Vienna University of Technology
Information Systems Institute
Distributed Systems Group

Elastic Process Optimization – The Service Instance Placement Problem

P. Hoenisch, D. Schuller, C. Hochreiner, S. Schulte, S. Dustdar

p.hoenisch@infosys.tuwien.ac.at

TUV-1841-2014-01 Oct. 31, 2014

The Vienna Platform for Elastic Processes (ViePEP) is a research Business Process Management System (BPMS) which additionally provides the functionalities of a cloud resource controller. As the name implies, the system is able to plan, schedule, and enact elastic processes in the cloud. ViePEP allows the integration of different optimization approaches, which could aim, e.g., at minimum makespans, minimum costs, or a combination thereof. Within this Technical Report we present an optimization model to tackle the challenges of scheduling service invocations among cloud-based computational resources. This specific optimization approach – the Service Instance Placement Problem – considers different kinds of QoS attributes and aims at cost-efficiency.

Keywords: Cloud Computing, Elastic Processes, Optimization, Scheduling, Business Process Management

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
http://www.infosys.tuwien.ac.at/

Elastic Process Optimization – The Service Instance Placement Problem

Philipp Hoenisch*, Dieter Schuller†, Christoph Hochreiner*, Stefan Schulte*, Schahram Dustdar*
*Vienna University of Technology, Austria
Email: {p.hoenisch, s.schulte, c.hochreiner, dustdar}@infosys.tuwien.ac.at †Technische Universität Darmstadt, Germany
Email: schuller@kom.tu-darmstadt.de

Abstract—The Vienna Platform for Elastic Processes (ViePEP) is a research Business Process Management System (BPMS) which additionally provides the functionalities of a cloud resource controller. As the name implies, the system is able to plan, schedule, and enact elastic processes in the cloud. ViePEP allows the integration of different optimization approaches, which could aim, e.g., at minimum makespans, minimum costs, or a combination thereof.

Within this *Technical Report* we present an optimization model to tackle the challenges of scheduling service invocations among cloud-based computational resources. This specific optimization approach – the Service Instance Placement Problem – considers different kinds of QoS attributes and aims at cost-efficiency.

1 Introduction

In recent years, Business Process Management (BPM) evolved to an important factor in many companies [8]. Business processes are composed of human and software services to realize a specific business logic, functionality or service. Managing the execution of such business processes in an automatic way is a prominent field of research and various concepts, methodologies and (software) frameworks have been proposed to tackle these challenges. In the field of computer science, solutions have been proposed where web service technologies and service compositions are used to model and execute business processes automatically [6], [7].

However, BPM may span several companies and is responsible for providing services to a variety of users. Hence, BPM has to be able to handle potentially thousands of process requests simultaneously [1]. Further, while some processes are requested on a fixed interval, others may be requested rather ad-hoc and have a higher priority. In combination, with the complexity of handling hundreds of services, a BPMS is needed which is able to control a process landscape in a proactive and reactive way. This means that the BPMS has to schedule process executions, lease and release resources in advance, execute processes and has to be able to find countermeasures in the case of failures.

In our former work, we presented ViePEP – a framework which is such a BPMS and is able to control a complex process landscape. ViePEP makes use of Cloud-based computational resources for deploying software services, executing processes, monitor ing their executions. Thus, the platform is able to enact *elastic processes*, i.e., business processes carried out using elastic cloud resources [2]. ViePEP is able to reduce the risk of over- and under-provisioning of resources, while guaranteeing the needed level of Quality of Service (QoS) [3], [4], [9], [10]. To find an optimal scheduling and resource allocation for elastic processes, an elastic reasoning mechanism (ERM) is needed [2]. This ERM can be based on different approaches, e.g., linear optimization, genetic algorithms, or other heuristics.

Within this *Technical Report* we present one possible solution approach for the ERM. For this, we apply mixed integer linear programming (MILP), following a worst-case analysis. The resulting *Service Instance Placement Problem* (SIPP) can be solved using a solver like CPLEX[1]. The SIPP allows to handle complex processes and schedule their executions among cloud-based computational resources in a cost-efficient way while considering given Service Level Agreements (SLAs).

2 Optimization problem

The following table describes an optimization model to solve the SIPP. Through the application of this model, it is possible to minimize the occurring costs for leased resources, i.e., Virtual Machines (VMs) and SLA penalties which could accrue due to delayed process instances.

