A Middleware-centric Optimization Approach for the Automated Provisioning of Services in the Cloud

Karolina Vukojevic-Haupt, Santiago Gómez Sáez, Florian Haupt, Dimka Karastoyanova, Frank Leymann
Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

***Abstract*—The on-demand provisioning of services, a cloud-based extension for traditional service-oriented architectures, improves the handling of services in usage scenarios where they are only used rarely and irregularly. However, the standard process of service provisioning and de-provisioning shows still some shortcomings when applying it in real world. In this paper, we introduce a middleware-centric optimization approach that can be integrated in the existing on-demand provisioning middleware in a loosely coupled manner, changing the standard provisioning and de-provisioning behavior in order to improve it with respect to cost and time. We define and implement a set of optimization strategies, evaluate them based on a real world use case from the eScience domain and provide qualitative as well as quantitative decision support for effectively selecting and parametrizing a suitable strategy. Altogether, our work improves the applicability of the existing on-demand provisioning approach and system in real world, including guidance for selecting the suitable optimization strategy for specific use cases.**

Keywords— on-demand provisioning; cloud; service-oriented computing; eScience; optimization; dynamic provisioning; SOC

I. INTRODUCTION

Service-oriented architectures (SOA) typically realize the always-on semantics of services by keeping them constantly up and running [14]. Especially for business applications, this approach suits well. Whenever services are called often and regularly, they are constantly needed and in most cases well utilized. In the eScience domain, the concepts of service-oriented computing have been successfully applied for building and running complex applications [1]. One example are simulation workflows [2], where simulation experiments are modeled as workflows that orchestrate simulation services. These workflows and services are in many cases only executed rarely and irregularly. Keeping services up and running all the time in these cases is a waste of resources. In our previous work in the scope of SimTech (Cluster of Excellence in Simulation Technology), we have introduced the concept of the on-demand provisioning of workflow middleware and services including the underlying infrastructure and middleware (ODP) that aims at solving the mismatch sketched before [3]. The ODP approach extends traditional service-oriented architectures by connecting them to cloud services. It creates a highly dynamic cloud-based runtime environment and at the same time hides all the technical complexity in the ODP middleware. The main idea is that services are only provisioned when they are needed (i.e. they are called) and de-provisioned as soon as they are not needed anymore (i.e. they process no more service calls). Similarly, a workflow middleware is only provisioned when it is needed and de-provisioned as soon as it is not needed anymore. At the time when a user models a workflow, no middleware or services are provisioned or running. The workflow execution is then started with only one click and everything else (provisioning of the workflow middleware and services) is handled automatically and hidden in the background. The ODP approach can be integrated loosely coupled into existing service-oriented architectures.

Although the ODP behavior tackles the shortcomings in traditional SOA systems that we have identified before, in practice it also shows some shortcomings. When the same service is used repeatedly (e.g. it is called from inside a loop), the ODP approach provisions and de-provisions this service multiple times although keeping the service running seems more efficient. In this paper, we introduce a set of optimization strategies that adapt the default ODP behavior to better suit such cases. As a common base for all strategies, we present a loosely coupled extension of our existing ODP architecture to enable the middleware-centric integration of the optimization strategies. We evaluate the identified optimization strategies by applying them to a real-world simulation workflow showing that they can improve the execution in terms of cost as well as time. In order to support the selection of an appropriate optimization strategy and its parametrization, we contribute a qualitative as well as a quantitative decision support method.

II. A MIDDLEWARE-CENTRIC OPTIMIZATION APPROACH

The core functionality of the on-demand provisioning of services has been designed as a middleware component. The main ingredients of the corresponding architecture are shown in Fig. 1 (including the optimization extension, which will be discussed later). All service calls are received and routed by the *ODP service bus*, an enterprise service bus (ESB) extended for ODP. In the original ODP setup, the service bus communicates directly with the *service registry*, a global directory of all available services, as well as with the *provisioning manager*. The provisioning manager realizes all provisioning and de-provisioning related functionality and accesses the *service*

package repository to retrieve service packages (which contain all artifacts needed to provision a service).

