Scalable Business Process Execution in the Cloud

Sven Euting, Christian Janiesch, Robin Fischer, Stefan Tai
Institute of AIFB
Karlsruhe Institute of Technology
Karlsruhe, Germany
e-mail: sven.euting@student.kit.edu, janiesch@kit.edu, robin.fischer@kit.edu, tai@kit.edu

Ingo Weber
Software Systems Research Group
NICTA
Sydney, NSW, Australia
e-mail: ingo.weber@nicta.com.au

***Abstract*—Business processes orchestrate service requests in a structured fashion. Process knowledge, however, has rarely been used to predict and decide about cloud infrastructure resource usage. In this paper, we present an approach for BPM-aware cloud computing that builds on process knowledge to improve the timeliness and quality of resource scaling decisions. We introduce an IaaS resource controller based on fuzzy theory that monitors process execution and that is used to predict and control resource requirements for subsequent process tasks. In a laboratory experiment, we evaluate the controller design against a commercially available state-of-the-art auto scaler. Based on the results, we discuss improvements and limitations, and suggest directions for further research.**

Keywords—Cloud Computing; Business Process Management; Fuzzy Control; Elasticity

I. INTRODUCTION

The performance of process execution in business process management (BPM) is commonly optimized through organizational redesign, improved implementations of the services, which constitute the process, as well as the elimination of media discontinuity. BPM systems traditionally use a fixed (and thus limited) amount of computing resources for executing processes. However, the performance of processes also depends on the actual processing time within the execution environment, i.e. the infrastructure on top of which the services underlying the process are running. This is specifically the case for enterprise integration scenarios which are automations of data transformation or lookup operations. Examples include – but are not limited to – cases from order management, master data management, and address verification [25].

With the advent of cloud computing, however, BPM systems can benefit from the full potential of scalable cloud resources and use as many or as few resources as actually needed. In this paper, we address cloud-based BPM systems by introducing a novel BPM-aware IaaS resource controller.

Business processes orchestrate tasks as service requests in a structured, procedural fashion. This knowledge about execution sequence is valuable additional data for predicting resource demand, which current infrastructure scaling mechanisms do not take into account. Hence, pre-existing knowledge present in BPM systems can be used to enable scalable process execution in the cloud. In a manner of speaking, if task 1 requires the amount x of computing resources at t_1, we can assume ahead of time that the subsequent task 2 will require the amount y at t_2, and that task 3 will need the amount z at a future point in time t_3. The relation between tasks in the process can be exploited to infer relative performance requirements of the individual tasks and – taken together – on the overall process performance. A key research question is thus: *if the demand for a process changes (e.g., frequency/ complexity of requests), how should the underlying cloud resources be scaled to meet this change?* For instance, if an SLA specifies that 90 % of process instances should complete in three minutes or less and the average time between requests is cut in half, how many virtual machines are needed for task 1?

We suggest using process knowledge to improve infrastructure resource controllers through the process-based alignment of available computing resources with the respective demand. This paper makes the following contributions[1]: (i) an automated controller for process-aware auto scaling to align cloud resources to a processes' computational demand and (ii) a comparative evaluation of this controller. In particular, for (i) we propose a process-aware IaaS resource controller based on fuzzy theory; for (ii) we compare the controller's performance against a standard auto scaler in a laboratory experiment.

In the following we introduce the fundamentals and related work (Section II) and derive requirements for the BPM-aware controller from a process, resource, and controller perspective (Section III). Section IV presents the design of the overall controller as well as its decision making and execution processes. The evaluation in described in Section V. Finally, we discuss the findings and close with an overview of possible future work.

II. FUNDAMENTALS AND STATE-OF-THE-ART

A. BPM, Cloud Computing, and Fuzzy Theory

BPM refers to a collection of tools and methods for understanding, managing, and improving an enterprises' process portfolio [1]. While BPM is not limited to computer science, our focus is on run time execution of processes – often in the form of instances of a process model. The technical

[1] In a previous publication [21], we discussed the problem at hand, proposed a meta model to capture the various entities involved, and performed a manual experiment to show that process-aware scaling can yield benefits. Here we are proposing and evaluating an automated system to this end.

means to support BPM are provided by a business process management systems (BPMS), which "… allows for the definition, execution, and logging of business processes" [2].

Cloud computing "is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [3]. Two of the main characteristics of the cloud computing model are, according to [3], on-demand self-service and rapid elasticity, i.e., the option to acquire or release a virtually unbounded number of resources as needed. In cloud computing, three basic service models are distinguished [4]: Software (SaaS), Platform (PaaS), and Infrastructure as a Service (Iaas). The service models differ in terms of abstraction and the control that the service consumer has over the cloud resources used. In particular, the fine-grained scaling control required in this work is often only provided on the IaaS layer. Business Process Management as a Service (BPMaaS) is a term that is sometimes used to denote a cloud-based BPM SaaS offering.

In control theory, *controllers* are classified into two groups [5]: open-loop (non-feedback) controllers and closed-loop (feedback) controllers. The former execute a predefined model without observing the system to be controlled; the latter continuously observe the system's state and choose their actions accordingly. Closed-loop controllers are required whenever uncertainty exists in the system to be controlled, i.e., when the system's state may change due to circumstances not influenced by the controller. In this paper, user demand and uncertainty of the cloud are such circumstances. The objective of a feedback controller is to operate a system at a desired point or level. Sensors monitor the condition of the system and actuators allow influencing the system should a deviation from the set point take place. A variety of IT applications nowadays implement control theory in order to improve products like email servers and databases [6], web servers [7, 8], and computing clusters [9]. A prerequisite for applying a closed-loop controller is that the system provides sensors and actuators – we will discuss what these are in the given context later in the paper.

A *fuzzy controller* is a special controller design building on fuzzy theory. The main characteristic of a fuzzy controller is that its control model is described in linguistic rules rather than mathematical equations. Similar to human reasoning, it allows for a certain degree of ambiguity. Therefore, automating an expert's knowledge or implicit know-how is facilitated. Cf. e.g., [10] for the fundamentals of fuzzy logic and [11] for a fuzzy controller's precise mode of operation.

