Optimizing the Performance of Automated Business Processes Executed on Virtualized Infrastructure

Christian Janiesch[1], Ingo Weber[2,] Jörn Kuhlenkamp[1], Michael Menzel[3]

[1]Karlsruhe Institute of Technology
Karlsruhe, Germany
firstname.lastname@kit.edu

[2]NICTA &
University of New South Wales
Sydney, Australia
firstname.lastname@nicta.com.au

[3]FZI Forschungszentrum Informatik
Karlsruhe, Germany
menzel@fzi.de

Abstract

With few exceptions, the opportunities cloud computing offers to business process management (BPM) technologies have been neglected so far. We investigate opportunities and challenges of implementing a BPM-aware cloud architecture for the benefit of process runtime optimization. Processes with predominantly automated tasks such as data transformation processes are key targets for this runtime optimization. In theory, off-the-shelf mechanisms offered by cloud providers, such as horizontal scaling, should already provide as much computational resources as necessary for a process to execute in a timely fashion. However, we show that making process data available to scaling decisions can significantly improve process turnaround time and better cater for the needs of BPM. We present a model and method of cloud-aware business process optimization which provides computational resources based on process knowledge. We describe a performance measurement experiment and evaluate it against the performance of a standard automatic horizontal scaling controller to demonstrate its potential.

1. Introduction

The performance of business processes is crucial to the success of a business. In order to ensure appropriate performance, monitoring of process execution as well as the continuous re-evaluation of the deployment architecture is necessary. Changes in the execution environment, demand, or the constraints of business processes such as service level agreements (SLAs) may influence the profitability of a process.

As part of business process management (BPM) the performance of a process can be monitored by metrics such as the turnaround time, i.e., the total execution time of a process instance. Figure 1 gives an overview of the composition of process turnaround times. When executed, a process instance may traverse several states from instantiation to completion. Its turnaround time consists of *idle time* (the time before a resource is scheduled), *change-over time* (when a resource has been scheduled but not started processing yet, e.g., while data is copied), and the actual *processing time* (when the scheduled resource performs the actual work), which may be *suspended* (paused) [1].

Most BPM projects which target the optimization of process turnaround time are of organizational nature and aim at reducing wait and suspension times of manually performed tasks, as well as reducing change-over times through digitizing inputs and output documents [2, 3]. Also, they may opt for switching to a different implementation, e.g., an alternative Web service, for automated tasks. However, there is also the potential of reducing the actual task processing times of automated tasks which rely on computational resources, without changing the implementation. In this work we focus on the latter: not the actions of a process task are altered, but the computational resources are better aligned with the demand.

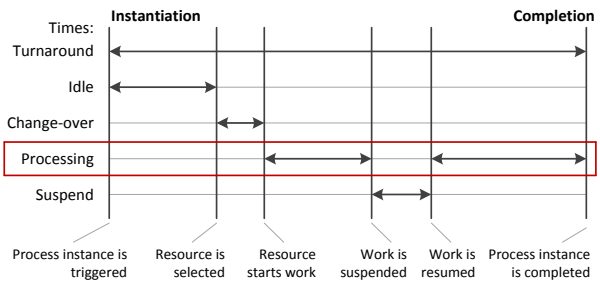

Figure 1. Process instance times – based on [1]

There are several enterprise integration scenarios in which the performance of processes primarily depends on the timely provisioning of adequate computational resources to complete automated tasks [4].

A financial analyst may for example need to translate and transfer large amounts of stock data between his system and a legacy mainframe. Similarly, a car manufacturer may need to transform and process large amounts of data received from one of his suppliers. Nowadays, most business data is communicated using XML – however, legacy systems, e.g., written in COBOL, may provide or expect other file structures. Hence translation can be required. Both tasks are of

repetitive nature and the performance strongly depends on the input parameters.

While an overprovisioning of computational resources ensures a high performance of the processes, it is not cost-effective, as argued above. Therefore, it is important to align the demand and supply of these resources as closely as possible.

With the advent of cloud computing, elastic, virtualized resources have become available, which are metered and billed according to actual use. Horizontally distributed applications deployed on commodity hardware in combination with load balancing provide high availability and scalability of services. Hence, it becomes possible to provide additional resources in a short amount of time to store data and/or perform computations [5]. In the context of BPM, the processing time of individual tasks may not be as important as the turnaround time of the overall process since only its turnaround time may be subject to a negotiated service level agreement (SLA). Hence, load balancing and horizontal scaling – the provisioning of additional processing nodes, as opposed to vertical scaling, where e.g., more RAM or CPU cores are added to a machine – have to be done with regard to prior and estimated future process turnaround times rather than with respect to the individual task's processing time. This idea has been applied successfully, e.g., in the area of cloud-based database systems [6].

However, to date some opportunities of cloud computing have not been leveraged in BPM. Notable exceptions include, e.g., process modeling solutions such as Oryx built on the Software-as-a-Service (SaaS) model [7], and first proposals to deploy services utilized in workflows on infrastructure resources offered in the Infrastructure-as-a-Service (IaaS) model [8, 9]. But in the efforts, no baseline was established in order to quantify the potential.

This paper addresses the following research questions: (i) Is there a potential for optimizing BPM through BPM-aware horizontal scaling? (ii) If so, which contextual circumstances influence the amount of possible gains?

