Self-Adaptive Resource Allocation for Elastic Process Execution

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

Srikumar Venugopal
The University of New South Wales, Australia
Email: srikumarv@cse.unsw.edu.au

***Abstract*—Especially in large companies, business process landscapes may be made up from thousands of different process definitions and instances. As a result, a Business Process Management System (BPMS) needs to be able to handle the concurrent execution of a very large number of workflow steps. Many of these workflow steps may be resource-intensive, leading to ever-changing requirements regarding the needed computing resources to execute them.**

Using Cloud technologies, it is possible to allocate workflow steps to resources obtained on demand from Cloud platform providers. However, current BPMS do not feature the means to make use of Cloud resources in order to execute workflows. This work presents an approach to automatically lease and release Cloud resources for workflow executions based on knowledge about the current and future process landscape. This approach to self-adaptive resource allocation for elastic process execution is implemented as part of ViePEP, a research BPMS able to handle workflow executions in the Cloud.

I. Introduction

Today, Business Process Management is a matter of great importance in many industries. Business processes involve both software services and human services, which are composed in order to deliver a specific business logic [1]. The part of a business process which can be executed using machine-based computation, is also known as a workflow [2]. The automatic processing of such workflows is a prominent field of research and has resulted in various concepts, methodologies and frameworks from the field of computer science [3]. In recent years, a number of approaches have been proposed which apply Web service technologies and especially service composition in order to model and execute workflows [4], [5].

Execution environments for business processes are commonly known as Process-Aware Information Systems (PAIS), i.e., systems “that manage(s) and execute(s) operational processes involving people, applications, and/or information sources on the basis of process models” [6]. BPMS are central components in a PAIS and may have to be able to handle potentially thousands of process definitions, which are themselves composed from a number of different process steps. There are processes which are instantiated on fixed intervals, while other processes are instantiated rather ad hoc. Such a process landscape may therefore be highly dynamic, leading to ever-changing demands regarding the computing resources needed in order to automatically carry out workflows. Nowadays, resource-intensive tasks are not only part of Scientific Workflows (SWFs), e.g., [7], [8], but occur in all industries which rely on the processing of large amounts of data in short time, e.g., the energy industry [9].

Obviously, the permanent provision of fixed IT capacities that could handle peak loads in such resource-intensive process landscapes would generate very high costs and at the same time lead to overprovisioning, as resources will not be fully utilized most of the time. In contrast, if the provided fixed IT capacities are not sufficient (underprovisioning), this will inevitable lead to the situation that particular workflows cannot be executed at all or provide a poor Quality of Service (QoS) [10]. In order to avoid over- and underprovisioning, a BPMS should be able to lease and release computing resources during runtime, i.e., while processes are invoked and executed. A technology is needed which is able to react to the dynamic changes of needed computing resources while still ensuring the needed QoS levels. In recent years, both the software industry and the research community have put a remarkable focus on the advancement of Cloud technologies, which are capable to achieve exactly this [11]. However, surprisingly little effort has been put into the exploration of Cloud technologies in BPMS and especially as a technology to execute workflows.

As stated in the seminal definition of the National Institute of Standards and Technology (NIST) [12], “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”. In order to allow a BPMS to lease and release computing resources based on dynamic requirements, especially the essential Cloud characteristics to deliver *on-demand self-service*, i.e., that Cloud-based resources are provided on-demand by request, *rapid elasticity*, i.e., that the underlying Cloud infrastructure allows dynamically scaling up and scaling down by adding or removing computing resources, and *measured service*, i.e., that the consumption of resources is measured and serves as a foundation for elastic pricing and billing models [12], are of great interest.

In our former work [13], we have presented a first version of the *Vienna Platform for Elastic Processes* (ViePEP), which is a BPMS able to control a process landscape in a reactive way, schedule the individual workflow steps, lease and release Cloud resources when necessary, and execute services implementing these workflow steps in the Cloud.