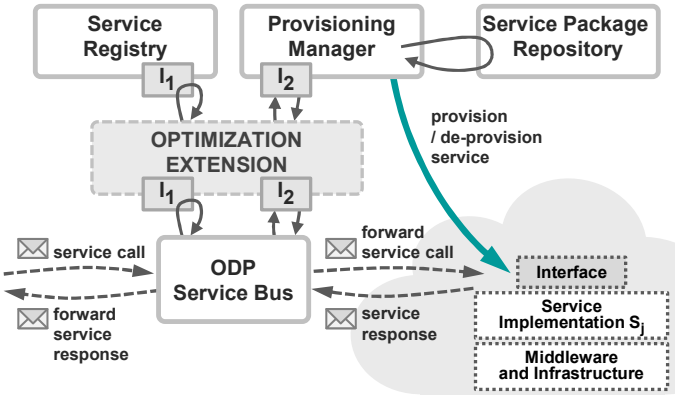

Fig. 1. Integration of the optimization extension

All interactions of the ODP service bus with the other components follow a fixed process [4]. In order to realize optimization strategies, the architecture is extended with an additional *optimization extension* component. This component offers the same interfaces as the service registry (I_1) and the provisioning manager (I_2) and contains the realization of optimization strategies. It acts as a proxy, potentially rerouting, changing, dropping, or delaying any requests it receives. This design enables a seamless and loosely coupled integration of optimization strategies into the existing system while the existing components stay unchanged. Same as the ODP middleware itself, the optimization extension is independent of any specific workflow language or workflow management system. The optimization strategies introduced in this paper are middleware-centric, i.e. they change the way in which service calls are processed by the ODP middleware. Clearly, there are other aspects of the overall system that can be subject to optimization. Service packages can for example be rewritten or exchanged [5] or workflows can be improved by restructuring them [6]. Such approaches are out of scope of this work. The goal of the optimization strategies, i.e. the relevant metrics considered in this work, is time and cost.

Given the optimization approach and integration architecture described so far, optimization strategies can be grouped in three classes. First, a strategy can change the point in time when a service is provisioned. The general idea is that a service is provisioned before it is needed, so that it can immediately process a service call when it arrives (*prospective provisioning*). Second, whenever the ODP service bus receives a service call, it might hold it back for some time. Collecting service calls and then processing them by one service in a row (*request queueing*) results in high utilization of this service and promises cost reduction. Third, the de-provisioning of a service can be delayed. If a service is kept running, it might be reused by subsequent service calls, which promises to reduce time as well as cost. We will focus on the latter class of optimization strategies to tackle our motivating scenario.

III. Optimization Strategies

In this section, we present a set of five optimization strategies. We will refer to the default (i.e. un-optimized) behavior of the ODP service bus as the *default ODP strategy*.

A. Cost Model Exploitation Strategy

Cloud resources are by definition offered in a pay-as-you-go manner, i.e. users pay only for resources as long as they use them. Today, most Cloud providers offer their resources based on a rather coarse-grained billing interval of one hour. The knowledge about the billing interval for cloud resources can be exploited to optimize the on-demand provisioning of services. The main idea of the cost model exploitation strategy (in short: cost model strategy) is illustrated in Fig. 2. The lower diagram represents the behavior of the default ODP strategy. When a service is called (SC), it is provisioned, processes the service call, sends a result message (SR) and is then de-provisioned (represented by the solid line). As soon as the service is being provisioned, the associated cost (dashed line) increases by the amount due for the just started billing interval. For a second service call, the service is again provisioned and the cost increases again, as the provisioning requests resources and by that starts another billing interval. In the example shown in Fig. 2, the default ODP behavior with two service calls results in the cost of two billing intervals. The upper diagram shows the behavior of the cost model strategy. Upon a service call, a service is provisioned, processes the service call, sends a result message (SR), but is not de-provisioned, as the current billing interval has not yet ended. When the second service call arrives, the service is still running and immediately processes the request without any additional provisioning time or costs. Only when the billing interval ends and the service is not used at this time, the service is de-provisioned. The cost model strategy can reduce the response time (*Δ response time*) as well as the costs (*Δ costs*).