We develop a conceptual frame of context to be considered, formalized as deployment architecture meta-model. We then experimentally quantify the magnitude of the potential improvements based on a specific architecture.

In the following, we briefly introduce relevant foundations of cloud computing and BPM. Then we describe a model of cloud-aware business process management (Section 3). We use this model to elaborate on the case of process performance optimization using virtualized hardware in Section 4. Based on the analysis of process log data from a lab experiment, in Section 5 we identify optimization potential which exists because of unfavorable configuration of computational resources. This is done in an experiment where the auto-scaling mechanism of a major cloud provider is compared with manual, process-aware scaling. Subsequently, we discuss related work and summarize our conclusions.

2. Foundations

After outlining core concepts from BPM, we will give a brief overview of cloud computing.

Business processes are generally seen as collections of tasks performed within or across companies or organizations in a particular order [10]. A task in a process can either refer to work which needs to be done manually, or to work which can be done automatically. One task's output often serves as input for another task. An instance of a process and its context always have a particular state.

BPM refers to a collection of methods and tools for achieving an understanding of, as well as for managing and improving an enterprise's process portfolio [11]. BPM can be applied in planning, controlling, and monitoring intra and inter-organizational processes with regards to existing operational sequences and structures in a consistent, continuous, and iterative way of process improvement [2]. A BPM system (BPMS) allows for the definition, execution, and logging of processes.

Cloud computing enables protected access to a shared pool of configurable computing, storage, and networking resources as well as applications, which can be tailored to the consumer's need. Cloud resources can be rapidly provisioned and released, and are billed based on actual use – thus reducing initial investment costs [5]. Cloud computing does not make other software or resource provisioning paradigms obsolete, but outlines how infrastructure, platforms, and software can be commoditized and made available as a service rather than as bespoke or product. The application of cloud computing for enterprise resource planning, and thus BPM, is a growing market [12].

Cloud computing usually distinguishes Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) layers [13]. SaaS is an approach to make software ubiquitously available as a service. PaaS provides development components and deployment platforms to architects or developers as a service so that they can engineer applications based on these. Finally, IaaS enables users to access virtual resources via a network. This can include storage, servers, or network components and virtualizes physical infrastructure. We assume using an IaaS infrastructure on which automated task implementations are deployed, since it allows deeper access to monitoring data and more control over performance-relevant options.

3. Deployment Architecture Meta Model

In this section, we provide a deployment architecture meta-model which we use as the basis of our research on cloud-aware BPM. As such, it describes which object types are of relevance, and which relations may exist among them. We start with the definitions for virtual machine instances and systems.

Definition 1: *We define a virtual machine instance as a tuple* $vmi := (cs, mi, loc, DSys)$*, where* $cs \in CS$ *is a compute service from the set* CS *of available compute services,* $mi \in MI$ *is a machine image from the set* MI*,* $loc \in LOC$ *is a location from the set* LOC*, and* $DSys = \{dsys_1, dsys_2, \dots\}$ *is the set of systems directly deployed on* vmi*.*

Definition 2: *We further define the set of systems* $SYS = \{sys_1, sys_2, \dots\}$*, where* $sys \in SYS$ *is a named system (like "Apache Tomcat"). A deployed system* $dsys \in DSYS$ *is defined as a triple* $dsys := (sys, v, DSys)$*, where* $sys \in SYS$*,* v *is a version number, and* $DSys$ *is the set of deployed subsystems contained in* $dsys$*.*

For instance, a Web Application Server (WAS) can contain a Web Service (WS) container, which can contain various WSs. Or a WAS can contain a BPMS, or a BPMS can be stand-alone. SYS is the set of system names in the model, and $dsys$ is a concrete deployment of a system within some vmi or other deployed system $dsys'$.

In the following, we refer to process models as instances of a certain class of Petri-Nets, called Workflow-Nets or WF-Nets [14][1]: a Petri-Net with two special places (p_{start}, p_{end}), where p_{start} has no incoming and p_{end} no outgoing arcs, but adding a transition t^* from p_{end} to p_{start} would make the net strongly connected. In the following, we assume that a WF-Net is both free-choice and sound.

Definition 3: *A BPMS deployment is a specific deployed system, and has a certain context, i.e., its configuration parameter settings. A deployed process model* $pm \in PM$ *is defined as a WF-Net with additional deployment data:* $pm := (P_{pm}, T_{pm}, A_{pm}, p_{start}, p_{end}, D_{pm})$*, containing a set of places* P_{pm}*, a set of transitions* T_{pm}*, a set of arcs* A_{pm}*, one start place* p_{start}*, one end place* p_{end} *– all of the above with the usual meaning in WF-Nets – and* D_{pm} *being its deployment data. A process instance* $pi \in PI$ *then is a marking for a given* pm*, captured by mapping* $\mu: PI \rightarrow PM$*, along with process instance context (variables, etc.), and runs inside a specific BPMS.*

Deployment data D_{pm} is all the information that is needed to execute a process, which is not part of the process definition itself. If the process uses WS technology, D_{pm} usually includes WSDL interfaces. If the process uses forms for human tasks, D_{pm} may include representations of these forms.