The first column in Table 1 comprises the objective function for the SIPP. The objective function is subject to minimize the overall cost in terms of leasing and

1. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

penalty cost. Table 2 presents the variables used in Table 1. The objective function comprises 4 terms:

The first term $\sum_{v \in V} c_v \cdot \gamma_{(v,t)}$ computes the total cost which accrue when $\gamma_{(v,t)}$ VM instances of type v are leased in a certain time period t. Each VM type may have different leasing cost per time period (Billing Time Units – BTUs) which is expressed as c_v. The second term $\sum_{w \in W} \sum_{i_p \in I_p} c^p_{i_p} \cdot e^p_{i_p}$ computes the cost which accrue if deadlines of process instances are violated, i.e., a process instance is delayed. The penalty cost have to be paid per delayed process instance. For computing these penalties we apply a linear function as described in [5]: Each process instance comprises cost per time unit which have to be paid if it gets delayed, i.e., $c^p_{i_p}$. This value is multiplied with the actual period for which the process instance is delayed, i.e., with $e^p_{i_p}$.

The third term, i.e., $\sum_{v \in V} \sum_{k_v \in K_v} (\omega^C_f \cdot f^C_{k_v} + \omega^R_f \cdot f^R_{k_v})$ is used to reduce the overall cost of *unused* resources. Unused resources are defined as the sum of free resource capacities $f^C_{k_v}$ in terms of CPU and $f^R_{k_v}$ in terms of RAM for all leased VM instances. Free resource capacities are multiplied with certain weights ω^C_f and ω^R_f, respectively, to be not too dominant within the whole model.

The fourth term, i.e., $\sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in J^*_{i_p}} \frac{1}{DL_{i_p} - \tau_t} x_{(j_{i_p}, k_v, t)}$ is used to compute the urgency of process instances. For that, we subtract the current point in time τ_t from the deadlines DL_{i_p} and compute the corresponding reciprocal value. As this value ($\frac{1}{DL_{i_p} - \tau_t}$) gets larger the closer the deadline is, more urgent process instances are assigned a higher priority.

Beside of the objective function, the SIPP consists of several other constraints which help to minimize the overall cost. In addition, after solving the SIPP model, the involved variables provide detailed instructions what service (of a certain service type) should be deployed onto which VM and what service instance should be invoked for a particular process instance step.

REFERENCES

[1] Breu, R., Dustdar, S., Eder, J., Huemer, C., Kappel, G., Köpke, J., Langer, P., Mangler, J., Mendling, J., Neumann, G., Rinderle-Ma, S., Schulte, S., Sobernig, S., Weber, B.: Towards Living Inter-Organizational Processes. In: 15th IEEE Conf. on Business Informatics (CBI 2013). pp. 363–366. IEEE (2013)

[2] Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of Elastic Processes. IEEE Internet Computing 15(5), 66–71 (2011)

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

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

[5] Leitner, P., Hummer, W., Dustdar, S.: Cost-Based Optimization of Service Compositions. IEEE Trans. on Services Computing 6(2), 239–251 (2013)

[6] Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the Effectiveness of Process-Oriented Information Systems: Problem Analysis, Critical Success Factors, and Implications. IEEE Trans. on Systems, Man, and Cybernetics, Part C 38(3), 280–291 (2008)

[7] Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B.J.: Service-Oriented Computing Research Roadmap. In: Service Oriented Computing (SOC). pp. 38–45. No. 05462 in Dagstuhl Seminar Proceedings, Intern.es Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006)

[8] Rosemann, M., vom Brocke, J.: The Six Core Elements of Business Process Management. In: Handbook on Business Process Management 1, pp. 107–122. Springer-Verlag Berlin Heidelberg (2010)

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

[10] Schulte, S., Schuller, D., Hoenisch, P., Lampe, U., Dustdar, S., Steinmetz, R.: Cost-Driven Optimization of Cloud Resource Allocation for Elastic Processes. Intern. J. of Cloud Computing 1(2), 1–14 (2013)