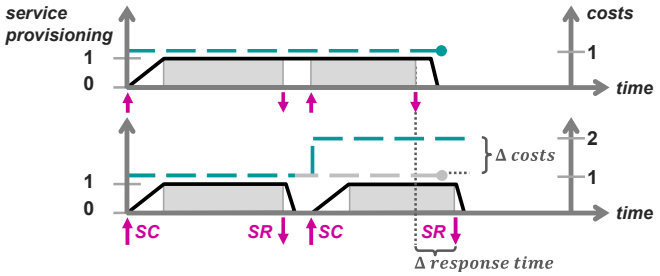

Fig. 2. Cost model exploitation strategy

In this strategy, if a service is not needed anymore, it will not be de-provisioned as long as it does not cause any additional costs. For this, the cost model of the cloud provider has to be known and clocked. In addition, the billing intervals have to be long enough so that it is possible to process more than one service call (at least partially) in one interval. The main effect of this strategy is the *reduction of costs*. If service calls can be processed during an already paid billing interval, then they do not cause any additional costs. Even if a service call can only partially be processed in an already paid billing interval, in the sum the cost of one billing interval may be saved. Besides cost, this strategy can also save time. For subsequent service calls the service does not need to be provisioned, in this case the provisioning time can be saved. Regarding cost and time, this strategy poses no risk as services are de-provisioned before they cause any additional cost.

The integration of the cost model strategy into our existing ODP system is shown in Fig. 3. The existing components (i.e.

everything outside the *optimization extension* box) stay unchanged. For this and all other strategies, the optimization manager is the core component of the corresponding optimization extension and realizes the optimization logic. Each optimization extension can contain additional components; in case of the cost model strategy these are a *cloud cost model knowledgebase* and a *stopwatch manager*. The cloud cost model knowledgebase provides consolidated data about the cost models of cloud providers; it can be realized by integrating existing solutions [7]. The *stopwatch manager* component enables the timed execution of arbitrary operations.

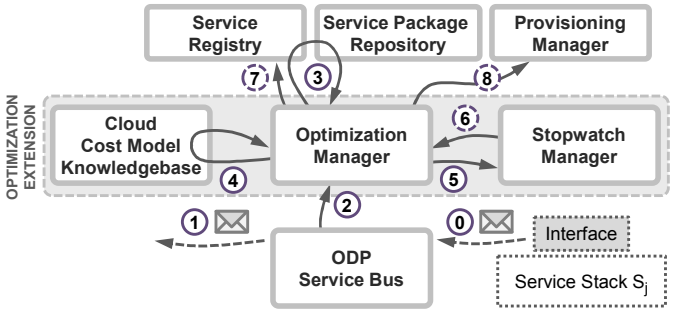

Fig. 3. Integration of cost model exploitation strategy

The optimization logic is triggered by the reception of a service response (Fig. 3, step 0). The response is forwarded to the service consumer (step 1) and then the de-provisioning of the service is requested (step 2). Here the optimization logic hooks in. The optimization manager retrieves data from the service registry, the service package repository (step 3) as well as the cloud cost model knowledgebase (step 4), and then calculates how long the service can stay provisioned without causing any additional cost. After that, it accesses the stopwatch manager and starts a corresponding countdown (step 5). If the countdown ends and the service has not been called, it is removed from the service registry (steps 6, 7) and then de-provisioned (step 8). If the service is however called while the countdown is still running, the countdown is cancelled.

B. Keep-Alive Strategy

If a service is needed multiple times in a row, e.g. when it is called from inside a loop or when multiple instances of the same process model are executed in parallel, it may be useful not to de-provision it but to keep it running. The core idea of the keep-alive strategy is to estimate the chances that a service is used again (in the near future) after it has processed a service call and is free. Depending on this chance of reuse, the service is either de-provisioned or kept running. There are multiple approaches for determining the reuse probability, which results in different variants of the keep-alive strategy. In the following, we will describe the common characteristics for all these variants and then introduce four keep-alive strategies.

