs of the SAP reference model," *Data & Knowledge Engineering*, vol. 64, no. 1, pp. 312–329, 2008.

[30] S. Genaud and J. Gossa, "Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds," in *4th Intern. Conf. on Cloud Computing (CLOUD 2011)*. IEEE, 2011, pp. 1–8.

[31] M. E. Frincu, S. Genaud, and J. Gossa, "On the Efficiency of Several VM Provisioning Strategies for Workflows with Multi-threaded Tasks on Clouds," *Computing*, vol. 96, pp. 1059–1086, 2014.

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

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

[34] J. Xu, C. Liu, X. Zhao, and S. Yongchareon, "Business Process Scheduling with Resource Availability Constraints," in *On the Move to Meaningful Internet Systems (OTM 2010)*, ser. LNCS. Springer, 2010, vol. 6426, pp. 419–427.

[35] A. Pla, B. López, and J. Murillo, "Multi-Attribute Auction Mechanism for Supporting Resource Allocation in Business Process Enactment," in *Sixth Starting AI Researchers' Symp. (STAIRS 2012)*. IOS Press, 2012, pp. 228–239.

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

[37] K. Bessai, S. Youcef, A. Oulamara, and C. Godart, "Bi-criteria strategies for business processes scheduling in cloud environments with fairness metrics," in *IEEE 7th Intern. Conf. on Research Challenges in Information Science (RCIS 2013)*. IEEE, 2013, pp. 1–10.

[38] Y. Wei and M. B. Blake, "Proactive virtualized resource management for service workflows in the cloud," *Computing*, vol. 96, no. 7, pp. 1–16, 2014.

[39] M. Amziani, T. Melliti, and S. Tata, "Formal Modeling and Evaluation of Stateful Service-Based Business Process Elasticity in the Cloud," in *On the Move to Meaningful Internet Systems (OTM 2013)*, ser. LNCS, vol. 8185. Springer, 2013, pp. 21–38.

[40] S. Euting, C. Janiesch, R. Fischer, S. Tai, and I. Weber, "Scalable Business Process Execution in the Cloud," in *IEEE Intern. Conf. on Cloud Engineering (IC2E 2014)*. IEEE, 2014, pp. 175–184.

[41] L. F. Bittencourt and E. R. M. Madeira, "HCOC: A Cost Optimization Algorithm for Workflow Scheduling in Hybrid Clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[42] X. V. Wang and X. W. Xu, "An interoperable solution for Cloud manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 29, pp. 232–247, 2013.

[43] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *The Journal of Supercomputing*, vol. 63, no. 1, pp. 256–293, 2013.

[44] Z. Cai, X. Li, and J. N. Gupta, "Critical Path-Based Iterative Heuristic for Workflow Scheduling in Utility and Cloud Computing," in *11th Intern. Conf. on Service Oriented Computing (ICSOC 2013)*, ser. LNCS, vol. 8274. Springer, 2013, pp. 207–221.

[45] A. Klein, F. Ishikawa, and S. Honiden, "Towards Network-aware Service Composition in the Cloud," in *21st World Wide Web Conf. (WWW 2012)*, 2012, pp. 959–968.

Philipp Hoenisch is a 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.

Dieter Schuller is a Postdoctoral Researcher at the Multimedia Communications Lab of Technische Universität Darmstadt, Germany. 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.

Stefan Schulte is an Assistant Professor 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.

Christoph Hochreiner Christoph Hochreiner is a PhD student at the Distributed System Group at Vienna University of Technology. Before starting his PhD, he has gathered research experience in the area of security at SBA Research. Christoph's research interests cover the whole spectrum of Cloud computing, specifically service monitoring, workload prediction and security implications.

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).