B. Related Work

The conjunction of BPM and cloud-based resources is a recent topic and sparsely covered in research literature. Dustdar et al. use the term "elastic processes" to describe adaptive processes that are elastic in the dimensions resource, cost, and quality [12]. They propose a system that manages these multiple service objectives by allocating various resources dynamically. The service objectives are defined by a client in regard to cost and quality. Schulte et al. take on this concept and suggest an advanced BPMS that incorporates these properties [13]. Dörnemann proposes a workflow system in which a process engine and a load balancer are enhanced to provide automatic and dynamic resource provisioning [14]. These works differ from the approach taken in this paper in that they propose to modify or enhance self-contained systems. We follow an approach of providing a controller component bridging existent systems to utilize the potential of cloud resources for the timely execution of a business process.

The conjunction of computing systems and control theory has been researched more intensely. The extensive work of Hellerstein et al. includes a general approach on applying control theory to computing systems [5] and using fuzzy control to maximize profits in service level agreements (SLA) [15]. Yet it has been observed that the focus is more on self-contained systems rather than on distributed cloud-based systems as discussed in this paper. A lot of effort is put into optimizing performance of software systems from the inside with the help of integrated feedback controllers and with detailed knowledge of the system. Significantly less attention is given to external controllers connecting domains, especially BPMS and cloud infrastructure.

Dejun et al. [26] 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 one, and the effect needs to be observed before a next decision can be made. In our own prior work [21] we showed that the general approach taken in this paper is promising. However, we did not provide any automated controller; instead, a human controller manually made scaling decisions. In said prior work, we also argued that the technique of Dejun et al. [26] with increment/ decrement sizes of 1 would have likely performed poorly in our experiment. This is due to a comparatively long ramp-up phase of several 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 flows.

III. Requirements of a BPM-Aware Resource Controller

In order to design a system that includes a BPM-aware controller, we first consider the requirements stemming from the process, the resources, and the controller.

A. Process Requirements

Requirements from a process perspective are mostly driven by the business goal to meet a requested process quality. Efficiency considerations dictate to achieve this with as little resource consumption as possible. Taking a closer look at BPMS, three major influencing factors become apparent.

First, there is the aspect of the BPMS itself. Even with formal standards defining elements and execution semantics of business processes, the approaches of technical realization may differ considerably. The time consumed for supporting tasks such as logging reduce the time available for the actual processing of the subject-matter in a specific task of the process. A BPMS is expected to handle concurrency. It is common that a number of instances of one or more deployed processes are executed at the same time. Therefore, the strategy

for efficiently handling this concurrency such as scheduling of processes is a central feature of the BPMS implementation. Depending on the design and chosen techniques, BPMS may vary in their ability to store and manage a pool of running instances. Oftentimes, the overhead increases with the size of concurrent instances. In summary, the BPMS design principles realized in the system's implementation determine time spent within the execution of process instances.

Secondly, there is the aspect of the execution environment of the BPMS. The resources enabling the execution of such a software system play an important role with respect to the performance of the business process instances. The execution environment comprises the properties of the underlying hardware as well as peripheral resources such as shared usage of databases or network capacity. If these shared resources present bottlenecks the performance of the BPMS can be impacted. Software configuration plays an important role as well. Efficiency and stability is affected by configuration on various levels, including the operating system underlying the BPMS. Apart from these endogenous parameters the exogenous influences of the processes' input from the process client deserves further considerations. The input data can be variable in regard to data size, temporal aspects of arrival, and the complexity it embodies for process execution. Either of these aspects can have a negative effect on the timely execution of the process instances.

Thirdly, there is the specific character of concurrent process instances per process model. Process instances of a single model may differ due to variation of inputs and external circumstances (e.g., time). An important quality of service (QoS) constraint is the overall duration for process instances – also called *turnaround time*. This can, e.g., be set to a fixed maximal time, or such that a certain percentage of instances have to complete within a given time (such as "95% of instances have to complete within 5 minutes"). To stay within the promised turnaround time, one can try to compensate for slow tasks early in the process by speeding up later tasks. As the execution proceeds, the room for corrective measures naturally narrows. At any point in time, new instances can be started and existing instances may complete. The order of instantiation is not necessarily reflected in the order of termination. The order of completion depends on the (proprietary) strategy of the BPMS for scheduling among concurrent process instances and the capacity of supporting resources, such as servers providing an automated process task. Manipulating either strategy based on the progress and desired remaining process duration is an approach to aim at fulfilling a turnaround time-based quality constraint.

B. Resource Requirements

IaaS provide the computing resources necessary to complete the automated tasks of a business process. Horizontal scaling (i.e., adding more machines, as opposed to scaling vertically by adding more RAM, CPU, etc.) is achieved by using a load balancer distributing requests among customized virtual machine (vm)-instances.

In IaaS, the unit of scaling is virtual machines (one or more). The granularity of scaling and, thus, the effect of adding or removing one vm is determined by the variety of virtual machine types offered by the cloud service provider. We do not consider vertical scaling at this stage.

Another important aspect is the time consumed for scaling. Scaling-out is determined by boot time (starting up a vm and its specific software system) and registration time (the process of making the started vm-instance accessible). The scale-in strategy is determined by the way a vm-instance is terminated. Stopping a vm-instance not only results in almost immediate termination but also in the loss of volatile data. An orderly shutdown procedure is more time-intensive but allows for storing volatile data and completing running jobs. Besides functional aspects, monetary factors should be taken into account as well. Although cloud computing generally follows a pay-per-use model, the pricing models vary among the competing cloud service providers. For instance, Amazon Web Services (AWS) charges per started hour of a running instance, whereas other providers charge per minute. This is a non-functional factor influencing scaling decision.

Even though virtual machines are logically separate machines, they run on shared physical machines. It has been observed that the performance of vm-instances requested with equal parameters can vary [16, 17]. Since at the moment it is not possible to directly influence the neighborhood of a new vm-instance, this variability is accepted as given and not examined further.

C. Controller Requirements

Generally the distinguishing features of a controller include its objective, input variables, control model and the effect of its output.

The possible objectives of a BPM-aware resource controller are manifold and may be driven by business goals derived from processes. Naturally, they may be very context-specific. A simple objective is e.g. a minimum overall process turnaround time. More complex objectives include conditional or combined target values driven by factors like business requirements, compliance constraints, or SLAs.