978-0-7695-5028-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CLOUD.2013.126

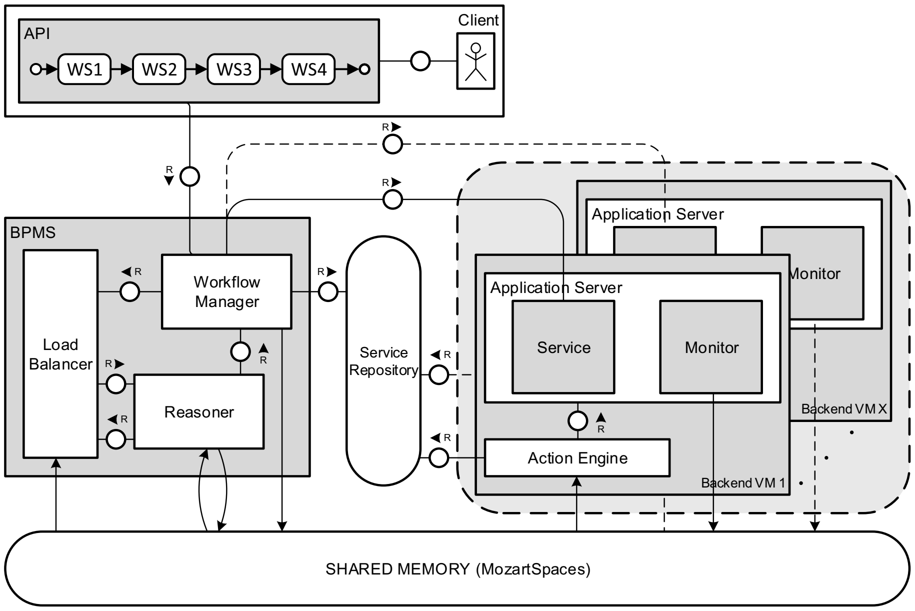


Fig. 1: ViePEP Architecture

ViePEP makes use of Cloud-based resources on the Platform as a service (PaaS) level [12] and is therefore able to deploy software services representing single workflow steps on Virtual Machines (VMs). By integrating ViePEP into a PAIS, it is possible to decrease the risk of over- and underprovisioning while guaranteeing the needed QoS levels of workflows.

In this paper, we extend ViePEP by the means to predict future resource demands based on knowledge about the current and future process landscape, thus extending ViePEP into an self-adaptive business process and Cloud resources management system. The main research challenges we focus on concern the reasoning about the current and future need of computing resources including the lease and release of such resources from a Cloud provider. In addition to that, we focus on the scheduling of workflow steps (services) as well as the actual execution of business processes in the Cloud including the automatic deployment and undeployment of services.

The remainder of this paper is organized as follows: In Section II, we introduce ViePEP as foundation for the work conducted within this paper. Afterwards, we will define the needed functionalities and according approaches to realize them. In Section IV, we present the prediction and reasoning mechanisms necessary to achieve self-adaptive Cloud resource allocation for elastic process execution. We evaluate the prediction and reasoning mechanisms through testbed experiments (Sections V and VI). In Section VII, we comment on the related work. Finally, Section VIII concludes this paper and presents our future work.

II. VIEPEP – THE VIENNA PLATFORM FOR ELASTIC PROCESSES

The execution of workflows composed from single services in the Cloud needs mechanisms and architectural components that are beyond the capabilities of current PAIS and BPMS. Thus, we have designed and implemented ViePEP, a platform that is able to:

- Provide an interface to the Cloud, allowing to lease and release Cloud resources on demand
- Deploy workflow steps by instantiating services on VMs and invoking these service instances in workflows
- Dynamically arrange incoming requests for single workflow steps based on their QoS requirements, i.e., timeliness in terms of maximum execution time or a defined finish time
- Monitor the VMs in terms of resource utilization and monitor the QoS of individual service invocations

As depicted in Figure 1, ViePEP has five top level entities:

The *Client* models service-based workflows and can optionally define Service Level Agreements (SLAs) for the overall workflow and each single service. A Client may request many workflows consecutively or even at the same time. ViePEP is able to serve several clients in parallel.

The *BPMS VM* offers the central functionalities of ViePEP in terms of a BMPS and Cloud control solution. It runs in an own VM. As the BPMS VM is responsible to plan and control the future executions of workflows, it comprises a *Workflow Manager*, a *Load Balancer*, and the *Reasoner*, which capsules the essential workflow scheduling and resource allocation logic. Within the BPMS, the Workflow Manager is the subcomponent responsible for the actual service scheduling and invocation. It accepts Client-issued workflow requests and maintains a queue of workflow steps which is ordered by their individual priorities, which are sorted by the deadline for a workflow execution. The Workflow Manager also measures the execution time for services, which is a prerequisite to detect deviations from the expected QoS behavior of single services and starts corresponding countermeasures, if necessary.

As the Reasoner is providing the actual scheduling and allocation logic, we will discuss its capabilities in more detail in Section IV.

