Introducing the Vienna Platform for Elastic Processes

Stefan Schulte[1], Philipp Hoenisch[1],
Srikumar Venugopal[2], and Schahram Dustdar[1]

[1] Distributed Systems Group, Vienna University of Technology, Austria
{s.schulte,dustdar}@infosys.tuwien.ac.at
[2] School of Computer Science and Engineering,
The University of New South Wales, Sydney, Australia

Abstract. Resource-intensive tasks are playing an increasing role in business processes. The emergence of Cloud computing has enabled the deployment of such tasks onto resources sourced on-demand from Cloud providers. This has enabled so-called elastic processes that are able to dynamically adjust their resource usage to meet varying workloads.

Traditional Business Process Management Systems (BPMSs) do not consider the needs of elastic processes such as monitoring facilities, tracking the current and future system landscape, reasoning about optimally utilizing resources given Quality of Service constraints, and executing necessary actions (e.g., start/stop servers, move services). This paper introduces ViePEP, a research BPMS capable of handling the aforementioned requirements of elastic processes.

1 Introduction

Business Process Management (BPM) is a multidisciplinary approach, covering organizational, management, and technical aspects, and "includes methods, techniques, and tools to support the design, enactment, management, and analysis of operational business processes" [1]. One particular subtopic of BPM is the automatic execution of modeled processes (process automation), which needs to be supported by concepts, methodologies and frameworks from the field of computer science [18]. The automated part of a business process is also known as a business workflow [17]. Very often, (Web) services are composed to create flexible, dynamic business workflows that may span organizations and computing platforms [20].

Currently, resource-intensive tasks not only are present within scientific workflows (SWFs), but are also becoming more prevalent in business processes. For example, compute and data-intensive analytical processes are found in the finance industry and in managing smart grids in the energy industry. In the latter, data from a very large number of smart grid sensors needs to be automatically gathered, processed, analyzed, and stored in order to offer customers consumption reports or even guarantee grid stability [21,22].

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 179–190, 2013.

Apart from the functional requirements of resource-intensive tasks, processes comprising them are subject to a number of non-functional requirements (Service Level Objectives – SLOs), especially with regard to the timeliness of the tasks – some of these processes need to be carried out in real-time, while others can be postponed but need to be executed within a particular deadline. As the amount of data, or the number of process instances that need to be concurrently handled, could vary to a very large extent, it is difficult to estimate the ever-changing resource demands of such processes.

Workflows could utilize the *resource elasticity* – to acquire and release resources as and when needed – to scale with shifting workloads [3]. Permanently provisioning IT capacities that are able to handle peak loads is not the best solution for this, as the capacities will not be utilized most of the time, leading to unnecessary high costs. With the advent of Cloud computing, organizations presently have a much more cost-efficient alternative that enables the use of computing resources in an on-demand, utility-like fashion [6]. While resource elasticity is a common way to describe the scalability of single applications as well as workflows, elasticity is not the only constraint that should be taken into account in the context of workflow scalability [8]. Notably, Quality of Service (QoS) in terms of criteria like response time does not necessarily reflect resource elasticity in a linear way, i.e., there may be no proportional relationship between involved resources and QoS. As a result, it is necessary to define *quality elasticity*, which describes the responsiveness of quality regarding changes in resource usage [8]. Last but not least, many Cloud providers make use of dynamic pricing models, which should also be taken into account if Cloud resources are used in order to realize scalable processes. These dynamic pricing models are reflected in *cost elasticity*, i.e., a resource provisioner's responsiveness to changes in costs [8].

To the best of our knowledge, so far, surprisingly little effort has been put into the investigation of methods, algorithms and tools to integrate automatic process execution and Cloud computing in order to realize so-called *elastic processes* under the above-mentioned elasticity constraints. In our experience, there is a lack of a Business Process Management System (BPMS) able to carry out many interdependent service-based workflows in parallel, estimate their current *and* future resource demand under user-specified constraints and preferences, and allocate Cloud resources dynamically to meet them. This needs analysing the process definition to discover which of its steps determine the performance of its execution and prioritising them, reasoning on an optimal resource allocation under the given resource, costs, and quality elasticity constraints, monitoring the actual service execution, and balancing load on the resources.