In these strategies, if chances are high that a service is needed again, it is not de-provisioned. The goal of all keep-alive strategies is the saving of time. Whenever an idle service receives another service call, this call is processed without any delay. The reduction of cost is also possible if subsequent service calls are (partially) processed in the same billing interval as the previous service call. Concerning time, the strategy does not contain any risk. Additional cost (compared to the default ODP strategy) may be caused when the duration between two service calls is so long that additional billing intervals apply or no subsequent service calls arrive at all. In the following, we will present four different variants of the keep-alive strategy that differ in how the decision about de-provisioning a service or not is taken.

1) Keep-Alive Based on Auditing Data

If auditing data is available, these data can be used to estimate if it is useful to keep a service provisioned or not. If a service has been called repeatedly in the past, then we assume that this may also happen in the present. Whenever a response arrives from a service, a query on the auditing data is performed. If the probability P that this service will receive another service call in a given duration D is greater than a given threshold T, the service is not de-provisioned. If the service is however not called in a duration of D + B (buffer), then the service is de-provisioned.

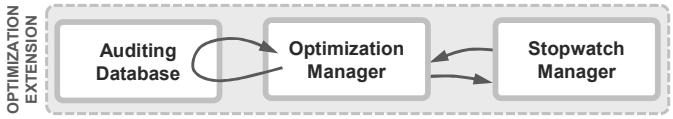

Fig. 4. Integration of keep-alive based on auditing data strategy

For this strategy, auditing data has to be available in sufficient amount and it has to be suitable to derive conclusions from it that are meaningful in the current context. The effective application of this strategy requires an appropriate selection of the parameters D, T, and B, which highly depends on specific use case scenarios and domain knowledge. The optimization extension for this strategy comprises the *optimization manager*, an *auditing database* and a stopwatch manager (Fig. 4).

2) Keep-Alive Based on CCPM Structure

If knowledge about the structure of a process model is available to the ODP middleware, this can be used for optimization. Depending on the structure of a process model, it can be determined, if a service that has just finished it's processing will be used again in this process. A control flow analysis determines if the successor activities of the current activity will call the same service again.

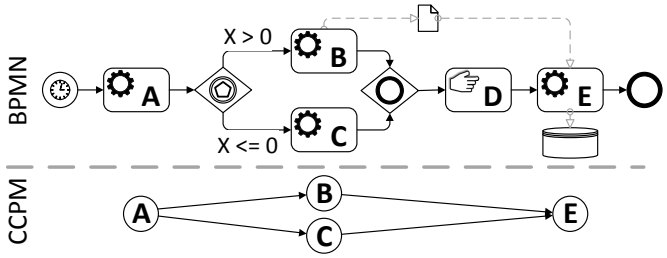

Fig. 5. Mapping example from BPMN to CCPM

Our optimization is encapsulated in the ODP middleware, which is independent of any specific workflow language or engine. Consequently, the optimization based on process models also has to be independent of any specific workflow language. To enable this, we introduce the concept of *communication centric process models* (CCPM). A CCPM is a directed graph where nodes represent communication activities (service calls) and edges represent the control flow. A CCPM is a simplified view on process models of specific workflow languages. It reduces a process model to the control flow

structure between all activities that interact with the ODP middleware. An exemplary mapping of a BPMN process model to the corresponding CCPM is shown in Fig. 5. All communication activities (A,B,C,E) are preserved. Activity D represents a manual task that will be executed independent of the ODP middleware and is therefore omitted. Data flow elements and conditions are omitted as they are not visible to the ODP middleware at runtime.

The CCPM-based control flow analysis starts at one of the activities of the CCPM and then analyzes the preceding control flow. If all possible paths of a given length D contain at least one activity that calls the same service as the current activity, then this service is guaranteed to be called again (within the given analysis distance D). If only some of the analyzed paths contain a call to the service, no qualified statement can be made, if the service should be de-provisioned or not. For this strategy, a CCPM has to be available for each process. Besides the optimization manager, the architecture (see Fig. 6) contains a CCPM repository and a CCPM analysis component responsible for the control flow analysis.