TABLE 1: Service Instance Placement Problem – Constraints

Constraint	Description
$\min \sum_{v \in V} c_v \cdot \gamma_{(v,t)} + \sum_{w \in W} \sum_{i_p \in I_p} c^p_{i_p} \cdot e^p_{i_p} + \sum_{v \in V} \sum_{k_v \in K_v} (\omega^C_f \cdot f^C_{k_v} + \omega^R_f \cdot f^R_{k_v}) - \sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in J^*_{i_p}} \frac{1}{DL_{i_p} - \tau_t} x_{(j_{i_p},k_v,t)}$	This is the overall minimization function. For its description see above the textual description
$\tau_{t+1} + e_{i_p} + e^{run}_{j_{i_p}} \leq DL_{i_p} + e^p_{i_p}$	This constraint demands the deadlines for the single process instances not to be violated for all process instances
$\tau_{t+1} \geq \tau_t + \epsilon$	This constraint computes the starting point of the next optimization period
$e_{i_p} = e^{seq}_{i_p} + e^{L_a}_{i_p} + e^{L_x}_{i_p} + e^{RL}_{i_p}$	Computes the remaining execution time for a whole process instance considering the different durations for sequences, AND-Blocks, XOR-Blocks and Loops
$e^{seq}_{i_p} = \hat{e}^s_{i_p} - ex_{j^*_{i_p}}$, if $x_{(j^*_{i_p},k_v,t)} = 1$ $e^{seq}_{i_p} = \hat{e}^s_{i_p}$, else	Computes the remaining execution time for a sequence of process steps
$e^{L_a}_{i_p} = \max_{l \in L_a}(\hat{e}^l_{i_p} - ex_{j^*_{i_p}})$, if $x_{(j^*_{i_p},k_v,t)}$ $e^{L_a}_{i_p} = \max_{l \in L_a}(\hat{e}^l_{i_p})$, else	Computes the remaining execution time for an AND-Block while taking the maximum of the involved paths
$e^{L_x}_{i_p} = \max_{l \in L_x}(\hat{e}^l_{i_p} - ex_{j^*_{i_p}})$, if $x_{(j^*_{i_p},k_v,t)}$ $e^{L_x}_{i_p} = \max_{l \in L_x}(\hat{e}^l_{i_p})$, else	Computes the remaining execution time for an XOR-Block while taking the maximum of the involved paths
$e^{RL}_{i_p} = re \cdot \hat{e}^s_{i_p} - ex_{j^*_{i_p}}$, if $x_{(j^*_{i_p},k_v,t)}$ $e^{RL}_{i_p} = re \cdot \hat{e}^s_{i_p}$, else	Computes the remaining execution time for a Loop while taking the maximum of all sub-paths multiplied by the maximum of iterations
$ex_{j^*_{i_p}} = \sum_{v \in V} \sum_{k \in K_v} ((e_{j^*_{i_p}} + \Delta_{j^*_{i_p}} + \Delta) x_{(j^*_{i_p},k_v,t)})$	Computes the variable $ex_{j^*_{i_p}}$ which involves the remaining execution time, the remaining VM-startup time, and the remaining deployment time
$\hat{e}^s_{i_p} = \sum_{j_{i_p} \in J^{seq}_{i_p}} (e_{j_{i_p}} + \Delta_{j_{i_p}} + \Delta)$	Computes the variable $\hat{e}^s_{i_p}$ which involves the remaining execution time, the remaining VM-startup time, and the remaining deployment time
$\hat{e}^l_{i_p} = \sum_{j_{i_p} \in J^l_{i_p}} (e_{j_{i_p}} + \Delta_{j_{i_p}} + \Delta)$	Computes the variable $\hat{e}^l_{i_p}$ which involves the remaining execution time, the remaining VM-startup time, and the remaining deployment time
$\sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in J_{i_p}} x_{(j_{i_p},k_v,t)} \leq (\beta_{(k_v,t)} + y_{(k_v,t)}) \cdot M$	Demands that if a service invocation is scheduled on a certain VM, the VM either has to be already running, or has to be started
$x_{(j^1_{i^1_{p^1}},k_v,t)} + x_{(j^2_{i^2_{p^2}},k_v,t)} \leq 1$	Demands that two different services types are not allowed to be run on the same VM
$\sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in (J^*_{i_p} \cup J^{run}_{i_p})} r^C_{(j_{i_p},k_v)} x_{(j_{i_p},k_v,t)} \leq s^C_v$	Demands that the host VM has enough resources in terms of CPU for all assigned service invocations
$g_{(k_v,t)} \cdot s^C_v - \sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in (J^*_{i_p} \cup J^{run}_{i_p})} r^C_{(j_{i_p},k_v)} x_{(j_{i_p},k_v,t)} \leq f^C_{k_v}$	Computes the amount of free resources in terms of CPU of a specific VM
$\sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in (J^*_{i_p} \cup J^{run}_{i_p})} r^R_{(j_{i_p},k_v)} x_{(j_{i_p},k_v,t)} \leq s^R_v$	Demands that the host VM has enough resources in terms of RAM for all assigned service invocations