In this paper, we present selected results from our ongoing research on elastic processes, more precisely the *Vienna Platform for Elastic Processes (ViePEP)*, which is a research-driven, prototypical BPMS capable to execute elastic processes, monitor the current utilization of invoked resources as well as reason about future resource demands, and carry out necessary actions.

The remainder of this paper is organized as follows: After a brief overview of the related work (Section 2), we will present the overall ViePEP architecture

and its functionalities (Section 3). Subsequently, we will give some information about our work on reasoning mechanisms (Section 4). Section 5 concludes this paper.

2 Related Work

While little effort has been put into the investigation of elastic processes, there is nevertheless fundamental work in related areas, which needs to be regarded.

First, scalability and cost-effective allocation of single tasks and applications have been studied by many researchers. The earliest research efforts often focused on minimizing Cloud consumer's costs while taking into account maximum allowed execution time [7] while later approaches considered holistic Service Level Agreement (SLA) enforcement [5]. Recently, research efforts have paid special attention to the infrastructure perspective, i.e., the adherence to consumer-defined SLAs under the objective of profit maximization [14] or high resource utilization [9,13,16]. While most of these approaches apply threshold-based fixed rules to identify necessary actions (e.g., stop/start servers, move services), Li and Venugopal [16] make use of a learning-based approach to automatically scale an application up or down based on incoming workload. To the best of our knowledge, existing approaches to scalable Cloud applications and cost-effective allocation lack a process perspective across utilized resources. Instead, allocation is performed based on present service requests, but information about possible future requests derived from the description of elastic processes is not taken into account.

Second, Cloud resources have been used for executing SWFs [2,11,12]. In SWFs, SLAs are typically not as much a concern as they are for business processes; in fact, the most common SLOs regarded in SWFs are the actual costs of a workflow invocation or earliest finishing time of the complete workflow [19]. From an execution point of view, SWFs are dataflow-oriented, i.e., the execution control follows the dataflow. In contrast, in business workflows, the execution control is explicitly modeled, making the integration of some workflow patterns easier but hampering the concurrent processing of data items [17]. In SWFs, (data-related) interdependencies between workflow instances occur very often, while in business workflows, it is quite common that a large number of independent workflow instances are carried out at the same time [17]. In the context of the work at hand, this allows for a higher degree of freedom, as service instances may be carried out on different machines without the need to make a potentially very large amount of data available to a particular machine.

3 Vienna Platform for Elastic Processes

3.1 System Overview

ViePEP aims at supporting the complete process/workflow lifecycle as presented, e.g., by Hallerbach et al. [10]. Common steps of process lifecycles are *Design and*

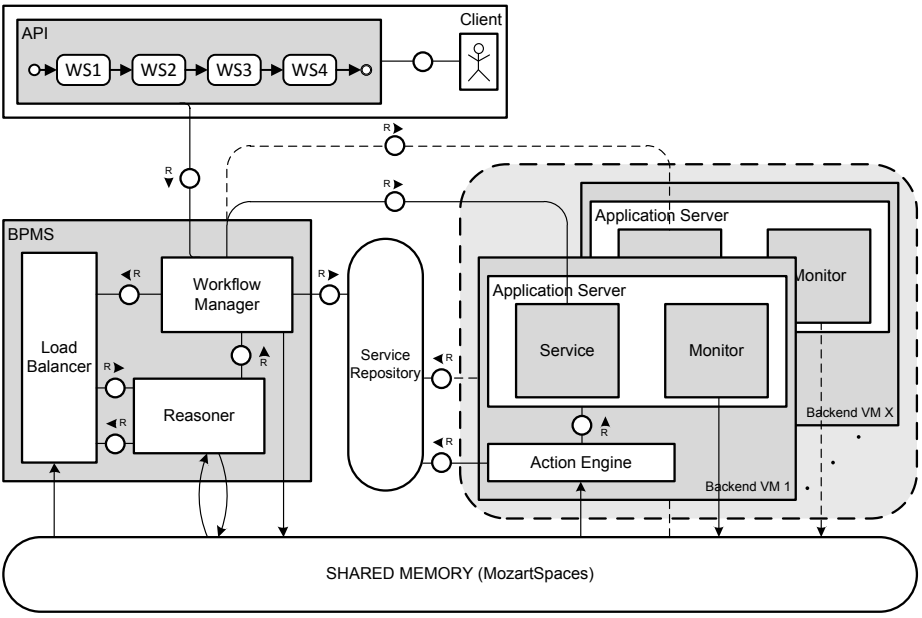

Fig. 1. Workflow Deployment and Cloud Provisioning in ViePEP

Modeling, *Instantiation and Selection*, *Execution and Monitoring*, and *Maintenance/Optimization*.