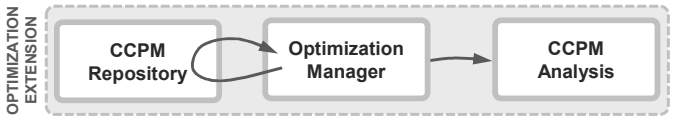

Fig. 6. Integration of keep-alive based on CCPM structure strategy

The distance D used in the control flow analysis describes the length of a path in a CCPM, but it does not allow any statements about the time needed to execute this path. Even if the result of the control flow analysis is, that a service is guaranteed to be called again, the duration until then can be arbitrary long. Therefore, it is reasonable to introduce a timeout for this strategy. If a service is, based on the control flow analysis, not de-provisioned but then not called during the given timeout, the service will be de-provisioned after the timeout (although we know, that this service will be needed at some time later).

3) Keep-Alive Based on CCPM with Metadata

The CCPM control flow analysis introduced so far considers exclusively the structure of a CCPM but no execution probabilities (for loops or forks) or estimated execution times (for processing activities). However, if these data are available, the control flow analysis can be refined in a way that it computes the probability, that a service will be called again (instead of only giving a yes/no answer). This probability can then be compared with a user defined threshold T, and if it is lower than T, the service will be de-provisioned, otherwise it is kept running. In addition, the distance D needed for the analysis can be expressed as the estimated time until a service is needed again instead of fixed path length. This strategy therefore requires data about execution probabilities and estimated execution times. The effective application of this strategy requires an appropriate selection of the parameters duration (D) and threshold (T).

4) Keep-Alive Based on Annotated CCPM

In this strategy, the user controls the de-provisioning of services using the CCPM. For each activity in the CCPM the user annotates if the corresponding service is to be de-provisioned after the activity is finished or if it is kept running. The user is responsible for providing meaningful annotations suitable for a specific process model and use case scenario.

IV. Qualitative Decision Support

The effects of all optimization strategies regarding cost and time are summarized in Fig. 7. All strategies have the potential to reduce the execution time of a process. At the same time, the process execution time will never be worse than default ODP. The reason for this is that all strategies follow a common behavior. Whenever a service has processed a service call, they decide (based on different criteria) if they de-provision the service or if they keep it running (and for how long). If the service is kept running and then receives another service call, the time for answering this service call does not include any provisioning time and therefore reduces the overall execution time of the process. The worst case (regarding time) for all optimization strategies is that the provisioning time can never be eliminated. As this case equals the default ODP behavior, no optimization strategy gets worse than this.

	cost	time
cost model exploitation	+	+
keep-alive based on auditing data	+ / □	+
keep-alive based on CCPM structure	+ / □	+
keep-alive based on CCPM with metadata	+ / □	+
keep-alive based on annotated CCPM	+ / □	+

+ / □ = can be better or worse than default ODP
\+ = potentially better than default ODP, but not worse

Fig. 7. Optimization strategies, impact on cost and time

Regarding cost, the effect of the most optimization strategies is ambiguous. Besides the cost model strategy, all strategies can have a positive as well as a negative impact on costs. The main reason is that these strategies do not include any knowledge about the cloud cost model in their optimization at all. Which impact on cost a strategy has always depends on the specific process model executed and the specific parametrization of the strategy.

A comparison of all five optimization strategies is shown in Fig. 8. The graphs characterize each strategy with respect to four dimensions, classifying each as low, middle or high. The upper axis represents the *parametrization complexity (PC)*, i.e. how many parameters have to be set for a strategy, and how