Based on the inputs and the control model, the controller makes a scaling decision – i.e., in this case to scale-in, out, or do nothing. The potential controller inputs can be divided into general process parameters (e.g., number of concurrently active instances) and parameters specific to a given process instance (e.g., complexity of the required computation). Any accessible data indicating the computational requirements is regarded as a potential input, since it may contribute to making the optimal scaling decision. Yet, any additional input increases the complexity of the controller with potentially negative effects, e.g., low performance may prevent timely decisions and a high amount of complex rules can decrease maintainability. Thus, it is reasonable to reduce the number of controller inputs, e.g., by preprocessing elementary data. This comes at the risk of possible loss of precision. The challenge is to find the best level and method of abstraction exploiting a variety of necessary information and reducing the number of inputs as far as possible at the same time.

The control model of the controller describes how the input is transformed into an output. It can either be static or variable. In a static model the mechanisms for transformation are set in advance and stay in place until modified by an administrator. A variable control model is designed to automatically adapt over its runtime. The control model is required to take into account that a virtual machine is a discrete scaling unit and the reaction is not very prompt, as previously discussed. Depending on the

scale of application and the size of the virtual machines, the scaling units may be rather large. In conjunction with the discussed delay this is disadvantageous for a precise control.

An overview of the discussed factors is given in Table I.

TABLE I. FACTORS INFLUENCING REQUIREMENTS

Perspective	Factors
Process	Implementation of the BPMS Execution environment of the BPMS Rate and complexity of process instance requests Special character of concurrent process instances
Resource	Granularity of horizontal scaling Delay in scaling-out Consequences of scaling-in Price model Performance variance of vm-instances
Resource controller	Control objective Basis for scaling decision Control model

IV. CONTROLLER DESIGN

A. Overview

The basic idea of a BPM-aware resource controller is to imitate the behavior and considerations of a *process expert*. By process expert we mean a person responsible for the timely execution of instances of a process. It is the expert's task to continuously decide on the amount of resources needed to keep the overall process turnaround time close to a specific point. A human operator may approach the task as follows: before the process becomes operational, process key performance indicator (KPIs) are identified. These are the metrics defining the timely execution and they influence the amount of deployed resources. Once the process is operational, the expert decides on the appropriate resources periodically. Thus, the time of process execution is structured by control cycles. At the beginning of a control cycle, the expert starts to monitor the KPIs. At the end of a control cycle the monitoring results are analyzed and a resource decision is made and executed. The expert then waits for the change of resources to become effective and starts a new control cycle to determine the next resource decision based on the (possibly changed) resource situation. All these operations are to be taken care of by the resource controller.

This approach is implemented by a controller prototype based on a Mamdani fuzzy controller [18]. It monitors process KPIs for a period of time. The resulting data is preprocessed and evaluated by the fuzzy rules of the control model to determine how many vm-instances to add or to remove from the load balancer. This scaling decision is then executed. After the scaling execution has completed or timed out due to failure or excessive delay, the procedure is started again from the beginning.

B. Scaling Rules of a Mamdani Controller

Generally a rule consists of a premise describing the input from one or more sensors and a conclusion describing the output relayed to one or more actuators.

The control rule can only be useful if the occurring linguistic terms are defined. Therefore, all appearing terms from the set of terms of a linguistic variable have to be defined for the respective codomain by an associated fuzzy set. This is achieved in consultation with the process expert. It is important to note that equal linguistic terms may be used for different linguistic variables. The terms for each variable have to be defined for the variable-specific semantics and co-domain.

Generally, the rule-base for a Mamdani controller with multiple inputs and one output can be depicted as a matrix. For each rule there is a row $r \in [0, \dots, k]$. For each input variable $v \in [0, \dots, n]$ there is a column. An additional column is intended for the conclusion. Table II in Section V.C is an example for such a matrix.

The general structure of a control rule is the following:

$$IF\ I_1\ is\ P_{(r,1)} \dots AND\ I_v\ is\ P_{(r,v)} \dots AND\ I_n\ is\ P_{(r,n)}$$
$$THEN\ O\ is\ C_r$$

with $I_v\ is\ P_{r,v}$ as a term of the premise of rule r for input v, and $O\ is\ C_r$ as a term for the output of the conclusion of rule r. The rule's conclusion is applicable only if the situation described by the rule's premise is observed. Both premise and conclusion are defined using linguistic variables and linguistic values. In fuzzy theory, linguistic terms are mathematically represented in form of membership functions describing fuzzy sets. Hence, each linguistic term of each linguistic variable has a respective fuzzy set associated. $\mu_{(r,v)}$ is the fuzzy set describing the linguistic term $P_{(r,v)}$ of the premise and μ_r the fuzzy set associated with the linguistic term of the conclusion C_r.

To derive a control action from sensor input the fuzzy controller's control logic has to evaluate the rule base. This is done in two steps: (a) evaluate all rules individually and (b) aggregate individual results of the rules into a global result.

The first step is structured as follows: determine the grade of membership $\mu_{(r,v)}(x_v)$ with x_v being a crisp input variable for v. The resulting value represents the applicability of the single premise term $P_{(r,v)}$. The grade of applicability is computed for every term of the premise. Then the degree of applicability for the complete premise α_r is determined. Since the complete premise is a conjunction of the n premise terms, the overall applicability of the premise is identified as the minimum of the applicability of the respective terms: $\alpha_r = \min\{\mu_{(r,1)}(x_1), \dots, \mu_{(r,n)}(x_n)\}$.

The level of confidence α_r that the premise is applicable to the current input values is reflected in the result of the evaluation of rule r. Only if all elements of the premise are completely fulfilled, then the resulting fuzzy set of the rule r, $\mu_{result}(r)$, is identical to the fuzzy set $\mu_{(r)}$ associated with the linguistic term of the conclusion. Otherwise $\mu_{result}(r)$ is set to $\mu_{(r)}$ with an upper bound of α_r.

The result of the first step is a fuzzy-set $\mu_{result}(r)$ per rule. This fuzzy set is based on the conclusion of the rule r, yet embodies the level of confidence for the applicability of the premise α_r, as per above.

The second step takes all resulting fuzzy-sets that were individually derived from the respective rules and merges them into a single fuzzy-set. The resulting fuzzy-set μ_{output} represents the overall conclusion of all rules applicable to the current input. The fuzzy-set resulting from the evaluation of all rules is finally computed into a single value, which forms the controller's output.

C. Input Parameters for Scaling Decision

Currently, the scaling decisions are derived based on four input parameters obtained from the BPMS and the load balancer: *average complexity of started instances*, *average turnaround time of completed instances*, *change in process backlog*, and *number of active vm-instances*.