As depicted in Figure 1 (using an FMC Block Diagram), there are five top level entities within ViePEP: First, the *Client* models service-based workflows and defines the necessary SLOs through an Application Programming Interface (API). At the moment, workflows are modeled using an XML-based description template that also defines non-functional constraints and preferences for each step. This description is handed over as a workflow request to the *Workflow Manager* (WfM) of the BPMS in order to instantiate and execute a workflow. A Client may request execution of multiple workflows at the same time. Second, the *BPMS*, performs the central functionalities of controlling the service infrastructure, including the Cloud resources leased from the provider as well as scheduling workflows over service instances. Thus, the BPMS balances the load across the service instances.

Third, a *Backend VM* hosts an instance of a particular service. In a typical ViePEP-based system, many Backend VMs exist at the same time and are controlled by the BPMS. The BPMS and the Backend VM are the central entities in ViePEP and will be discussed in detail in the next section. Fourth, the *Shared Memory* is used to provide data sharing between the BPMS and the different Backend VMs. We chose MozartSpaces[1] for this, as it allows to easily deploy and access a peer-to-peer-based, distributed database. Last but not least, the

[1] `http://www.mozartspaces.org/`

Service Repository hosts service descriptions as well as their implementations as portable archive files, which enables the BPMS to search for services and deploy them on an arbitrary ViePEP Backend VM.

3.2 ViePEP BPMS and Backend VMs

As can be seen in Figure 1, the BPMS features three components: The already mentioned *Workflow Manager*, the *Load Balancer*, and the *Reasoner*. We will introduce these components in the following:

Workflow Manager: The WfM controls and schedules workflow and service executions. It gets the necessary workflow data, i.e., information about the single steps in a workflow and the accompanying SLOs as a *workflow request* from the Client. Afterwards, it looks up services functionally matching the workflow steps in the Service Repository and maps services and workflow steps. This mapping is needed in order to identify already running services instances through the Load Balancer. Based on this information, the WfM is able to issue *service invocation requests* to a particular Backend VM hosting this service and possessing enough resources to serve this invocation under given QoS constraints as defined by the SLOs.

To execute a workflow and its services, a *Workflow Executor* (not depicted in Figure 1) is started. Based on the workflow request and a workflow/service scheduling obtained from the Reasoner (see below), the Executor queries the Load Balancer for the best fitting service instances (Backend VMs) for the workflow steps. Through the executor, the WfM is also able to measure the service response time. This functionality is needed in order to identify deviations from the expected service behavior at an early point of time and avoid breaches of SLAs, which can lead to penalties [15]. Response time includes the service invocation time as well as the network latency. To log service invocations, the response time is stored in the Shared Memory.

In case of a deviation, the WfM may perform replanning for the workflow [4]. Furthermore, the Reasoner needs to take into account this updated information about the current process landscape. Hence, the WfM provides an interface allowing the Reasoner to get information about the number and kind of workflows and services in the queue, the related QoS constraints and preferences, the number of executors, workflows, and services currently running, and how many service instances have to be invoked at what point of time. Furthermore, information about new user-issued workflow requests as well as occurring and likely deviations in non-functional service behavior are provided to the Reasoner.

Load Balancer: As the name implies, this component balances the load on the Backend VMs and thus makes sure that the utilization of a Backend VM does not exceed a critical (upper) threshold. Importantly, the Load Balancer is a passive component. On request, it prepares and provides information from the Shared Memory to the WfM and Reasoner, but it does not control the Backend VMs by itself.