A *Backend VM* hosts a particular service. In a typical ViePEP-based process landscape, many Backend VMs exist at the same time and have to be managed. Together with the BPMS, the Backend VM is the central entity in ViePEP. It hosts an application server on which the single services are deployed. For monitoring the services regarding their response time and the VMs' CPU and RAM load, a *Monitoring* component is deployed. The *Action Engine* performs actions such as deploying a service, undeploying a service, or shutting down the VM. This Action Engine is able to *start* a new Backend VM and instantiate a service on it, *terminate* a Backend VM, or *duplicate* a Backend VM, if correspondingly instructed by the ViePEP BPMS VM. Furthermore, it is possible to *move* a running service to another Backend VM, or *exchange* the service hosted by a Backend VM by another service.

The *Shared Memory* and *Service Repository* are helper components. The latter hosts the service descriptions as well as their implementations as portable Web application ARchive (WAR) files. This repository allows to search for services and deploy them on a ViePEP Backend VM. The Shared Memory is used to provide data sharing between the BPMS VM and the different Backend VMs. We chose MozartSpaces [14] for this, as it allows to easily deploy and access a peer-to-peer-based, distributed shared memory. We use this shared memory as a database where the monitoring data is stored. Further, we make use of the notification feature of MozartSpaces for sending action commands from the BPMS VM to the single Backend VMs.

ViePEP is implemented in Java and provides an interface to OpenStack-based Clouds. However, by changing the corresponding interfaces, it is possible to deploy ViePEP with arbitrary Clouds as long as they provide a corresponding API to carry out the necessary actions. For further details about ViePEP, we refer to [13].

III. Self-Adaptive Resource Allocation for Elastic Process Execution

As has been described, ViePEP is a BPMS fully equipped to control a process landscape in terms of scheduling workflows and single workflow steps and lease, allocate, and release Cloud resources. However, ViePEP is currently only able to act in a reactive way, i.e., depending on the next workflow step in the workflows queue (cp. Section II), the system deploys this step as a software service on Cloud resources and executes the service. Apart from sorting the workflow steps to be executed based on their individual QoS requirements and most importantly the timeliness, ViePEP does not provide any optimization of the workflow (step) scheduling.

While ViePEP therefore ensures that all workflow steps (and therefore complete workflows) are carried out in time by leasing and allocating Cloud resources, this is done in a rather ad hoc manner. As an alternative, ViePEP could also make use of knowledge about future workflow steps in order to find a more optimal service scheduling and resource allocation and therefore provide a more cost-efficient approach to elastic process execution. For this, ViePEP should make use of information available from the future steps of a process definition, i.e., the information which services need to be invoked in the (near) future and what are their individual QoS requirements. This way, ViePEP would provide self-adaptive resource allocation for elastic process execution.

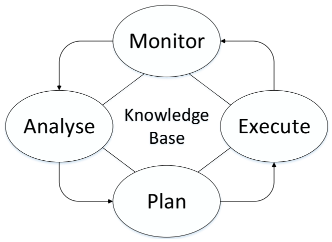


Fig. 2: MAPE-K Cycle (adapted from [15])

Self-adaptation (also sometimes known as self-management) is a common concept from the field of Autonomic Computing and includes *self-healing*, i.e., the ability of a system to detect and recover from potential problems and continue to function smoothly, *self-configuration*, i.e., the ability of a system to configure and reconfigure itself under varying and unpredictable conditions, and *self-optimization*, the ability to detect suboptimal behavior and optimize itself to improve its execution [15]. So far, ViePEP provides self-healing and self-configuration as has been shown in the last section. The focus of this paper is on self-optimization.

In order to motivate the functionalities and components needed to provide self-optimization of a Cloud-based process landscape, we make use of the well-known MAPE-K cycle shown in Figure 2. As the scenario at hand is highly dynamic due to permanently arriving workflow requests and changing Cloud resource utilization, a continuous alignment to the new system status is necessary. Using the MAPE-K cycle, the system is continuously monitored and optimized based on knowledge about the current system status. In the following, we will briefly discuss the four phases of this cycle and how this influences the self-management capabilities of ViePEP:

- **Monitor:** In order to adapt a system, it is first of all necessary to monitor the system status. In the scenario at hand, this includes the monitoring of the status of single VMs in terms of CPU and RAM utilization, and the non-functional behavior of services in terms of response time and availability. The according monitoring functionalities are already part of ViePEP as presented in Section II.
- **Analyze:** To achieve elastic process execution, it is necessary to analyze the monitored data and reason on the general knowledge about the system. In short, this analysis is done in order to find out if there is currently under- and overprovisioning regarding the Cloud resources (VMs) and to detect SLA violations in order to

carry out according countermeasures (e.g., provide further resources or reinvoke a service).