Average complexity of started instances provides an indicator of the dimension of the requested workload during the current control cycle. It is a value calculated from variables either populated during process execution or used for process initialization, i.e. instantiation. The value of this variable can be used to estimate (future) resource requirements. The *average turnaround time of completed instances* provides an indicator of the timeliness of process instances that terminated (successfully). This provides insight into whether the preceding resource situation allowed timely completion. Yet, looking at the instantiation complexity and completion time of process instances in a sliding time window does not provide sufficient information whether bottlenecks occur within the process. This information is obtained by calculating the difference between the number of completed instances and the number of started instances during a control cycle. This value forms the *change in backlog* and indicates whether the number of currently active process instances has increased or decreased. The *number of active vm-instances* indicates the amount of computational resources and is determined by the number of vm-instances actually in service behind a load balancer.

D. Scaling Execution

Starting and stopping vm-instances takes a certain amount of time, simply registering or de-registering a running vm with a load balancer is considerably faster. Also, a vm-instance may be charged in chunks such as by the hour counting from the start of the vm-instance. Economic considerations therefore dictate that a vm-instance should be stopped close to the end of the commenced computing cycle. Cf. Figure 1 for an overview of the scaling processes (a) in BPMN [19].

Scaling-in – Figure 1 (b) – is intended to reduce the number of vm-instances in service behind a load balancer by a certain number. This can be achieved by stopping or merely de-registering vm-instances. Stopping instances with remaining computing time is not efficient if a pre-charged computing period is not fully exploited. It is possible that at the end of the next control cycle the very same vm-instance is identified to be re-added. This would result in additional costs and loss in time. Also, if a vm-instance is stopped prematurely, ongoing tasks may not complete. De-registering a number of vm-instances allows for a more promptly execution of the scaling decision. The vm-instances removed from the load balancer may complete running tasks and may be kept idle for the remaining commenced computing period without additional costs. If additional vm-instances are needed before the cycle is over, any of the still running instances can be registered again. This can dramatically shorten the start-up time of vm-instances.

Therefore, the implemented scaling-in strategy for a given number n of vm-instances is as follows: first the vm-instances to be removed from the load balancer must be identified. Hence, the vm-instances behind the load balancer are retrieved and ordered by their respective remaining computing time. In order to use the computing cycle of each vm-instance to the maximum, the n instances with the least computing time left are chosen. These vm-instances are then de-registered from the load balancer. Afterwards, a new control cycle and a cleanup procedure are started. The clean-up procedure stops the unregistered and idle vm-instances expected to begin a new computing cycle before the end of the current control cycle.

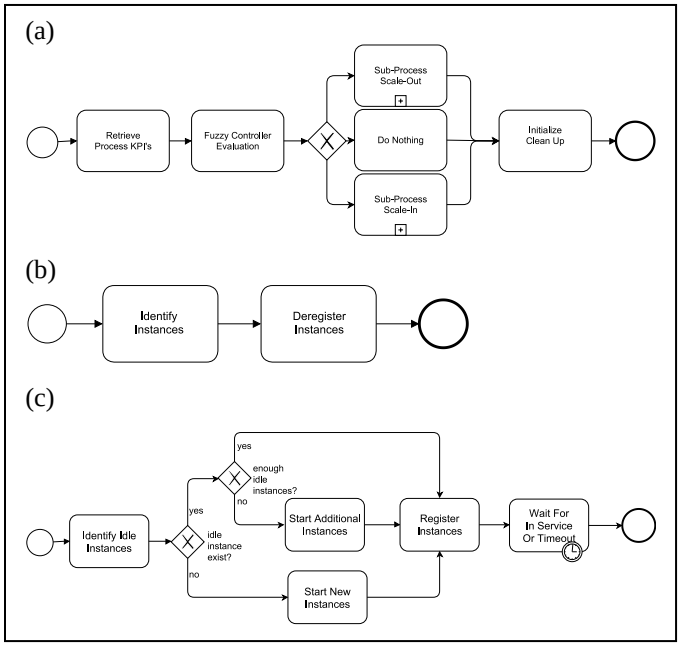

Fig. 1. Scaling Decision Execution Processes.

Scaling-out – Figure 1 (c) – is the task of increasing the number of vm-instances in service behind a load balancer, say by the amount m. Considering the previously described vm-instances, the process of scaling-out is as follows: Since registering idle vm-instances is preferred over starting and registering new vm-instances, the first step is to identify the set of running, yet idle and not registered vm-instances. We then take as many of the m required vm-instances as possible from this set and re-register them. If necessary, the number of additionally needed vm-instances is started concurrently. The procedure waits for all of these to be in service, yet no longer than a defined timeout period. Eventually, a new control cycle is started and the clean-up procedure is initiated.

The third option is that no scaling is needed, i.e., to not change the current distribution of vm-instances. The controller performs no action and eventually starts the next control cycle.

V. Experiments

A. Experiment Test Bed

To evaluate the design and implementation of the controller and benchmark its performance against a commercially available auto scaler we have implemented an experiment test bed.

In the test bed, a BPMS is deployed on a virtual machine. The executed processes invoke services also running on virtualized infrastructure in the cloud. Each service is implemented by an independent scaling group behind a load balancer. Figure 2 shows an overview of the test bed. Our BPM-aware auto scaler is used to horizontally scale vm-

instances based on KPI retrieved through the IaaS providers API as well as the BPM engine.

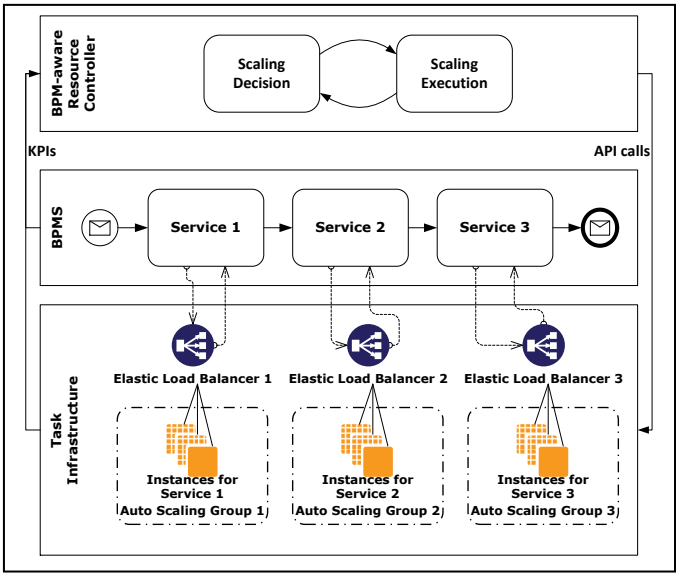

Fig. 2. Experiment Test Bed.