The Load Balancer is invoked by the WfM in order to identify the best fitting Backend VM for a service request. For this, it retrieves the actual Backend VM states (in terms of occupied CPU and RAM resources) for the service from the Shared Memory and takes into account scheduling information about current and future service invocations provided by the Reasoner. Based on this information, the Load Balancer links the service request from the WfM to a particular service instance running on a Backend VM.

At the moment, the Load Balancer makes use of a rules-based approach to determine the best fitting Backend VM: Service requests are linked to this VM instance where the degree of utilization is closest to a predefined upper threshold. Thus, the Load Balancer allows the WfM to invoke services until a VM's load reaches this threshold. If the Load Balancer is not able to allocate a service request to a VM, e.g., because all VMs are overloaded or there is currently no VM running this service, it gives this information back to the Workflow Executor, which then may conduct a replanning as mentioned above or trigger a new reasoning.

The Load Balancer is also invoked by the Reasoner in order to determine the best fitting Backend VM for an action. For example, the Reasoner may decide that it is necessary to duplicate a Backend VM, as the service provided by it needs to be invoked too often as if a single VM would be able to handle it in a particular time span. Usually, the VM to be duplicated will be the one with the least load, since the duplication action will produce some additional CPU load. As another example, the Reasoner may decide that a Backend VM is not required anymore as the requests of the hosted service can be handled by another Backend VM. In this case, the Load Balancer will select the VM with the least number of current service invocations; furthermore, it will effectively block further invocations of this service instance, i.e., if a Workflow Executor sends a particular service request to the Load Balancer, the VM to be terminated will not be regarded anymore. As it is a passive component, the actual command to duplicate or terminate a VM will be issued by the Reasoner, not the Load Balancer.

Reasoner: While the WfM controls the execution of single workflows, the Reasoner is responsible for the optimization of the complete process and (Cloud) system landscape. It finds a scheduling for workflows and the included steps (services) under the given cost, resource, and quality constraints and forwards this scheduling to the WfM, which itself will invoke workflows and services based on it. Scheduling is directly related to the control and shaping of the Backend VMs, i.e., the decision to start, terminate, and duplicate Backend VMs, move a service from one VM to another, or exchange the service running on a particular VM. The Reasoner needs to take into account knowledge about the currently and future running workflows and their QoS constraints from the WfM. It also considers deviations from the expected workflow execution behavior in order to find an appropriate countermeasure. The Reasoner gets information about the currently free VM resources (CPU and RAM usage) from the Shared Memory and communicates with the Load Balancer in order to decide whether a

particular Backend VM is sufficient to carry out a service, if another VM hosting that service needs to be started (duplication), or if a VM can be terminated due to low load.

At the core of the Reasoner, an elastic reasoning mechanism (ESM) is employed. In our opinion, the investigation of mechanisms which are capable to take into account resource, costs, and quality elasticity and reason about optimal resource allocation is a major research issue for elastic processes. Even though the focus of this paper is not on reasoning, we will present information about possible approaches in Section 4.

While the BPMS controls the process and (Cloud) system landscape, the actual service execution is done on Backend VMs. Each VM provides Software as a Service (SaaS) in terms of a particular Web service, which can be requested and invoked by the WfM. As can be seen in Figure 1, a Backend VM features two major components: The *Application Server* and the *Action Engine*.

Action Engine: The Action Engine is responsible to execute commands to the Application Server. It gets according commands from the Reasoner through the Shared Memory data structure. The most important commands in the context of the work at hand are [16]:

Start a new Backend VM: An empty VM is started by the Reasoner by issuing a corresponding command to the Cloud infrastructure hosting the Backend VMs (in our case: OpenStack; not depicted in Figure 1). When the Backend VM is running, the Action Engine obtains the needed service from the Service Repository and deploys it on the Application Server.

Terminate the Backend VM. Again, the according command is issued by the Reasoner. If the Action Engine receives the command to terminate itself, it first requests information from the Application Server about currently running service invocations. If there are any, the Action Engine regularly polls the Application Server until all service invocations have been finished. The Action Engine unregisters the VM by pushing according status information to the Shared Memory, and finally terminates the VM. Afterwards, the Reasoner and Load Balancer will not take this Backend VM into account anymore.