- **Plan:** While the analysis of the monitored data and further knowledge about the system aims at the current system status, the planning also takes into account the future resource needs derived from the knowledge about future workflow steps in the queue and their SLAs. For this, ViePEP needs to carry out reasoning and generate a scheduling and resource allocation plan.
- **Execute:** As soon as the plan is set up, each service is carried out according to this plan.
- **Knowledge Base:** While not really a part of the cycle, the Knowledge Base stores information about the system configuration. In our case, this is the knowledge about which service instances are running on which VMs, how many VMs are currently part of the system, and of course the knowledge from the workflow and services queue.

As it can be seen, ViePEP needs to provide further analysis and planning capabilities in order to self-optimize the process landscape and the involved Cloud resources. In the next section, we will present how this is achieved.

IV. Reasoner

While the Workflow Manager controls the invocation of the single workflows, the Reasoner aims to optimize the complete system landscape in terms of workflow scheduling and Cloud resource leasing and allocation. It is responsible to find a scheduling plan for the workflows and the included workflow steps under the given SLAs and cost, resource, and quality constraints [16]. This plan is then forwarded to the Workflow Manager and executed by it.

In any case, the Reasoner needs to take into account the information about the currently running workflows as well the requested workflows which have not yet been started. For this, the QoS constraints (maximum execution time or a defined finish time) are taken into account. Further, the Reasoner reacts to deviations from the expected non-functional service behavior in order to find an appropriate countermeasure that can then be carried out by the Workflow Manager. The Reasoner reads the information about the currently VM resources (CPU and RAM usage) from the MozartSpaces and interacts with the Load Balancer in order to decide whether a particular Backend VM is sufficient to carry out a service invocation, or if another Backend VM hosting that service needs to be started, or if a VM can be terminated due to low load. The Reasoner produces the schedule plan following the following three steps:

Step 1: In the first step, the Reasoner creates a prediction model which is able to compute the need of resources for one single service invocation. Since a Backend VM may serve many service invocations at the same time, it is not possible to measure the utilization of one single service invocation in a direct way. Instead, it is necessary to derive the needed resources from the monitored data from the according Backend VMs.

The measured data can be represented as a vector m_t:

$$m_t = \{V, S_V, C_t, R_t, I_t\} \tag{1}$$

- V represents a Backend VM's ID, indicating at which machine the data was monitored
- S_V specifies the service running on V
- I_t defines the number of concurrent service invocations at time t
- C_t defines the CPU usage in percent at time t
- R_t defines the RAM utilization in percent at time t

In order to find the relation between the CPU load and a single invocations we make use of Ordinary Least Square (OLS) Linear Regression [17]. OLS is based on the notion that a dependent variable y – corresponding in our case to the CPU load – can be derived through the linear combination of a set of independent variables x_i. Using OLS, it is possible to find a linear combination of $x_2, ..., x_K$ and a constant representing a precise approximation for y:

$$\tilde{\beta}_1 + \tilde{\beta}_2 x_2 + ... + \tilde{\beta}_K x_K \tag{2}$$

$\tilde{\beta}_1, ..., \tilde{\beta}_K$ are the constants to be computed. If we index the observations by the variable i for $i = 1, ..., N$ we can express the difference between an observed value y_i and its linear approximation as

$$y_i - [\tilde{\beta}_1 + \tilde{\beta}_2 x_{i2} + ... + \tilde{\beta}_K x_{iK}] \tag{3}$$

In order to simplify this derivations we combine x-values for an individual i in a vector x_i including the constants and introduce the following short-hand notation:

$$x_i = (x_{i2}\ x_{i3}\ ...\ x_{iK}) \tag{4}$$

After collecting the $\tilde{\beta}$ coefficients in a new K-dimensional vector $\tilde{\beta} = (\tilde{\beta}_1,\ ...,\ \tilde{\beta}_K)'$ we can define:

$$y_i - [\tilde{\beta}_1 + \tilde{\beta}_2 x_{i2} + ... + \tilde{\beta}_K x_{iK}] = y_i - x_i'\tilde{\beta}. \tag{5}$$