The BPMS manages the execution of process instances. We use the intalio bpms server[2], which is built on top of the open source Apache ODE BPEL process engine[3]. The system is deployed on an AWS Elastic Cloud Compute (EC2) [20] medium instance virtual machine running Windows Server 2008 located in Ireland. The process models are created using the intalio bpm designer, an Eclipse[4]-based BPMN modeling tool. Generally, intalio bpms provides both a user-oriented Web console and a SOAP/ WSDL interface for machine-to-machine interaction. Specifically, the instance management interface allows for starting, monitoring, and stopping specific process instances.

The cloud infrastructure provides the computational resources to complete the requests relayed by the process tasks. To provide horizontal scalability, two elements are needed: a load balancing mechanism and a vm-image containing the Web service along with its supporting software structure. The load balancer is a proxy representative of the Web service and therefore one load balancer is associated with each process task. The setup uses AWS Elastic Load Balancing (ELB)[5] to automatically distribute the requests originating from the process instances among vm-instances identical in function. ELB allows registering and removing vm-instances from the auto scaling group (ASG) which it uses to balance the incoming requests.

Additionally, the load balancer checks the state of the vm-instances in the ASG. In the test bed, the load balancer is configured to perform health checks every twenty seconds. The health threshold is set to two successive health checks. Therefore, if two successive health checks fail, the specific vm-instance is rendered out of service and no more requests will be relayed to this machine. If two successive health checks are successful, the concerned vm-instance is considered in service and receives requests. These frequent health checks were chosen in order to achieve a more agile horizontal scaling since newly added vm-instances are being used more promptly, and failing vm-instances are identified more quickly.

B. *Experiment Setup and Scenarios*

The process model is a linear sequence of three identical automated tasks. This ensures resource requirement predictability. Each task calls a Web service calculating factorials to mimic a CPU-intensive load profile. A process instance is created through a BPMS interface from an initial input *fact_max*. This single input value may be regarded as the indicator for computational complexity of the specific process instance as it is related to the amount of factorials to be calculated by each service.

The vm-instances implement the factorial service and are located behind the load balancers. The vm-instances are derived from a single, publicly available vm-image[6] and are deployed on an AWS EC2 *small* instance virtual machine. The services provide a SOAP interface to receive service requests. When invoked, the services calculate the factorial of every positive integer smaller than *fact_max*. On completion, the Web services return the computing time in milliseconds, which can be taken as KPI by the BPM-aware auto scaler.

Two different workload scenarios were conducted: *ramp-up* and *scale-cycle*.

- In the *ramp-up* scenario, every 10 seconds a new process instance is started. Every 60 seconds the process instance input value is increased by 10,000 starting at 40,000. In total 54 instances are started over a time of 9 minutes.
- The *scale-cycle* workload scenario starts with the above ramp-up scenario. The ramp-up is followed by a plateau of constant high complexity input for three minutes. Finally, the complexity input decreases symmetrically. Overall, 114 process instances are triggered over a time of 19 minutes, with input values ranging from 40,000 to 120,000.

The ramp-up workload scenario is considered as an own scenario independently of the scale-cycle scenario. We identified that particularly in the process of scaling-out significant improvements can be achieved [21]. The workload scenario scale-cycle has a structure similar to a wave. It is intended to represent an exemplary basic element of volatile demand. Scaling-in does not yield the same improvement potential over the existing state-of-the-art as an individual test scenario [21] and has thus not been experimented with as a stand-alone scenario in this work.

The initiation of a business process originates from an internal or external client. These clients are simulated by a workload generator which can act as multiple concurrent clients requesting process instances. We used Apache JMeter[7] as a workload generator. It allows creating repeatable test plans

[2] http://www.intalio.com/products/bpms/
[3] http://ode.apache.org/
[4] http://www.eclipse.org/
[5] http://aws.amazon.com/elasticloadbalancing/

[6] BPMScalr Multi Factorial Benchmark WS (timeout-2): ami-8839a0e1
[7] http://jmeter.apache.org/

with a predefined time schedule. Technically, an individual process client is represented by a thread which sends a predefined SOAP request over HTTP to the Web service interface of the BPMS. The SOAP request contains the input value for a desired process instance, triggers its initialization and then waits for the BPMS to return the process instance's output.

The BPMS aims to meet a quality of service (QoS) constraint requiring process instances to complete within 500 seconds.

C. Resource Controller Prototype

The resource controller is implemented in Java. It monitors the process execution for one minute, computes a scaling decision and waits for the scaling execution to complete. The resource controller does not wait for confirmation of the scaling execution for longer than two minutes. Note that scaling will be conducted even if it takes longer than two minutes. Since the process tasks are identical, scaling decisions are executed for all scaling groups in accord. These timeframes have been chosen deliberately as a starting point for analyses.

The fuzzy controller is configured with a set of rules to guide the scaling decisions, so that the above QoS constraint can be satisfied. Due to the fuzzy matching procedure, incompleteness of the rules with respect to the input space is not necessarily a problem: since a fuzzy match between the rules is computed, there is no need to cover the whole parameter space. Six such rules are shown in Table II. Note that the 4 columns from left form the premise, and the right column is the conclusion (see also Section IV.A). The values in the cells are absolute (from "XS", extra small, to "L", large) or relative (from "--", strongly negative, to "++", strongly positive). For the actual controller, those values are translated into exact values for the respective metrics.

TABLE II. FUZZY SCALING RULE EXAMPLES

Change in backlog	Average process instance complexity	Average turnaround time	Active machines	Conclusion: scaling decision
+	M	M	1	+
++	M	S	1	+
++	L	S	2	+
++	L	L	1	++
--	XS		2	-
--	XS		3	--

The output of the resource controller is a fractional number and lies in the interval [-4; 4]. This number is rounded to an integer value *X*, which is the scaling decision: a negative value represents scaling-in *X* vm-instances, a positive number represents scaling-out by *X* vm-instances. In this scenario, we have limited the amount of vm-instances to be added or removed from a load balancer to a total of four, with a minimum of one vm-instance. The conclusion can be seen as a confidence-based "vote" of all rules: if there is some confidence that the rule applies, then its conclusion is part of the overall decision – basically weighed with the confidence. Furthermore, the set of active rules can be extended at any time.