Duplicate an existing Backend VM. If the Reasoner determines that the resources on the Backend VM hosting a particular service are not sufficient, it can issue an Action Engine the command to duplicate itself. For this, the Action Engine will start a new Backend VM which hosts the same service.

Exchange the hosted service by another service. In some cases, the hosted service is not needed anymore, as there will be no further invocations in the (near) future. However, another service needs to be started. In order to speed up the deployment of a Backend VM running this service, it makes sense to reuse the former Backend VM by exchanging the service running on it. To exchange a service, the Action Engine behaves similarly to the termination of a Backend VM: First, it is checked if the provided service is currently

invoked; if this is the case, the Backend VM waits until the invocations have been finished. Second, the current service is replaced by another service from the Service Repository.

Move a running service to another Backend VM: The Action Engine is able to copy the whole system state and move it to another server. Again, running service invocations need to be finished first.

Notably, the delays occurring because a service invocation is still running when terminating a Backend VM or exchanging/moving a service need to be taken into account by the Reasoner. Furthermore, the Load Balancer needs to consider that a service cannot be invoked any further when it is planned to terminate the Backend VM or exchange/move the service instance.

Application Server: In order to host a Web service, a Backend VM needs an Application Server capable to run it. At the moment, we employ Apache Tomcat, but it is possible to switch to any other J2EE application server like Glassfisch or JBoss. The Application Server comprises two components, namely the actual *Service* and a *Monitor*:

Service: As written above, services are stored in the Service Repository. To host a service within the Application Server, the Action Engine retrieves the according *Web application ARchive* (WAR)-file from the repository and deploys it. In the current version we support any RESTful Web service which can be called using an `HTTP GET` request or can be invoked using a remote procedure call.

Monitor: As explained above, the BPMS makes use of information about a Backend VM's resources in terms of CPU and RAM utilization. Hence, ViePEP-enabled Backend VMs feature an Application Server Monitor. Monitoring is conducted on a Platform as a Service (PaaS) level, i.e., the CPU and RAM utilization is measured for the VM, but not the underlying infrastructure. We apply `psi-probe`[2] as server monitoring tool. Monitor data is stored in the Shared Memory.

4 Elastic Reasoning Mechanisms

An ERM is at the heart of a Reasoner and therefore responsible for the scheduling of workflows/services and allocation of (Cloud) resources. As mentioned in our former work [8], an ERM decides how to utilize resources in an optimal way under multi-dimensional constraints. It takes into account dynamic resource, cost, and workflow information and provides not only a scheduling, but also controls Cloud resources by triggering different actions like moving services from one VM to another or starting and terminating servers. In the following, we briefly discuss different concerns that influence the development and success of an ERM:

[2] `http://code.google.com/p/psi-probe/`

Exact vs. Heuristic Reasoning: The usage of linear, mixed or branch-and-bound integer programming, dynamic programming, or multi-constrained optimal path selection is a natural choice for multi-objective optimization and therefore ERMs [26]. As multi-objective optimization is necessarily an NP-hard problem, the so-far mentioned exact approaches may not lead to a solution in polynomial time. Hence, (meta-)heuristic approaches like greedy algorithms, genetic algorithms, or modified integer programming also need to be taken into consideration. While such optimization approaches have also been used in QoS-aware service compositions, as presented, e.g., in [15,23,24,26], these composition algorithms do not take into account resources. Instead, they are based on QoS assurances given by the service providers.

Apart from the mentioned "classic" optimization approaches, it is also possible to apply an approach to find patterns in workflow requests and relate them to resource demand in order to generate rules for resource allocation and workflow/service scheduling. In general, Machine Learning algorithms are capable to generate such rules [27].

Global vs. Local Reasoning: Currently, ViePEP allows ERM through one central reasoner. Such an approach is deemed "global", as it controls the complete system landscape. However, there might be situations where a global optimization is not efficient or not even possible because a central entity is not able to accumulate the data necessary for the global approach. In such cases, it is helpful to make use of a decentralized, local reasoning, which only takes into account the requirements of either a single workflow or a single Cloud resource (Backend VM).