Definition 4: *A task* $t \in T$ *from the set of tasks* T *is a unit of work. We define a task-process model mapping from the set of tasks to a subset of all places* $tpm: T \rightarrow 2^{\cup_{pm \in PM} P_{pm}}$*, where* $p \in tpm(t)$ *means that place* p *is a place where task* t *is executed. Specific types of tasks are human tasks* $T^h \subset T$*, automated tasks* $T^a \subset T$*, and third-party services (automated or otherwise)* $T^{3p} \subset T$*, with* $T^h \cup T^a \cup T^{3p} = T$ *and* T^h, T^a, T^{3p} *being pairwise disjoint.*

A task instance $ti \in TI$ *belongs to a specific task and a specific process instance, as captured by the projection* $\pi: PI \times T \rightarrow TI$*, where* $\pi(pi, t) = ti$ *implies that* $\exists p \in tpm(t),\ p \in P_{pm}$ *and* $\mu(pi) = pm$*.*

Finally, a set of tasks (human or automated) can be supported or implemented by a deployed system, as expressed by mapping $\rho: DSYS \rightarrow T^h \cup T^a$*.*

The above definition allows us to reason about the tasks in a process, as well as the systems and infrastructure supporting / implementing a task.

Definition 5: *We define a generic mechanism to express attributes of artifacts, as a mapping from an artifact* x *to a list of key-value pairs:* $x \rightarrow \{(k_1, v_1), (k_2, v_2), \dots\}$*. We require that any* $k_i = k_j \rightarrow i = j$*, i.e., the keys are unique in a given list. Artifact is hereby any named entity defined above, e.g.,* $vmi, cs, \dots$ *. Further, every artifact must have an attribute* status*, and its value must be one out of a defined list of allowed status values for this artifact type.*

Attribute lists allow us to track, e.g., the Linux kernel version, or the amount of memory or CPU type used by a given vmi. Important attributes also include the process model version or input / output of a task.

4. Process Performance Optimization based on Cloud Resources

Automated tasks T^a in business process models pm are often implemented as Web services, deployed on a particular system $dsys$, and called from a BPMS. The reuse of an automated task, respectively a Web service implementation, generates workload which grows with the number of business processes models and, in particular, with the number of instances of the processes. However, the workload may also differ depending on its input – e.g., document size in the running example of data transformation.

In general, scalable Web services, stateless or stateful, are capable of adjusting to an increased workload. Elasticity features of cloud infrastructure services, i.e., on-demand provisioning of additional infrastructure resources which are perceived to be available in unlim-

[1] Note that other process modeling languages can be substituted for WF-Nets.

ited numbers at a constant price, are a natural fit for Web services. Using these features, a system deployment can be adapted to growing demand within few minutes. However, automated provisioning and deprovisioning of infrastructure resources based on demand requires control. Some providers of compute clouds offer transparent control mechanisms. In many cases, these mechanisms rely on reactive threshold-based reasoning for infrastructure changes, based on monitored system-level metrics of instances. Because provisioning times for infrastructure resources must be taken into account, SLA violations can occur under rapidly increasing workloads. Different approaches account for provisioning times by applying methods for workload forecasting, requesting the provisioning of additional resources in advance. In many cases, the identification of reliable and accurate workload forecasting models and metrics is far from trivial [15].

We suggest using existing, accessible knowledge in the BPMS about the workload for a process model pm, as well as current turnaround times for process instances, to provide more reliable and accurate information for the provisioning of infrastructure resources. For deployed Web services $dsys$ triggered by an automated task T^a of a business process model pm, forecasting logic can profit from data of business process instances pi. For instance, the status of currently active tasks ti in process instances pi can be used to derive upcoming workload of later tasks.

Such data can be provided by a BPMS which manages and monitors business process instances pi. With an extension to currently available BPMSs to provide an interface for Web service elasticity algorithms, both Web services and process instances can benefit. Web services' workloads can be forecasted and resources can be adapted to changing demands. Business processes can reduce the total processing times of automated tasks so that the processes can fulfill the negotiated SLAs, since Web services are more likely to have sufficient resources available at the time needed.

To allow forecasting of future workloads on Web services associated with particular business process tasks T^a, a forecasting elasticity algorithm must know the timely status of the following to be able to analyze and decide on actions:

- Which process models pm are deployed?
- Which automated tasks T^a are executed by pm?
- Which tasks T^{a_n} follow T^{a_m}?
- Which process instances pi are active?
- How many task instances ti are executed for process activity T^a and what is their resource requirement x?
- Which set of vmi execute T^a, captured by ρ?
- What is the utilization of the available vmi for each task T^a?
- What is the utilization of the available vmi for the preceding tasks T^{a_m}?
- What is the provisioning time before a new vmi is available for T^a?

Given the information about aforementioned status parameters, a forecasting elasticity algorithm needs a decision-making method to derive an action which either retains the current number of vmi providing a Web service $dsys$, or adds or removes additional instances. The number of process instances pi proceeding to an activity T^a as well as the timeframe in which this will happen can be estimated by observing the behavior of the preceding tasks T^{a_m}. The workload for the first task of a process model pm can only be calculated based on the queue of scheduled but not yet started process instances pi.