Since the process tasks are identical, the number of in-service vm-instances is queried from the first load balancer. All other resource controller inputs are queried from the BPMS: change in backlog, complexity parameter, and average turnaround time.

The execution of the scaling decisions is tailored to the use with AWS EC2. A running vm-instance is charged by the commenced hour counting from its start. Hence, we de-register vm-instances from the load balancer to be possibly re-added before removing them at the end of an hour – as discussed in the previous Section. A cleanup procedure terminates a de-registered vm-instance before the vm-instance commences a new computing hour.

D. Comparison Baseline: AWS Auto Scaling

AWS provides a standard auto scaling service[8]. In this paper it is used to provide a reference to the results of the BPM-aware resource controller. AWS's auto scaling uses scaling rules based on vm-instance monitoring. It does not obtain any data from the BPMS. AWS auto scaling was configured to manage the pool of vm-instances for each load balancer individually. Yet, the scaling rules are identical for each load balancer. One rule for scaling-out and one for scaling-in were defined. These rules are thresholds based on the average CPU load among the instances in an ASG. Auto scaling was configured to a threshold of 40 % average CPU load of an ASG for scaling-in and threshold of 80 % for scaling-out. If the average CPU usage exceeds the upper threshold for one minute, the associated ASG is increased by one vm-instance. If the average CPU load drops below the lower threshold, the associated ASG is reduced by one vm-instance. After a scaling activity is performed, scaling is paused for one minute for the system to adapt to the changed resource situation. Scaling activities are only conducted if they do not violate a global minimum and maximum size restriction. The minimum of instances in an ASG was set to one and the maximum to four vm-instances.

E. Experiment Results

The results for the ramp-up scenario show an improvement of process execution and resource allocation regarding the observed KPIs. The incline in turnaround time of the fuzzy controlled scaling is considerably slower in the beginning of the scenario, between minute 1 and 4. Overall, the average turnaround time of the fuzzy controlled scaling is lower (174.5s) than of the auto scaling (244.75s). Although the fuzzy resource controller violates the defined QoS constraint of 500s in 5 instances, 90.1% are within the QoS constraint. AWS has violated the QoS in 8 out of 54 instances, i.e., in 14.8% of the cases.

The BPM-aware resource controller keeps a smaller backlog throughout the ramp-up workload scenario and completes earlier. In view of the resource usage, our resource controller scales later than the auto scaling. Also, it uses a maximum of eight vm-instances which is one more vm-instance than AWS's auto scaling.

Figure 3 (a) shows the turnaround time with AWS auto scaling in the scale-cycle scenario. Purple sequences are failing

[8] http://aws.amazon.com/autoscaling/

instances. AWS auto scaling is not able to scale appropriately as to complete all requested process instances. Even with lowered thresholds of 20 % and 60 % less than half of the 114 started process instances complete – to be precise, 57 % fail due to insufficient resources. This can either be due to time-outs because of unavailable vm-instances or because of pre-maturely terminated vm-instances. In contrast, the results of the BPM-aware resource controller are considerably better as Figure 3 (b) shows. All process instances complete within the QoS constraint.

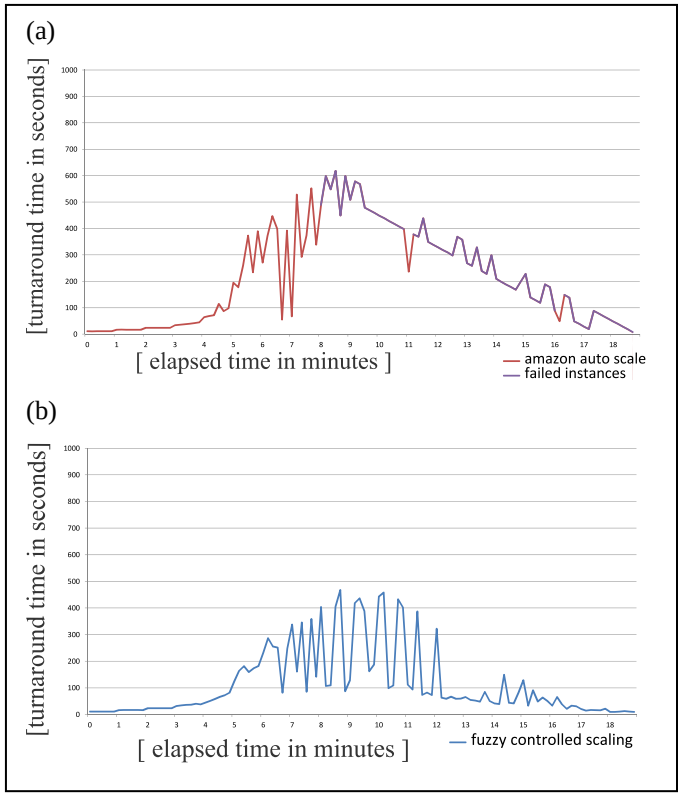

Fig. 3. Process Turnaround Time Scenario Scale-cycle.

This also entails that the size of the backlog is considerably different. Cf. Figure 4 for an overview.

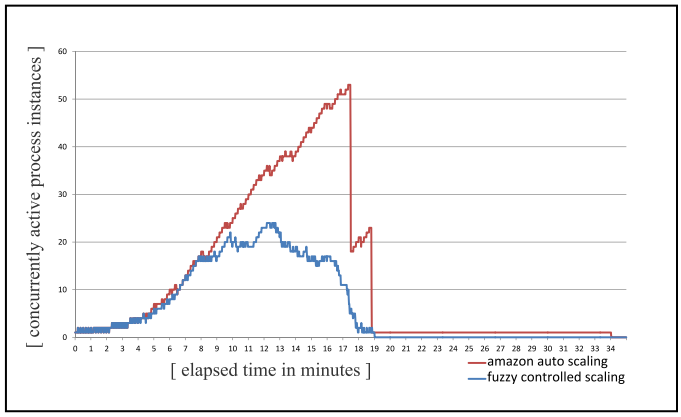

Fig. 4. Process Backlog Size per Minute Scenario Scale-cycle.