Continued on next page

TABLE 1 – Continued

Constraint	Description
$g_{(k_v,t)} \cdot s_v^R - \sum_{p \in P} \sum_{i_p \in I_p} \sum_{j_{i_p} \in (J_{i_p}^* \cup J_{i_p}^{run})} r_{(j_{i_p},k_v)}^R x_{(j_{i_p},k_v,t)} \leq f_{k_v}^R$	Computes the amount of free resources in terms of RAM of a specific VM
$g_{(k_v,t)} \geq \beta_{(k_v,t)}$	Helper constraint to check if a VM is already running
$g_{(k_v,t)} \geq y_{(k_v,t)}$	Helper constraint to check if a VM has to be started
$g_{(k_v,t)} \leq \beta_{(k_v,t)} + y_{(k_v,t)}$	Helper constraint to ensure that a VM is either already running or has to be started
$(e_{j_{i_p}} + \Delta_{j_{i_p}} \cdot (1 - z_{(j_{i_p},k_v,t)}) + \Delta \cdot (1 - \beta_{(k_v,t)})) x_{(j_{i_p},k_v,t)} \leq d_{k_v,t} + y_{(k_v,t)} \cdot D$	This constraint demands, that the remaining leasing duration of a certain VM instance needs to be long enough to finish the service invocation and to deploy the service, if required
$e_{j_{i_p}}^{run_{k_v}} \leq d_{k_v,t} + y_{(k_v,t)} \cdot D$	Demands that a VM's remaining leasing duration is long enough to finish all service invocations which have been started in a former time period
$\sum_{k \in K_v} y_{(k_v,t)} \leq \gamma_{(v,t)}$	Computes the amount of leased VMs
$\sum_{v \in V} \sum_{k \in K_v} x_{(j_{i_p},k_v,t)} \leq 1$	Demands that a specific service instance can only be invoked on one VM
$x_{(j_{i_p},k_v,t)} = 1$	Demands that $x_{(j_{i_p},k_v,t)} = 1$ if a service instance is assigned to a specific VM k_v in time period t
$x_{(j_{i_p},k_v,t)} \in \{0,1\}$	Demands that $x_{(j_{i_p},k_v,t)}$ is either 0 or 1
$g_{(k_v,t)} \in \{0,1\}$	Demands that $g_{(k_v,t)}$ is either 0 or 1
$y_{(k_v,t)} \in \mathbb{N}_0$	Defines the limitation for $y_{(k_v,t)}$, i.e., to be a *natural* number
$e_{i_p}^p \in \mathbb{R}^+$	Demands that the variable $e_{i_p}^p$ (the execution time) is a *rational* number
$\hat{e}_{j_{i_p}} = e_{j_{i_p}} + \Delta_{j_{i_p}} + \Delta$	Demands that $\hat{e}_{i_p}^p$ is the sum of the execution time, deployment time, and VM-startup time
$e_{j_{i_p}}^{run} = 0$, if j_{i_p} finished $e_{j_{i_p}}^{run} = \max\left(0, \hat{e}_{j_{i_p}} - (\tau_t - \tau_{t_s})\right)$, else	Computes the remaining execution time of a running service invocation for a certain process step. This value is 0 if the service invocation is already finished, or otherwise represents a certain amount of time