It is desirable to choose $\tilde{\beta}_1, ..., \tilde{\beta}_K$ such that the differences are as small as possible. Applying OLS, this means to choose $\tilde{\beta}$ such that the *sum of square* is as small as possible. Therefore, we determine $\tilde{\beta}$ so that it minimizes the following function:

$$S(\tilde{\beta}) = \sum_{i=1}^{N} (y_i - x_i'\tilde{\beta})^2 \tag{6}$$

Taking the square ensures that positive and negative deviations do not cancel out each other when doing the accumulation. Using this equation, it is possible to apply a standard OLS estimator. After β has been computed, the predicted function using the linear regression is defined as:

$$y_i = [\tilde{\beta}_1 + \tilde{\beta}_2 x_{i2} + ... + \tilde{\beta}_K x_{iK}]. \tag{7}$$

where y_i is the CPU load when having $k - 1$ characteristics with $i...N$ observations.

Step 2: The result of Step 1 is a function we apply in order calculate the needed CPU resources based on the number of service invocations in the future. This number is inherent to the workflow queue and therefore provided by the Workflow Manager.

Finally, in **Step 3**, the result of Step 2 is compared to the current system state and thereupon, decisions about necessary

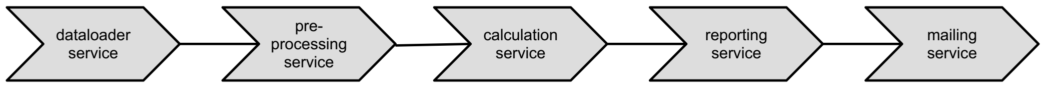


Fig. 3: Evaluation Workflow

changes to the workflow scheduling and actions regarding the Cloud resources are made. These actions include the above-mentioned options to start, terminate, or duplicate Backend VMs hosting particular service instances.

V. Evaluation

ViePEP has been prototypically implemented in Java 1.6 and has been tested in a Cloud running OpenStack[1]. The single services have been deployed using Apache Tomcat[2] as Application Server and have been monitored using PSI Probe[3]. The Reasoner makes use of Apache Commons Math[4] to solve the OLS Linear Regression problem.

In the following, we will present the performed evaluation including the evaluation scenario (Section V-A), the different workflow request arrival patterns we took into account (Section V-B), and the applied evaluation metrics (Section V-C). We will present and discuss the results in Section VI.

A. Evaluation Scenario

While ViePEP and the presented reasoning approach are applicable in arbitrary process landscapes and industries, we evaluate the Reasoner using a real-world data analysis process from the finance industry. Choosing this particular process does not restrict the portability of our approach to other domains. We apply a testbed-driven evaluation approach, i.e., real Cloud resources are deployed. For the single services, we simulate differing workloads regarding CPU and RAM utilization and service runtime as discussed below. However, once again, real services are deployed and invoked.

To simplify an interpretation of the chosen evaluation settings, we decided to make use of one single workflow which will be processed 20,000 times. This workflow consists out of five individual service steps as depicted in Figure 3. In order to execute the workflow, each single service has to be invoked in sequence. In the following paragraphs, we will briefly introduce the different services:

- Dataloader Service: The Dataloader Service simulates the loading of data from an arbitrary source. This could be a service which reads data from a database, from sensors or a service which communicates with other services. Since this service is not performing any calculations, this service does not need many resources.
- Data Pre-processing Service: After the data has been loaded, it needs to be pre-processed before it can be used by another service. This pre-processing is a bit more resource-intensive than loading the data. In order to simulate this load, every time this service is invoked, simple mathematical calculations are carried out.
- Calculation Service: After the data has been prepared, this service is called. It simulates the processing of the data. This service needs in contrast to the other services more computational resources, especially the CPU usage will be much higher than for the others. In order to simulate a high CPU usage, complex, resource-intensive mathematical calculations are carried out each time the service is invoked, i.e., by calculating the Fibonacci sequence for a certain period of time.
- Reporting Service: After the data has been processed, a report will be generated, e.g., in form of a PDF file. The needed CPU usage for generating such a report depends highly on the data being processed in the step before. In order to simulate this CPU load, some simple mathematical calculations as done in the Data Pre-Processing Service will be carried out again.
- Mailing Service: The last step in the workflow at hand is to deliver the created data analysis report. This can be achieved by sending of an email or pushing the created report onto a web server. However, this service is again not resource-intensive. Therefore, mathematical calculations similar to the ones in the Dataloader Service are carried out.