Scaling in the machine instances underneath a stateful Web services also requires modification, as some vmi cannot simply be shut down without risking the loss of data or raising faults in a process instance pi execution due to a non-responding Web service. In the running example, a vmi may be in the process of translating an XML document, but be shut down after a scale-in decision. Doing so will result in a timeout of the process instance pi waiting for an answer. Hence, a vmi shutdown must be scheduled, and potentially it may be redeemed in a future upscaling action.

A prerequisite derivable from that observation is an extension to stateful Web services to approach a status in which they can be shut down. A stateful Web service thus must signal that it does not accept further invocations, e.g., by deregistering it from the respective load balancer. Current vmi state models do not necessarily support this behavior. If this behavior is not supported by the vmi state models and load balancers, the BPM-aware controller needs to take care of this management. This can be done by deregistering a vmi so that it does not receive new requests, then to monitor when all existing requests have been fulfilled, and adding the vmi to a list of machines to be terminated. Just before the payment cycle ends, the machine is shut down. Should in the meantime the demand rise again, then the vmi can be removed from said list and be re-registered with the load balancer.

Note that our current implementation does not include an automated process-aware controller. While this is the obvious direction for our future work, herein we aim to assess the potential of having such a controller, and in our experiment one of the authors made the scaling decisions manually based on a subset of the aforementioned information. Our implementation and experiment are described next.

5. Implementation Architecture

To test our hypothesis, we implemented the test bed shown in Figure 2 as follows. The process has been modeled as a composition of three services, and de-

ployed to a BPMS. We use Intalio|Server[2] herein. The three tasks are implemented as Web services which simulate CPU-intensive loads, where the exact load is dependent on the input. Specifically, given parameter *n* as input, these placeholder services compute the factorial of each number between 1 and *n*.

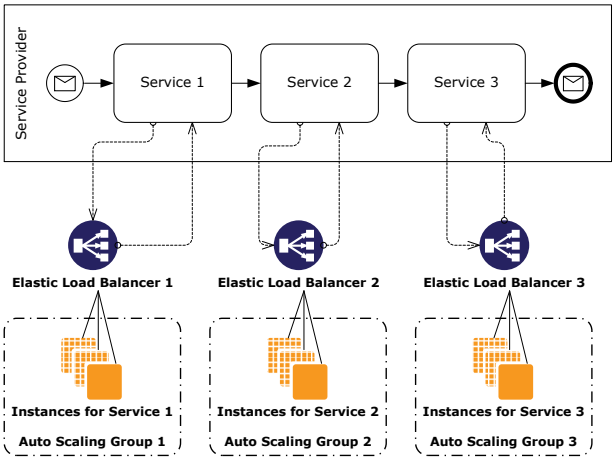

Figure 2. Implementation architecture

We use AWS as the IaaS provider. The services are deployed and configured within an Amazon Machine Image (AMI). An AMI is a pre-configured image of a VM, which is stored in AWS EC2. A *vmi* on EC2 can be created from these images in an elastic manner.

Hence, new (stateless) instances providing the respective service can be started without further configuration: once fired up, a new *vmi* can directly start serving requests. This facility allowed us to define Auto-Scaling Groups (ASGs) for each service, with appropriate scaling policies: between 1 and 5 instances per service are active; if the average CPU load is below 40 % for five minutes, one instance is terminated (if >1 is active); if the average CPU load is above 70 % for five minutes, one new instance is started up (unless 5 are already active). All three ASGs are configured to add/ remove their instances to/from a respective Elastic Load Balancer (ELBs). The ELBs distribute requests to the available machines within an ASG.

The same setup (ELB & ASG) could theoretically be chosen for the BPMS, but (i) that was not necessary since our experiments had low CPU load on the BPMS, (ii) BPMS machines cannot be left to be terminated by the ASG if there is no CPU load: among other reasons, there might be (currently suspended or idle) process instances which are running solely on one of the instances (application workload ≠ *vmi* utilization). Furthermore, BPMSs are not stateless.

In order to evaluate the relevance of our proposal, we conducted a lab experiment using our architecture with two workload scenarios and two scaling setups. The main goal is to assess if significant differences in process execution can be observed, i.e. turnaround time and/or resource utilization, for the different setups.

Assumptions. Given a setting where turnaround time cannot be optimized any further through the reduction of wait time, change-over time, or suspend time, we can only optimize the performance by changing the actual processing time. We assume a suitable setup, where the processing time can be increased or decreased by providing more or less computational cloud resources. Also, we assume that there is a linear relation between the processing times of tasks 1, 2 and 3: if task 1 has the duration of d, then task 2 has roughly the duration $a \times d$ and task 3 has roughly the duration $b \times d$ (for some values of a, b).

Also, we assume that the SLAs are concerned with the overall process, not with one of the tasks. This entails that we need to optimize the provisioning of computational resources (i.e. horizontal scaling) across multiple services invoked by stateful processes with regard to the overall process turnaround time rather than that of individual tasks. Further, the Web services to be called may in turn be distributed across other cloud computing providers so that local scaling and load balancing mechanisms do not necessarily yield optimal results, i.e., to stay (just) within agreed temporal boundaries of the SLAs for this process.

For the sake of the scenarios, we also assume that the virtual resources of the individual services are independent of each other – e.g., increased computational requirements of task 1 do not decrease the computational resources available to tasks 2 and 3. This entails that load balancing and scaling are managed for each task independently. Despite this separation into three ASGs, we assume that we have full control of the horizontal scaling of all three tasks.