Continuous vs. Interval Reasoning: A continuously operating Reasoner is triggered whenever a change in the ERM input data (workflow requests, monitored service and resource behavior) is assessed to be a significant event, i.e., makes it necessary to perform reasoning under changed constraints. If reasoning is done in predefined time intervals, changes within the system landscape (e.g., new workflow requests or an underperforming Backend VM) are not directly taken into account, but regarded within the next reasoning cycle.

The decision which approach to follow is directly related to the runtime performance of the ERM; if the ERM is relatively slow, it should be invoked in regular intervals; if the ERM is fast enough to produce output before the next significant event appears, a continuous approach is possible. In this context, the possibility to assess if an event is significant or not is crucial.

Hybrid forms of continuous and interval reasoning are also possible. The general workflow scheduling could be done in certain intervals, as its computation usually requires some time. The continuous reasoning could then be used in order to react to the mentioned significant events. In this case, the continuous reasoning would not be done for the complete system landscape but solely be responsible for compensating negative significant events like the aforementioned underperforming Backend VM, thus being some kind of local reasoning. Of course, the local reasoning needs to be taken into account in the next global reasoning cycle.

In the end, all ERM approaches are generally capable to build a model of an ideal future system state with regard to necessary service scheduling and resource allocation actions. It will be a major task of our future work to implement and evaluate according ERMs.

5 Conclusions

Online business processes are faced with varying workloads that require agile deployment of computing resources. One way to reach this goal is the usage of Cloud resources in elastic processes, which take into account resource, cost, and quality elasticity. Within this paper, we have introduced the *Vienna Platform for Elastic Processes* (ViePEP). This platform aims at supporting the complete process lifecycle by allowing consumers to model and request elastic processes, providing a BPMS able to select appropriate services for the single steps of the defined workflows, reason about the optimal scheduling of workflow and service invocations and assignment of resources to the single services, and execute and monitor them.

As ViePEP is a research prototype, not all the functionalities needed for the complete process lifecycle have been thoroughly implemented yet. Most importantly, the provision of feedback about workflow executions to the Client has not been examined in detail. This information could indicate how to optimize process templates with regard to their non-functional requirements (and thus addresses the Maintenance/Optimization steps of a process lifecycle). Nevertheless, the BPMS and Backend VMs as presented are fully functional and have been implemented using the OpenStack IaaS Cloud computing framework. Workflows are executed according to their QoS requirements and Backend VMs hosting service instances are automatically started and terminated based on the non-functional needs of the workflows. Cloud control and scheduling is provided through a first, rather simple reasoning algorithm. A short demo of ViePEP can be found at `http://www.infosys.tuwien.ac.at/prototypes/ViePEP/ViePEP_index.html` [25].

In the future, we will further extend the functionalities of ViePEP and most importantly work on different reasoning algorithms as indicated in Section 4. In fact, we primarily see ViePEP as a research tool helping us to investigate reasoning mechanisms for elastic processes.

Acknowledgements. This work is partially supported by the Austrian Science Fund (FWF): P23313-N23 and the Commission of the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201).

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. ViePEP is partly based on the work presented by Li and Venugopal [16].