B. Arrival Patterns

We make use of three different workflow request arrival patterns in our evaluation:

- Constant Arrival Scenario: In this scenario, workflow requests arrive in a constant manner, i.e., the same amount of requests arrives at predefined intervals (here: every 10 seconds). The number of simultaneously executed workflows is set to 200. However, this value is not relevant to the result of our optimization which will be clarified in the discussion below. A graphical depiction of this arrival cure is depicted in Figure 4a.
- Linear Arrival Scenario: In this scenario, the workflows are executed in the manner of a linear rising function, i.e., $y = k * x + d$ where y is the amount of concurrently requested workflows and d is the start value. The number of concurrently requested workflows is increased by $k = 10$ every 2 seconds. The Linear arrival function is depicted in Figure 4b.
- Pyramid Arrival Scenario: In this scenario, workflow requests arrive in form of a pyramid-like function which can be seen in Figure 4c. In this scenario we start with a low number of simultaneously executed workflows, increase

[1]http://www.openstack.org
[2]http://tomcat.apache.org
[3]http://code.google.com/p/psi-probe
[4]http://commons.apache.org/math/userguide/stat.html

TABLE I: Evaluation Results

	Constant Arrival		Linear Arrival		Pyramid Arrival	
	Baseline	Reasoner	Baseline	Reasoner	Baseline	Reasoner
Number of Workflow Requests	20,000					
Interval between two Request Bursts (in Seconds)	10		2		40	
Number of Requests in one Burst	200		$y = 10 * x + 80$		Based on Random Number	
Duration in Minutes (Standard Deviation)	73.33 (σ=2.89)	76.67 (σ=2.89)	66.67 (σ=2.89)	60 (σ=2.89)	70 (σ=0)	65 (σ=0)
Highest Number of Concurrent VMs (Standard Deviation)	9.33 (σ=0.57)	9.33 (σ=0.57)	16 (σ=0)	16.33 (σ=0.57)	12 (σ=0.0)	12.67 (σ=0.57)
Costs in VM-Minutes (Standard Deviation)	535 (σ=17.32)	526.67 (σ=11.55)	875 (σ=70)	758.33 (σ=53.93)	660 (σ=13.23)	576.67 (σ=15.28)

it to a randomly chosen maximum (between 50 and 400) and decrease it again to 0. The number of concurrently requested workflows is increased every 40 seconds by 10 until the maximum has been reached. Afterwards, the number of concurrently requested workflows is decreased in the same manner. We repeat these steps until the workflow queue is empty. For the evaluation, the randomly chosen maximum is always the same in order to ease comparison different evaluation runs.

The arrival functions have been chosen in order to show the performance of the ViePEP Reasoner in scenarios where it is easy to forecast incoming requests due to homogenous patterns (constant and linear arrival) and in a scenario following a seemingly unpredictable arrival pattern (pyramid arrival).

C. Metrics

We execute each arrival pattern three times in order to produce reliable results. During execution, we measure the following quantitative metrics:

- Duration in Minutes: The first metric is the overall execution duration of all 20,000 workflows, i.e., the timespan from the arrival of the first workflow request to the moment when the last workflow execution is done.
- Highest Number of Concurrent VMs: The second criteria is the total amount of needed VMs including their lifetime. The result of this is a list of VMs and how long they have been running. We derive the highest number of concurrent VMs from this information.
- Costs: The resulting costs are calculated by using the following approach: Since we assume that each VM has the same features and costs the same, we can add up the overall runtime of the VMs. The costs for 20 VMs running for 5 minutes are the same as for 10 VMs running for 10 minutes. Following this approach, costs are represented by VM-Minutes. 1 VM-Minute is defined as 1 VM running for 1 minutes, therefore, e.g., the overall costs for 20 VMs running for 5 minutes is 100 VM-Minutes.

We compare the results if applying the reasoning as presented in Section IV to a baseline in which workflow steps are scheduled in an ad hoc manner. For this, the Reasoner is disabled. Instead, workflow steps are scheduled solely based on their execution deadline and executed following this plan. Cloud resources are leased and released whenever an upper/lower utilization threshold is exceeded. This baseline mirrors the approach in Cloud resource allocation approaches which do not take into account the process perspective (cf. Section VII).