Also, we assume that we can estimate the approximate capacity requirements of the latter tasks 2 and 3 once task 1 has completed, as 2 and 3 depend on the output of task 1. In the experiment, task 1 through 3 are identical and should exhibit similar resource requirements.[3] Hence, as opposed to the ELB and the auto-scaling mechanisms of the ASG, a cloud-aware BPM system can derive the required computational resources required especially for task 3 earlier. Note that this advantage will vanish if the task throughput is high – i.e., if the load predominantly arises from the *number* of process instances, not the *complexity* of a few process instances – in which case starting another machine instance may take too long to support a particular process instance.

Scaling Setup. We evaluate the difference between *auto-scaling through the cloud service provider* based

[2] http://bpms.intalio.com/

[3] The results will still vary however, as seemingly identical virtual machines on EC2 rarely behave identical but exhibit differences in performance due to the underlying physical hardware among other aspects – see, e.g., [16].

on its observation of resource requirements (setup 1: standard auto-scaling (*std-as*)) and s*caling based on process knowledge* (setup 2: process-aware auto-scaling (*pa-as*)).

For *std-as* we used the auto-scaling rules presented in the previous section, i.e., per ASG, scale in if the CPU load is below 40 % for five minutes; scale out if the CPU load is over 70 % for three minutes.

The *pa-as* setup is simulated by manually applying the following policy. A 3-minute sliding average of process instance turnaround times is considered, as well as the current number of active machines (i.e., receiving requests) and the *backlog* of currently active pi. If this indicates that each machine works more than 70 % of the time, more machines were requested; at less than 40 %, less machines were needed. However, before starting more machines, pa-as checks if additional machines were started but are not yet active. This provisioning delay is discussed below.

Workload Generation. For varying the workload, there are two basic options of simulating load changes: (i) changing the amount of process instances per time unit and (ii) changing the computational requirements of tasks of the process. Also, the two can be blended.

We decided to use the option (ii), since an increase in the number of process instances would have side-effects, as follows. The BPMS may slow down due to increased logging and scheduling activities. Also, network latency may increase and adulterate the measurements. Hence, we vary the input data such that the computational requirements were increased or decreased.

We use JMeter[4] as a workload generation tool. For each complexity value, we start six process instances per minute over five minutes. Then we change the complexity and trigger the next 6×5 instances. While this is a very deterministic setting, it enables us to observe other seemingly randomized parameters such as scheduling and load balancing without further noise.

Scenario 1 – Workload Increase. In this scenario, we increase the load on the process in a uniform way. This is depicted in Figure 3 as a solid line. We initialize the system by simulating a constant flow of the new process instances which utilize the three auto-scaling groups with one VM each at about 70 % CPU. Once this has evened out, we increase the load on the process every five minutes, as described above. In std-as, we expect the auto-scaling to kick in at some point on all three auto-scaling groups. In pa-as, we trigger the scaling of task 2 and 3 so that the relative resource provisioning of task 2 and three 3 mirrors the scaling of task 1 delayed individually by the expected processing times of the prior tasks and the virtual machine's provisioning time.

Scenario 2 – Workload Decrease. In this scenario, we decrease the load on the three tasks of the process in a uniform way – see the dashed line in Figure 3. We initialize the system by simulating a constant flow of the new process instances which utilize the three scaling groups so that each consists of 5 virtual machines at about 40 % CPU. Once this has evened out, we decrease the load on the services as described. In std-as, we expect the auto-scaling to kick in at some point on all three auto-scaling groups to sequentially scale in to one virtual machine each. In pa-as, we trigger the scaling of task 2 and 3 so that the relative resource provisioning of task 2 and three 3 mirrors the scaling of task 1 delayed individually by the expected processing times of the prior tasks.

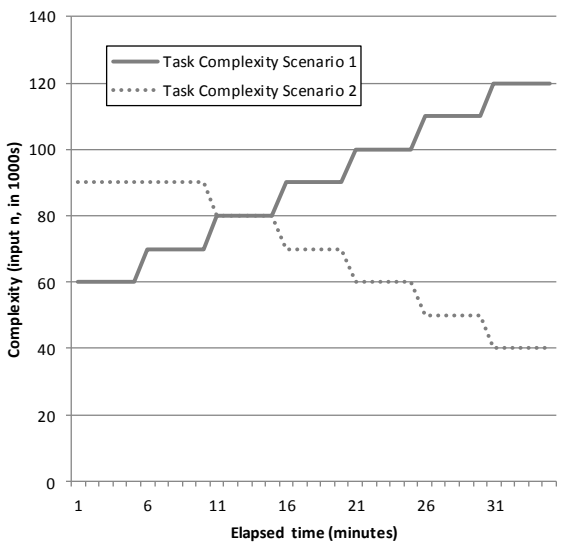

Figure 3. Workload scenarios

At this stage, we focus on the utilization of the virtual machines to determine an optimal point in time to provision and de-provision more compute instances. As introduced above, the optimal point in time to adapt resources is not only determined by the utilization of computational resources, but by the cost of the resources as well as the margins of the SLAs for a process. Hence, it may be useful to not scale at all, if there is no limit the process turnaround time (e.g., on a free service). On the other hand, it may be very economical to overprovision if the compensation for a single SLA violation is very expensive. At this stage, we have not integrated these options into this first evaluation, which rather serves as motivation for further research into cloud-aware BPM.