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business Process Management: A Survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
2. Abouelhoda, M., Issa, S.A., Ghanem, M.: Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. BMC Bioinformatics 13(77) (2012)
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A View of Cloud Computing. Communications of the ACM 53, 50–58 (2010)
4. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Dynamic Replanning of Web Service Workflows. In: Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007), pp. 211–216. IEEE Computer Society, Washington, DC (2007)
5. Buyya, R., Ranjan, R., Calheiros, R.N.: InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In: Hsu, C.-H., Yang, L.T., Park, J.H., Yeo, S.-S. (eds.) ICA3PP 2010, Part I. LNCS, vol. 6081, pp. 13–31. Springer, Heidelberg (2010)
6. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computing Systems 25(6), 599–616 (2009)
7. Cao, Q., Wei, Z.B., Gong, W.M.: An Optimized Algorithm for Task Scheduling Based on Activity Based Costing in Cloud Computing. In: 3rd International Conference on Bioinformatics and Biomedical Engineering (ICBBE 2009), pp. 1–3. IEEE Computer Society, Washington, DC (2009)
8. Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of Elastic Processes. IEEE Internet Computing 15(5), 66–71 (2011)
9. Emeakaroha, V.C., Brandic, I., Maurer, M., Breskovic, I.: SLA-Aware Application Deployment and Resource Allocation in Clouds. In: COMPSAC Workshops 2011, pp. 298–303. IEEE Computer Society, Washington, DC (2011)
10. Hallerbach, A., Bauer, T., Reichert, M.: Managing Process Variants in the Process Life Cycle. In: Tenth International Conference on Enterprise Information Systems (ICEIS 2008), vol. ISAS-2, pp. 154–161 (2008)
11. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the Use of Cloud Computing for Scientific Workflows. In: IEEE Fourth International Conference on e-Science (eScience 2008), pp. 640–645. IEEE Computer Society, Washington, DC (2008)
12. Juve, G., Deelman, E.: Scientific Workflows and Clouds. ACM Crossroads 16(3), 14–18 (2010)
13. Kertesz, A., Kecskemeti, G., Brandic, I.: An Interoperable and Self-adaptive Approach for SLA-based Service Virtualization in Heterogeneous Cloud Environments. Future Generation Computer Systems NN(NN), NN–NN (2013) (forthcoming)
14. Lee, Y.C., Wang, C., Zomaya, A.Y., Zhou, B.B.: Profit-Driven Service Request Scheduling in Clouds. In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010), pp. 15–24. IEEE Computer Society, Washington, DC (2010)
15. Leitner, P., Hummer, W., Dustdar, S.: Cost-Based Optimization of Service Compositions. IEEE Transactions on Services Computing (2012)
16. Li, H., Venugopal, S.: Using Reinforcement Learning for Controlling an Elastic Web Application Hosting Platform. In: 8th International Conference on Autonomic Computing (ICAC 2011), pp. 205–208. ACM, New York (2011)

17. Ludäscher, B., Weske, M., McPhillips, T., Bowers, S.: Scientific Workflows: Business as Usual? In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 31–47. Springer, Heidelberg (2009)
18. Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the Effectiveness of Process-Oriented Information Systems: Problem Analysis, Critical Success Factors, and Implications. IEEE Transactions on Systems, Man, and Cybernetics, Part C 38(3), 280–291 (2008)
19. Pandey, S., Wu, L., Guru, S.M., Buyya, R.: A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. In: 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010), pp. 400–407. IEEE Computer Society, Washington, DC (2010)
20. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B.J.: Service-Oriented Computing Research Roadmap. In: Service Oriented Computing (SOC). Dagstuhl Seminar Proceedings, vol. 05462, pp. 38–45. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
21. Rohjans, S., Dänekas, C., Uslar, M.: Requirements for Smart Grid ICT Architectures. In: Third IEEE PES Innovative Smart Grid Technologies (ISGT) Europe Conference. IEEE Computer Society, Washington, DC (2012)
22. Rusitschka, S., Eger, K., Gerdes, C.: Smart Grid Data Cloud: A Model for Utilizing Cloud Computing in the Smart Grid Domain. In: 1st IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 483–488. IEEE Computer Society, Washington, DC (2010)
23. Schuller, D., Lampe, U., Eckert, J., Steinmetz, R., Schulte, S.: Cost-driven Optimization of Complex Service-based Workflows for Stochastic QoS Parameters. In: 19th International Conference on Web Services (ICWS 2012), pp. 66–74. IEEE Computer Society Press, Washington, DC (2012)
24. Schuller, D., Polyvyanyy, A., García-Bañuelos, L., Schulte, S.: Optimization of Complex QoS-Aware Service Compositions. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 452–466. Springer, Heidelberg (2011)
25. Schulte, S., Hoenisch, P., Venugopal, S., Dustdar, S.: Realizing Elastic Processes with ViePEP. In: Zhu, H., Ghose, A., Yu, Q., Perrin, O., Wang, J., Wang, Y., Delis, A., Sheng, Q.Z. (eds.) ICSOC 2012, vol. 7759, pp. 439–442. Springer, Heidelberg (2013)
26. Strunk, A.: QoS-Aware Service Composition: A Survey. In: IEEE 8th European Conference on Web Services (ECOWS), pp. 67–74. IEEE Computer Society, Washington, DC (2010)
27. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann Publishers, San Francisco (2005)