In the auto scaling setup there is no significant decrease in the backlog – except for failing instances – while the BPM-aware resource controller is able to keep the backlog to 25 process instances which still allowed their timely execution.

The BPM-aware resource controller used a total of 12 vm-instances which it allocated fairly early in minute 12. The AWS auto scaler used a maximum of 10 instances. After the termination of the first batch of failed process instances, only a maximum of 6 vm-instances were allocated to process the remaining service requests. Cf. Figure 5 for an overview.

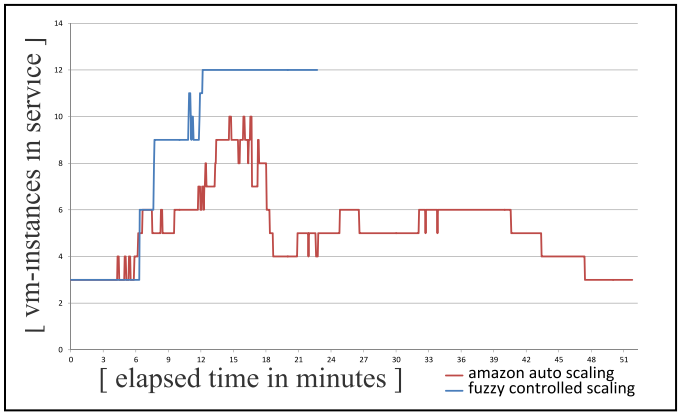

Fig. 5. Comparison of the Resource Usage Scenario Scaling-cycle.

F. Limitations

The limitations of the research conducted to this point concern the process model, the test bed, and the BPM-aware resource controller, as follows.

We believe that restricting the process to a linear control flow is reasonable assumption given the real life examples for automated processes. Parallel processing of tasks entails similar resource requirements. In future work we plan to consider more complex workflow structures in process models.

In this work, we assumed that dependencies in resource demand usually exist between process tasks, since the subject matter of the process is constant throughout all computed tasks. In order to assure predictability of the resource demand of each process task, all computed tasks were identical.

The chosen computational task of calculating factorials reduces possible side effects by reducing shared resources to a minimum. Yet, except for the fact that all tasks receive the same input value, in the given business process the individual process tasks are basically independent of one another except for being executed in succession. Using a process where process tasks build upon the result of the previous task would incorporate a more realistic data handling within the process. Nevertheless, one could determine relations between the processing requirements of the tasks which can be used to derive the estimate resource requirement of subsequent tasks. Another limitation of computing factorials is that the resulting load concerns primarily the CPU – other tasks may have other load profiles.

Assuming a constant rate of requests is a limitation intended to reduce the possible influential factors of a timely process execution. Furthermore, it allows observing the state of the BPMS: when process instance requests are sent periodically, yet process instances are created irregularly, this

provides an indicator as to the performance of the BPMS, where scheduling and logging may influence the performance. Also the capacity of the network is a factor which may limit the amount of requests at one time.

The main limitation of the test bed setup is its limited comparability due to the unpredictable behavior of the computing resources. It has been observed that numerous sources for performance variability exist when using AWS [22]. At times, launching vm-instances failed due to capacity constraints in certain regions, or due to a partial loss of the load balancer configuration. Yet, it is believed that a lack of predictability is to be expected in this domain: uncertainty is the norm, not the exception, in cloud computing. Adding to this observation is the fact that a lot of parameters concerning the provisioning of computing resources in a cloud environment cannot be influenced by the cloud service client. This includes the placement of running virtual machines within a datacenter as well as the balancing strategy employed in the load balancer. When comparing the individual experiments special attention is required since unobservable yet influential parameters may vary. It is nearly impossible to provide identical conditions for experiments in such a distributed environment.

The implemented prototype's input from the BPMS is limited to global process information. This design is sufficient given that the individual process tasks are identical and that the underlying resources scale in accord. If one of these two assumptions does not hold, bottlenecks within the process may occur. This would complicate the task of assuring a timely execution of the process since it would be necessary to scale the resources for each task individually. Such a controller would require additional inputs to capture the progress of instances along the process tasks. Additionally, it would require an output for each task's scaling decision, clearly increasing the controller's complexity. An alternative is to abandon the design of a central BPM-aware resource controller and use a distributed approach. This implies moving from a global to a local controller associated to each process task.

Also, we only considered horizontal scaling and did not research the impact a BPM-aware resource controller could have with vertically scaling up or down. The experiments were performed on Amazon EC2 small instances. We did not yet evaluate the impact on other virtual machine sizes.

VI. Discussion

Over thirty evaluated test runs have proven that the reliability and predictability of the AWS auto scaler can be further improved. Repeatedly the settings of the load balancers were partly lost, resulting in ineffective scaling decisions. The Web console provided by AWS for managing the usage of their Web services often presented outdated information. When using the API, the lag of the Web console became apparent. This gives reason that the overall scenario, which includes distributed and cloud-deployed sub systems, has unique properties. These have to be accounted for when transferring techniques from the field of control theory. In contrast to classic control applications, the behavior of the actuator in the considered scenario cannot be precisely foreseen. The timely execution of a scaling decision varies as discussed and may even fail completely.

We also experienced high volatility in the performance of cloud resources. The resources used in the experiments where hosted in eastern U.S. As most of the experiments were conducted during European morning or daytime, this was not an issue. When the experiments were repeated later in the day the added load during US daytime was noticeable.

A practical obstacle in the progress of experiments was the problem of connection timeouts. The AWS load balancers are configured for a maximal connection duration of sixty seconds. This timeout can be raised to 17 minutes only through the assistance by AWS's customer service. All other timeouts, like in the BPMS or the Web services, needed to be adapted as well – but can be easily set by the user.

The reasons behind the high turnaround times and failures of process instances are twofold: First, the AWS auto scaler reduces the ASG by terminating vm-instances. This means that instances are shut down while potentially still working on requests from the BPMS. AWS auto scaling does not consider exploiting the full computation hour which is paid in advance. Second, the scaling decision is based on local information only. Every load balancer's ASG is monitored individually. As the load on the first load balancer rises, it scales out. Yet, following requests gradually increase in complexity and need longer to compute. Hence, process instances are being jammed up at the previous task. When these process instances are passed on to the next auto scaler it is unable to cope with the stress the requests put on the available vm-instances.