6. Evaluation and Discussion

While the scenarios and setups are seemingly clear and straightforward, there are a few points to note when actually performing this laboratory experiment:

(i) ELBs in AWS have a default connection timeout of 60 seconds, which is too low for relevant task turnaround time in our experiment. Since it is not a configu-

[4] http://jmeter.apache.org/.

ration option that is available to customers, we had to ask Amazon support staff to increase this value (the maximum is 17 minutes, which we chose).

(ii) New VMs needed about 7 minutes, from requesting one until it is available. This time is split between AWS's provisioning time, boot time, and a safety margin delay before new machines are made available by the ELB. All commands also have a delay of ten to twelve seconds – see also [17].

(iii) std-as terminates VMs when the average CPU utilization in an ASG is low. The terminated machine might, however, still work on executing a task instance. std-as has a short grace period, but does not wait until all requests completed.

For scenario 1, Figure 4 provides an overview of results for pa-as vs. std-as, concerning the total number of available machine images vmi for all three tasks, the number of active process instances pi, and average turnaround time (of pi finished at this point in time). Note that, while the same scale is used, the units vary. Gaps in the graph indicate that no pi completed in the particular minute, an effect of all measures being aggregated per minute. Note further, that after minute 35 no new process instances were triggered by the workload generator.

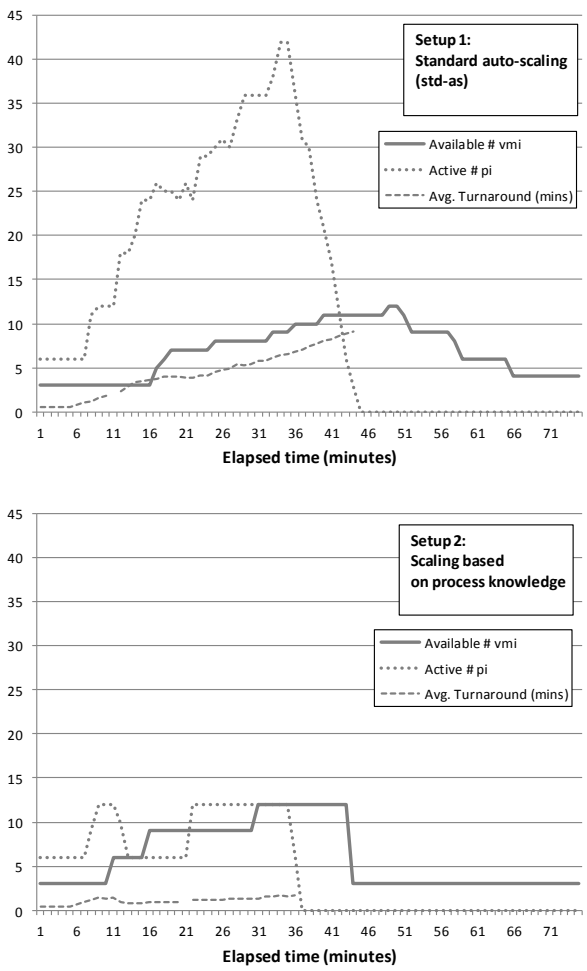

Figure 4. Scenario 1: std-as vs. pa-as

In this scenario, std-as scaled out later than pa-as, as it has no access to knowledge about the process flow or process/task turnaround times. This was not the only aspect in which pa-as was superior to std-as: at minute 49, std-as's eleventh vmi became active, but was never used. Further, std-as built up a considerable backlog of up to 42 active pi. The maximum turnaround time thus was 9:07 minutes, with an average of 4:23 minutes.

For pa-as, we observed a much smoother process execution. Due to the process knowledge, the scaling was quicker and better aligned with the requirements of the process input complexity. The backlog did at no time exceed 12 process instances. The maximum turnaround time was 1:37 minutes with an average of just 1:02 minutes. This was achieved at the additional cost of a twelfth vmi. In general we also observed that pa-as stayed within the target CPU corridor of 40 % to 70 % usage for most of the time, in contrast to std-as.

As expected, the results of scenario 2 (decreasing workload) did not exhibit as much difference as scenario 1. pa-as scaled in faster than std-as, but apart from one outlier, the maximum turnaround time was similar. std-as had a process turnaround average of 39.9 second, the average turnaround of pa-as was 39.5 seconds. Due to the faster scale-in, the CPU utilization of pa-as was generally higher but at no time impacted the task's processing speed. It was also below the scale-out threshold at all times. Neither setup caused a significant backlog. Both options scaled in from 15 to three vmi as expected.

However, it is important to note that in the specific case of Amazon EC2, billing for vmi is on an hourly basis. Hence, it may not be economical to scale in at an arbitrary point in time since the resources need to be paid for nonetheless. So instead of shutting down vmi, we option to shut them down and they are only terminated at the end of a billing cycle (i.e., at 0:57h, 1:57h, 2:57h, etc. since there is a two minute slack) if there is no new increase in workload. Obviously, AWS may reduce the billing frequency, so that one only pays for the number of minutes a vmi was active. In this case, the above consideration would be rendered void.