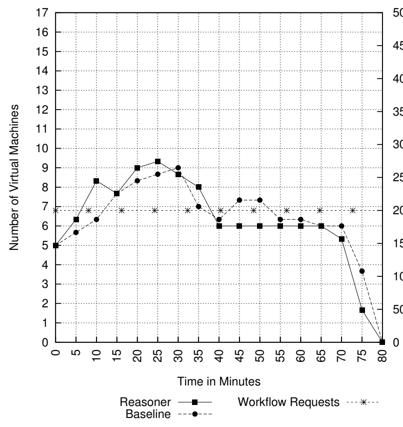
VI. Results

Table I and Figure 4 present our evaluation results in terms of the average numbers from the conducted evaluation runs. Table I presents the observed metrics as discussed in Section V-C for all three arrival patterns. For each arrival pattern, the numbers for evaluation runs are given for the baseline algorithm as well as with a deployed Reasoner. The table also states the standard deviation (σ) from the average values. In general, the observed standard deviation is low, and therefore indicates a low dispersion in the results of the different evaluation runs.

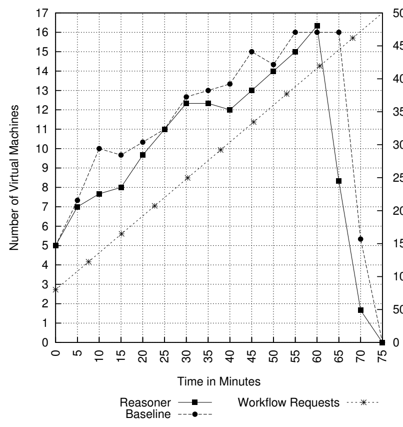
Figure 4 completes the presentation of the average evaluation results by depicting the arrival patterns (“Workflow Requests”) over time and the number of running VMs until all workflow requests have been served. Again, evaluation results with deployed Reasoner and applying only the baseline approach are shown.

The numbers from Table I indicate a substantial performance difference between the baseline and the Reasoner-based approach. Most importantly, the costs in terms of VM-Minutes are lower for all three applied arrival patterns. However, the degree of cost savings differs from pattern to pattern. In the constant arrival scenario, the difference is comparably low, showing that the ad hoc resource allocation approach of the baseline is already leading to good results and therefore, only minor improvements can be achieved. This interpretation is supported by the fact that the constant arrival pattern in general features the lowest costs.

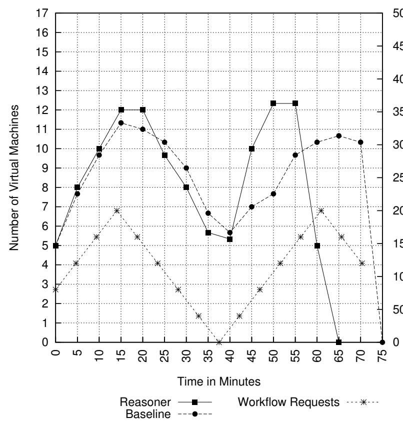
In contrast, the Reasoner-based approach leads to cost savings of 13.3% for the linear arrival scenario, while in the pyramid arrival scenario, cost savings amount to 12.63%. This can be traced back to the fact that the Reasoner helps to achieve a better utilization of VMs, thus preventing additional costs arising from overprovisioning of leased Cloud resources. Interestingly, both the baseline algorithm and the Reasoner lead to generally lower costs in the pyramid arrival scenario than those observed in the linear arrival scenario. This indicates that the costs do more depend on the actual number of concurrent workflow requests than the actual pattern itself. The higher the number of concurrent workflow requests, the



(a) Constant Arrival Results



(b) Linear Arrival Results



(c) Pyramid Arrival Results

Fig. 4: Evaluation Results

higher is the number of invoked VMs and therefore the costs. Of course, for a higher number of VMs, load balancing is generally more difficult to achieve and thus, there is a higher risk of under- and overprovisioning. This interpretation of the evaluation data is also supported by the numbers observed for the constant arrival scenario, which features both the lowest costs and number of concurrent number of VMs.

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

VII. Related Work

So far, surprisingly little effort has been put into the research in the field of elastic processes and elastic process execution [16]. Nevertheless, there is some related work from the fields of Grid computing and Cloud computing which should be regarded. Most importantly, scalability and cost-effective allocation of single tasks and services has been regarded by many researchers. Early research efforts often focus on minimizing the costs for Cloud consumers while taking into account the maximum allowed execution time and other QoS requirements, e.g., [18], [19]. Later research approaches also consider SLA enforcement, e.g., [20], [21], [22].

More recently, research efforts are focusing on the infrastructure perspective, i.e., the compliance of SLAs defined by the customer under the objective of maximizing the profit for the infrastructure provider or a broker [23] or a higher resource utilization [24], [25], [26], [27]. While most of these approaches apply threshold-based fixed rules to identify necessary actions (e.g., stop/start servers, move services), Li and Venugopal [26] make use of reinforcement learning to automatically scale an application up or down based on incoming workload.