VII. Conclusion

In this paper, we propose a BPM-aware resource controller for IaaS, based on fuzzy logic. It is intended to form a bridging component between cloud-based service resources and a BPMS. The resource controller is intended to maintain a quality of service time constraint defined for a process by dynamically provisioning cloud resources adequate to the state of process execution. The controller is BPM-aware in the sense that it derives its horizontal scaling decisions from monitoring the execution of process instances. We explained the design of the BPM-aware resource controller, and realized it in a prototypical implementation. The intended functionality of the prototype is then evaluated in a laboratory experiment using a test bed and two different workload scenarios. A standard commercially available auto scaler provides the performance benchmark for the evaluating of improvements over the state-of-the-art.

The tests clearly indicate the potential of a BPM-aware resource controller over traditional threshold based auto scalers which merely base their decisions on simple local measures such as CPU load and do not consider the workload of preceding tasks which can accumulate too large a number of work list items to be handled without violations by a subsequent ASG.

The evaluation also indicates that the controller prototype could be improved in regard to scaling-in more rapidly. This is expected to be solved with additional, appropriate rules. Furthermore, at this point no corrective measures are incorporated in the prototype's design for the case that an ASG does not scale in accord. While we have set up the scenario for the BPMS to execute processes with a certain maximum turnaround time, we have not yet implemented control

mechanisms to *not* scale if the turnaround time can still be met with the old configuration despite performance degradation.

To further improve the understanding and practicality of processes based on cloud-deployed services some enhancements of the specific setup and scenario appear promising. Considering the given work and its result, extended and realistic workload scenarios should be considered. Using historic traces as presented in [23] provides an approach. Thus, using a custom load balancer with process aware request distribution would provide mechanisms to control the timely execution of process instances considerably more precisely.

In regard to commercial applications, the resource controller could be enhanced by further incorporating cost. It could not only decide when to call in new vm-instances but also determine the best offer to use. This includes different suppliers or spot instances from different regions. Another option is to pursue conceptual variations. To increase the universality of the BPM-aware resource controller, we plan to investigate a variable control model using machine learning as in [24].

ACKNOWLEDGMENTS

This work is supported by the German Academic Exchange Service (DAAD) under the promotional reference “54392100”, by the Group of Eight (Go8) Australia-Germany Joint Research Cooperation Scheme, and by the German Federal Ministry of Education and Research (BMBF) under grant “01IS12031”. 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 M. Indulska, "Modeling Languages for Business Processes and Business Rules: A Representational Analysis", Information Systems, 35 (4), 2010, pp. 379-390.

[2] C. Janiesch, M. Matzner, and O. Müller, "A Blueprint for Event-driven Business Activity Management", In 9th International Conference on Business Process Management (BPM). Lecture Notes in Computer Science vol. 6896, Clermont-Ferrand, 2011, pp. 17-28.

[3] 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.

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

[5] J. Hellerstein, S. Singhal, and Q. Wang, "Research Challenges in Control Engineering of Computing Systems", IEEE Transactions on Network and Service Management, 6 (4), 2009, pp. 2006-2011.

[6] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia Arellano, "Incorporating Cost of Control into the Design of a Load Balancing Controller", In 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Toronto, 2004, pp. 376-385.

[7] T. Abdelzaher, L. Ying, Z. Ronghua, and D. Henriksson, "Practical Application of Control Theory to Web Services", In 2004 American Control Conference (ACC), vol. 3, Boston, MA, 2004, pp. 1992-1997.

[8] T. F. Abdelzaher, J. A. Stankovic, L. Chenyang, Z. Ronghua, and L. Ying, "Feedback Performance Control in Software Services", IEEE Control Systems, 23 (3), 2003, pp. 74-90.

[9] Y. Jiang, D. Meng, J. Zhan, and D. Liu, "Adaptive Mechanisms for Managing the High Performance Web-based Applications", In 8trh International Conference on High-Performance Computing in Asia-Pacific Region, Beijing, 2005, pp. 397.

[10] L. A. Zadeh, "Fuzzy Sets", Information and Control, 8 (3), 1965, pp. 338-353.

[11] R. Kruse, J. Gebhardt, and F. Klawonn, Foundations of Fuzzy Systems, Chichester: Wiley & Sons, 1994.

[12] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong, "Principles of Elastic Processes", IEEE Internet Computing, 15 (5), 2011, pp. 66-71.

[13] 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, Shanghai, 2012, pp. 179-190.

[14] 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.

[15] Y. Diao, J. L. Hellerstein, and S. Parekh, "Using Fuzzy Control to Maximize Profits in Service Level Management", IBM Systems Journal, 41 (3), 2002, pp. 403-420.

[16] A. Lenk, M. Menzel, J. Lipsky, and S. Tai, "What Are You Paying For? Performance Benchmarking for Infrastructure-as-a-Service Offerings", In 4th IEEE International Conference on Cloud Computing (CLOUD), Washington, D.C., 2011, pp. 484-491.

[17] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance", Proceedings of the VLDB Endowment, 3 (1-2), 2010, pp. 460-471.

[18] E. H. Mamdani, "Application of Fuzzy Algorithms for the Control of a Simple Dynamic Plant", Proceedings of the Institution of Electrical Engineers, 121 (12), 1974, pp. 1585-1588.

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

[20] Amazon Web Services, Inc., Amazon Elastic Compute Cloud (Amazon EC2), 2013, http://aws.amazon.com/ec2/.

[21] C. Janiesch, I. Weber, M. Menzel, and J. Kuhlenkamp, "Optimizing the Performance of Automated Business Processes Executed on Virtualized Infrastructure", In 45th Hawai'i International Conference on System Sciences (HICSS), Waikoloa, HI, 2014, pp. 3818-3826.

[22] 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.

[23] X. Jing, Z. Ming, J. Fortes, R. Carpenter, and M. Yousif, "On the Use of Fuzzy Modeling in Virtualized Data Center Management ", In 4th IEEE International Conference on Autonomic Computing (ICAC), Jacksonville, FL, 2007, pp. 25-35.

[24] H. Li and S. Venugopal, "Using Reinforcement Learning for Controlling an Elastic Web Application Hosting Platform", In 8th ACM International Conference on Autonomic Computing (ICAC), Karlsruhe, 2001, pp. 205-208.

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

[26] 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.