The results of this experiment show that even a rather simplistic scaling strategy based on process knowledge can provide significant benefits over a standard auto-scaler. While scenario 2 showed that CPU utilization can be improved and integrating knowledge about the economy of vmi billing can make a difference, the benefits in scenario 1 were more striking: it was possible avoid extensive queuing and reduce the maximum turnaround time by a factor of 5,5 and the average turnaround time by a factor of 4,2.

7. Related Work

The integration of BPM and cloud infrastructure is in its infancy. Currently, there is no agreed upon architecture or evaluation available. Amziani et al. [18]

propose a generic framework to structure horizontal scale-out and scale-in decisions for services within a workflow and conditions for the routing of requests within workflow instances. However, it is unclear how scaling strategies can be specified or learned.

Schulte et al. [8, 9] propose the Vienna Platform for elastic processes that controls the provisioning of vmi that host application containers and the deployment and replication of stateless services over application containers based on reasoning on near real-time data. Different design decisions for the implementation of a reasoning mechanism are discussed on an abstract level. It is not shown if and how knowledge about workflow models can be utilized to improve effectiveness and efficiency of such reasoning mechanism.

Marshall et al. [19] present Elastic Site, which includes a model and implementation of a resource management layer that controls the number of vmi of a horizontally distributed batch processing application based on job submission patterns. Three management policies are presented with a number of metrics to evaluate and compare management policies. The work is complementary to our work, as it shows how to control vmi numbers of a single Web service. We propose to utilize knowledge about workflows to coordinate the control of vmi numbers over a set of Web services.

Byun et al. [20] propose PBTS (Partitioned Balanced Time Scheduling) to determine the minimum number of hosts (vmi) to execute workflows under user specified time constraints. The approach is based on the idea to delay the execution of tasks that are not on the critical path of execution time of a workflow. The approach requires accurate forecasting of execution times of tasks and assumes that different tasks can be executed on the same vmi.

Dejun et al. [21] propose to analyze directed acyclic invocation graphs for making scaling decisions. Hereby each service communicates to the central controller how adding or removing a machine instance would impact its performance (or that of its child nodes). One weakness of this approach is that scaling decisions are made only in increments/decrements of 1, and the effect needs to be observed before a next decision can be made. As can be seen from Figure 4, this would have likely performed poorly in our experiment – given the delay of 7 minutes before a scaling decision's effect becoming visible. Furthermore, it is unclear as yet if the technique could be adapted to BPM scenarios with potentially complex control flow.

Dörneman et al. [22] present an approach to control the scheduling of tasks of BPEL workflows upon a set of vmi of controlled size. In contrast to our approach, knowledge about workflows is not utilized to improve control decisions.

8. Conclusion and Outlook

This paper is concerned with research questions around potential benefits by using BPM-aware scaling mechanisms: do such benefits exist, and if so, in which contexts?

We have argued that cloud computing provides a basis for business process optimization since the turnaround time of process tasks is directly influenced by the amount of provisioned computational resources. Hence, if a task cannot be implemented more efficiently, its execution can only be influenced by the underlying hardware. Optimization on the task level can be translated into benefits on the process level in scenarios where the turnaround times mainly depend on the actual processing time of tasks as opposed to wait time, change-over time, etc. Enterprise integration scenarios fall into this category. So do scientific workflows – however, their many specific requirements (such as adaptability) demand further investigation.

From an economic point of view, if managed well BPM-aware cloud resource control enables the process owner to manage his process execution in such a way that processes complete just within the negotiated SLA. From a technical point of view, this enables the process owner not to commit to a decision for physical hardware in advance but to defer the decision to the actual point of process execution. Furthermore, knowledge about the process flow and the resources used for preceding tasks enable a cloud-aware BPMS to plan horizontal scaling in advance.

We have performed a lab experiment to exemplify this potential. Already in rather simple scenarios, we have encountered several obstacles for cloud-aware BPM such as timeouts and the termination of virtual machine instances still processing data. Nevertheless, we were able to show that making use of process knowledge to influence horizontal scaling of resources can improve the overall turnaround time. Furthermore, if the process tasks were executed on physical hardware with no option to scale horizontally, many process instances might have failed due to timeouts or other errors due to extensive queuing.

As a limitation, it has to be noted that our lab experiment only sheds light on a very confined aspect of elasticity in cloud computing. We have increased the computational requirements of a process object for three tasks in a uniform way without increasing the frequency of new process instances. In a real world example, load changes usually coincide with an increase of throughput, which in turn leads to other side effects such as increased utilization of the BPMS due to logging or scheduling or simply to an increase in latency due to network traffic. To abstract from these effects was a deliberate decision in order to concentrate on one aspect of horizontal scaling rather than trying to capture all aspects.

This research should be seen as a primer for cloud-aware BPM and its integration in future BPMS or cloud computing offers. As we have pointed out in the discussion, leading cloud computing service providers have not configured their virtual machines in a way that Web services for long running and stateful pro-

cesses can be provisioned out of the box. Also, the standard horizontal scaling mechanisms which are provided do not take into consideration observations originating from preceding tasks in the same process model.