TABLE 2: Service Instance Placement Problem – Variables

Variable	Description
$v \in V = \{1, ..., v^{\#}\}$	V specifies the set of Virtual Machine (VM) types and v is a specific type.
$k_v \in K_v = \{1, ..., k_v^{\#}\}$	K_v specifies the amount of VMs of type v and k_v defines the k^{th} VM instance of type v.
$p \in P = \{1, ..., p^{\#}\}$	P specifies the set of process models and p is a specific process model.
$i_p \in I_p = \{1, ..., i_p^{\#}\}$	I_P is the set of all process instances and i_p represents a specific process instance of process model p.
$j_{i_p}, j_{i_p}^* \in J_{i_p} = \{1, ..., j_{i_p}^{\#}\}$	J_{i_p} is the set of process steps of a process instance i_p which have to be invoked to fulfill i_p. j_{i_p} is a specific process step of the process instance i_p and $j_{i_p}^*$ is the *next* process step of process instance i_p.
${j_{i_p}}^{run} \in J_{i_p}^{run} = \{1, ..., j_{i_p}^{run\#}\}$	$J_{i_p}^{run}$ defines a set of running process steps of process instance i_p and ${j_{i_p}}^{run}$ defines specific running process step of the process instance i_p.
$l \in L = \{1, ..., l^{\#}\}$, L_a, L_x, L_{re}	L indicates the set of all paths and l indicates a specific path within a process. L_a, L_x, L_{re} defines the paths for AND-Blocks, XOR-Blocks or Repeat Loops.
e_{i_p}	e_{i_p} is the remaining execution time of process instance i_p.
$e_{i_p}^p$	Defines the amount of penalties which accrue if the process instance i_p is delayed.
DL_{i_p}	Defines the deadline for the process instance i_p, i.e., a specific point in time represented as the time elapsed since 01/01/1970 in milliseconds.
$e_{j_{i_p}}$, $e_{j_{i_p}}^{run}$, $e_{j_{i_p}}^{run_{k_v}}$	$e_{j_{i_p}}$ is the remaining execution time of step j of process instance i_p. $e_{j_{i_p}}^{run}$ (or $e_{j_{i_p}}^{run_{k_v}}$) is the remaining execution time of the already running process step j of process instance i_p (on the k-th VM of type v).
$ex_{j_{i_p}}$, $ex_{j_{i_p}^*}$	These are helper variables defining the combined remaining execution time, remaining deploy time and VM start-up time if the process step j of process instance i_p is scheduled.
$\hat{e}_{i_p}^l$, $\hat{e}_{i_p}^s$	Defines the combined remaining execution time, remaining deploy time and VM start-up. s defines if this step is part of sequence or Repeat Loop and l defines if this step is part of a complex pattern, e.g., AND-Block, XOR-Block.
$e_{i_p}^{seq}$, $e_{i_p}^{L_a}$, $e_{i_p}^{L_x}$, $e_{i_p}^{RL}$	Defines the execution time for a sequence ($e_{i_p}^{seq}$), AND-Block ($e_{i_p}^{L_a}$), XOR-Block ($e_{i_p}^{L_x}$), or Repeat Loop ($e_{i_p}^{RL}$) for a specific process instance i_p.
t, τ_t, τ_{t+1}, τ_{t_s}	t defines the beginning of a time period, τ_t defines the current time period, and τ_{t+1} defines the next time period, i.e., a point of time in the future and τ_{t_s} defines a specific point of time.
s_v^C, s_v^R	Defines the total resource supply of VM type v in terms of CPU (s_v^C) and RAM (s_v^R).
$f_{k_v}^C$, $f_{k_v}^R$	Defines the available resources of the VM k_v in terms of CPU ($f_{k_v}^C$) and RAM ($f_{k_v}^R$) after subtracting already running or scheduled process steps.

Continued on next page

TABLE 2 – Continued

Variable	Description
$r^C_{(j_{i_p},v)}$, $r^R_{(j_{i_p},v)}$	Defines the required amount of resources in terms of CPU (C) and RAM (R) for a process step j_{i_p} on a VM of type v.
st_j	Defines the service type of the process step j.
Δ_{st_j},$\Delta_{j_{i_p}}$	Defines the time it takes to deploy a service of type st of process step j or of a specific process step j of process instance i_p.
Δ_v, Δ	Δ_v defines the time it takes to start a new VM of type v expressed in milliseconds. Δ defines the max of starting a VM of any type, i.e, $\max_{v \in V}(\Delta_v)$.
$z_{(st_j,k_v,t)}$	Indicates whether a specific service type st_j is deployed on the VM k_v in time period t.
$z_{(j_{i_p},k_v,t)}$	This variable indicates if the service type of a service step j_{i_p} has the same service type as the service instance which is deployed on VM k_v in τ_t.
$x_{(j_{i_p},k_v,t)}$	Defines if the process step j of process instance i_p should be invoked on VM k_v in time period t.
$y_{(k_v,t)}$	Defines how often a VM k_v should be leased in time period t, i.e., how many BTUs.
$g_{(k_v,t)}$	Helper variable indicating if the VM k_v is running in time period t or needs to be started.
$\beta_{(k_v,t)}$	Indicates if the VM k_v was running in time period t.
$\gamma_{(v,t)}$	Defines the amount of leased VMs of type v in time period t.
$d_{(k_v,t)}$, $d_{(k_v,t-1)}$	Defines the remaining leasing duration of VM k_v in time period t or $t-1$.
BTU	The *Billing Time Unit* (BTU) defines *one* leasing duration for which cost apply, i.e., a certain time period in milliseconds.
c_v	Defines the leasing cost of VM type v in BTUs.
re	Defines the maximum amount of repetitions within a Repeat Loop.
$c^p_{i_p}$	Defines the penalty (cost) per time unit of delay for the process instance i_p.
M	M is a constant needed to give some constraints a higher weight.
ϵ	Defines the a short time period in milliseconds which is used to prevent deadlocks.
ω^C_f, ω^R_f	Helper variables, representing constant values to give more weight to the *wasted resources* term in the optimization function.