In general, existing approaches for scaling and allocating Cloud resources in a cost-effective way are not taking the process perspective into account. Instead, allocations are mostly performed based on the current workload of currently invoked services. Information about potential future requests derived from the description of processes/workflows is not taken into account at all.

Even if not regarding elastic process execution, only few researchers make use of prediction mechanisms for the future resource utilization and take into account knowledge about the future state of a system and optimize resource allocation based on this. However, Lim et al. [28], [29] forecast resource utilization for Cloud-based data storage.

VIII. Conclusion and Future Work

Resource-intensive processes and their execution using workflow and service technologies play a more and more important role in many industries. The usage of Cloud resources to allow the execution of such processes in an elastic way seems to be an obvious choice, but so far, BPMS do lack the ability to lease and release Cloud resources and allocate them in order to execute workflows.

In this paper, our work on ViePEP has been substantially extended by proposing a prediction and reasoning algorithm for elastic process execution. The according Reasoner is able to schedule services and allocates Cloud resources under user-defined non-functional requirements in a cost-efficient way. As we have shown in the evaluation, compared to an ad hoc allocation of Cloud resources, this leads to cost reductions. In the evaluation, we use a limited set of services and only one workflow. Even though the evaluation is significant, we consider it as preliminary. A more detailed evaluation will be presented in our future work.

As discussed before, research on elastic process execution is just a the beginning and we believe that a lot of corresponding approaches will follow in the next years. In our future work, we are planning to enhance the complexity of elastic process execution by taking into account VMs offering different resources instead of the standardized VMs we are using at the moment. Furthermore, we are only allowing to scale in a horizontal way by adding more VMs to the overall system (*scaling out*). In the future, we will also allow *scaling up* of VMs, i.e., to add more resources to already leased VMs. We will continue our work on prediction and reasoning algorithms and take into account further approaches, e.g., from the field of linear programming or machine learning.

Acknowledgements

This work is partially supported by the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201) and by the Austrian Science Fund (FWF): P23313-N23. Part of the implementation has been done during Philipp's stay at the University of New South Wales, which was supported by a scholarship from Vienna University of Technology's International Office. We'd like to thank Han Li for his help with the implementation.

References

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

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

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

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

[5] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer, "Service-Oriented Computing Research Roadmap," in *Service Oriented Computing (SOC)*, ser. Dagstuhl Seminar Proceedings, no. 05462. Schloss Dagstuhl, Germany, 2006, pp. 38–45.

[6] M. Dumas, W. M. van der Aalst, and A. H. ter Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley & Sons, Hoboken, NJ, 2005.

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

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

[9] S. Rohjans, C. Dänekas, and M. Uslar, "Requirements for Smart Grid ICT Architectures," in *Third IEEE PES Innovative Smart Grid Technologies (ISGT) Europe Conference*. IEEE Computer Society, 2012.

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

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

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

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

[14] E. Kühn, R. Mordinyi, M. Lang, and A. Selimovic, "Towards Zero-Delay Recovery of Agents in Production Automation Systems," in *2009 IEEE/WIC/ACM Inter. Conference on Intelligent Agent Technology (IAT 2009)*. IEEE Computer Society, 2009, pp. 307–310.

[15] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

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

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

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

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

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

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

[22] H. Nguyen Van, F. Dang Tran, and J.-M. Menaud, "SLA-Aware Virtual Resource Management for Cloud Infrastructures," in *Ninth IEEE Inter. Conference on Computer and Information Technology (CIT '09)*. IEEE Computer Society, Washington, DC, USA, 2009, pp. 357–362.

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

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

[25] A. Kertesz, G. Kecskemeti, and I. Brandic, "An Interoperable and Self-adaptive Approach for SLA-based Service Virtualization in Heterogeneous Cloud Environments (forthcoming)," *Future Generation Computer Systems*, vol. NN, no. NN, pp. NN–NN, 2012.

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

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

[28] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated Control in Cloud Computing: Challenges and Opportunities," in *First Workshop on Automated Control for Datacenters and Clouds (ACDC09) in conjunction with the 6th Inter. Conference on Autonomic Computing and Communications (ICAC 2009)*. ACM, 2009, pp. 13–18.

[29] H. C. Lim, S. Babu, and J. S. Chase, "Automated Control for Elastic Storage," in *7th Inter. Conference on Autonomic Computing (ICAC 2010)*. ACM, 2010, pp. 1–10.