In future work, we plan to provide process-aware controllers that can autonomously learn the behavior of the environment. As such, load levels, backlog, current configuration, and SLA fulfillment should be taken into account when making scaling decisions. Furthermore, we plan to employ formal models that can relate usual process flow in terms of decisions, repetition, or concurrency, to scaling decisions of individual services. We believe that such a system can achieve maximal fulfillment of SLAs and other desired properties while minimizing resource usage and hence cost.

Acknowledgements

The project was funded by means of the German Academic Exchange Service (DAAD) under the promotional reference “54392100” and by the Group of Eight (Go8) Australia-Germany Joint Research Cooperation Scheme. The authors take the responsibility for the content. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] M. zur Muehlen and R. Shapiro, "Business Process Analytics", In J. vom Brocke and M. Rosemann (eds.), Handbook on Business Process Management: Strategic Alignment, Governance, People and Culture, vol. 2, Springer, Berlin, 2010

[2] J. Becker, M. Kugeler, and M. Rosemann (eds.), Process Management: A Guide for the Design of Business Processes, 2nd edn., Springer, Berlin, 2011.

[3] M. Hammer and J. Champy, Reengineering the Corporation: A Manifesto for Business Revolution, HarperBusiness, New York, NY, 1993.

[4] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Pearson, Boston, MA, 2004.

[5] C. Baun, M. Kunze, J. Nimis, and S. Tai, Cloud Computing: Web-Based Dynamic IT Services, 2nd edn., Springer, Berlin, 2011.

[6] L. Zhao, S. Sakr, and A. Liu, "A Framework for Consumer-Centric SLA Management of Cloud-Hosted Databases", IEEE Transactions on Service Computing, PrePrint, 2013.

[7] G. Decker, H. Overdick, and M. Weske, "Oryx - An Open Modeling Platform for the BPM Community", In 6th International Conference on Business Process Management (BPM). Lecture Notes in Computer Science vol. 5240, Springer, Milan, 2008, pp. 382-385.

[8] S. Schulte, P. Hoenisch, S. Venugopal, and S. Dustdar, "Introducing the Vienna Platform for Elastic Processes", In 2nd International Workshop on Performance Assessment and Auditing in Service Computing Workshop (PAASC). Lecture Notes on Computer Science vol. 7759, Springer, Shanghai, 2012, pp. 179-190.

[9] S. Schulte, P. Hoenisch, S. Venugopal, and S. Dustdar, "Realizing Elastic Processes with ViePEP", In 10th International Conference on Service Oriented Computing (ICSOC). Demo Track. Lecture Notes on Computer Science vol. 7759, Springer, Shanghai, 2012, pp. 439-443.

[10] Object Management Group Inc., Business Process Model and Notation (BPMN) Version 2.0, 2011, http://www.omg.org/spec/BPMN/1.2/PDF.

[11] M. zur Muehlen and M. Indulska, "Modeling Languages for Business Processes and Business Rules: A Representational Analysis", Information Systems, 35 (4), 2010, pp. 379-390.

[12] L. Herbert and J. Erickson, The ROI Of Software-As-A-Service. Forrester Research, 2009, http://www.forrester.com/rb/Research/roi_of_software-as-a-service/q/id/53885/t/2.

[13] National Institute of Standards and Technology (NIST), NIST SP 800-145, The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology, 2011, http://csrc.nist.gov/ publications/nistpubs/800-145/SP800-145.pdf.

[14] W. M. P. van der Aalst, "Verification of Workflow Nets", In 18th International Conference on Application and Theory of Petri Nets (ICATPN). Lecture Notes in Computer Science vol. 1248, Springer, Toulouse, 1997, pp. 407-426.

[15] C. Loboz, "Cloud Resource Usage: Extreme Distributions Invalidating Traditional Capacity Planning Models", In ScienceCloud: 2nd Workshop on Scientific Cloud Computing, San Jose, CA, 2011, pp. 7-14.

[16] J. Dittrich and J. Quian, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance", Proceedings of the VLDB Endowment, 3 (1), 2010, pp. 460-471.

[17] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers", In 10th ACM SIGCOMM Annual Conference on Internet Measurement (IMC), Melbourne, 2010, pp. 1-14.

[18] M. Amziani, T. Melliti, and S. Tata, "A Generic Framework for Service-based Business Process Elasticity in the Cloud", In 10th International Conference on Business Process Management (BPM). Lecture Notes in Computer Science vol. 7481, Springer, Tallinn, 2012, pp. 194-199.

[19] P. Marshall, K. Keahey, and T. Freeman, "Elastic Site: Using Clouds to Elastically Extend Site Resources", In 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), Melbourne, 2010, pp. 43-52.

[20] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost Optimized Provisioning of Elastic Resources for Application Workflows", Future Generation Computer Systems, 27 (8), 2011, pp. 1011-1026.

[21] J. Dejun, G. Pierre, and C.-H. Chi, "Autonomous Resource Provisioning for Multi-Service Web Applications", In 19th International World-Wide Web Conference (WWW), Raleigh, NC, 2010, pp. 471-480.

[22] T. Dörnemann, E. Juhnke, and B. Freisleben, "On-Demand Resource Provisioning for BPEL Workflows Using Amazon’s Elastic Compute Cloud", In 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), Shanghai, 2009 pp. 140-147.