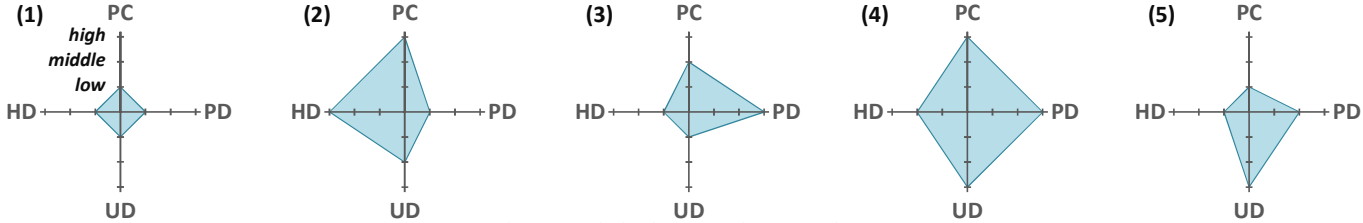

Fig. 8. Optimization strategies comparison

hard it is to determine a suitable parameter set. The right axis shows the *process model dependency (PD)*, i.e. how much the strategy depends on the process model. The lower axis represents the *user input dependency (UD)*, i.e. how much a strategy depends on user input. This dimension has some overlap with the PC dimension, as parameters are also user defined, but covers also additional user input. The left axis shows the *history dependency (HD)*, i.e. the degree of dependency on historical data. These dimensions can altogether be interpreted as an indicator for the complexity of the realization and application of each optimization strategy.

The *cost model strategy (1)* scores low for all dimensions as it solely depends on the knowledge about a cost model. As we will demonstrate in the following section, this strategy is nevertheless capable to yield competitive reduction in both, cost as well as time. The *keep-alive based on auditing data strategy (2)* mainly depends on historical data as well as an appropriate parametrization of the corresponding analysis. The *keep-alive based on CCPM structure strategy (3)* also requires some parametrization but depends even more on the process model, as it decides the de-provisioning of services solely based on structural process model analysis. The *keep-alive based on CCPM with metadata strategy (4)* scores high for nearly all dimensions. It requires nearly the same parametrization as (2), it analyses the process model structure similar as (3) and in addition depends on the annotated metadata given by the user. As this metadata typically originates from some kind of historical data, it also scores in this dimension. The *keep-alive based on annotated CCPM strategy (5)* in contrast pushes the decision about service de-provisioning to the user. The process model is in this strategy only used as a means to capture the user input.

V. Evaluation

In order to evaluate the optimization strategies, we implemented and applied them to a real world use case from the eScience domain. The OPAL simulation workflow, a Kinetic Monte Carlo Simulation for the simulation of solid bodies [8], is depicted in Fig. 9 as BPMN diagram. In addition, also the mean execution times for all activities are shown.

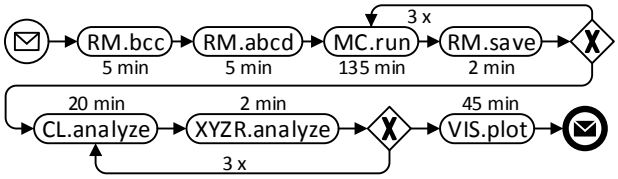

Fig. 9. OPAL simulation workflow, based on [8]

For our evaluation, we run in total more than 170 instances of the OPAL process. In order to reduce the required time to a manageable level we decided to replace the real ODP system with a simulator for the ODP middleware. The simulator supports the transformation between real time and simulation time, i.e. the simulation can run in shorter time than the duration that is simulated. This helps to speed up the evaluation allowing to simulate a broad range of optimization strategies and use case scenarios. The simulator application implements the same request processing logic as the real system, but instead of calling and provisioning "real" services, these steps are simulated by just waiting for a specific time. These times have been derived from our previous experience in running OPAL as well as our ODP system [8][9][10]. The implemented optimization strategies can easily be integrated into the real ODP middleware as all interfaces they offer or require are the same in both, the simulator and the real ODP system.

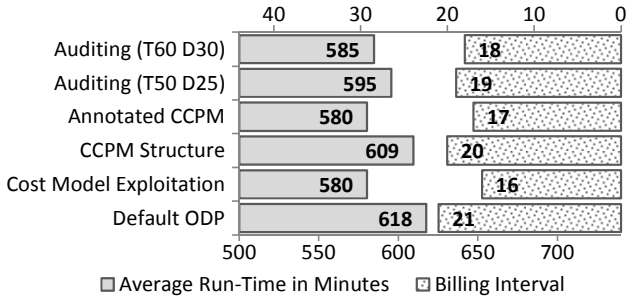

Fig. 10. Evaluation results (1)

In the evaluation, all instances of the OPAL process are started using SoapUI (www.soapui.org). The OPAL process is realized as BPEL process and executed on an Apache ODE workflow engine (ode.apache.org) running on a Tomcat server. The ODP simulator provides a web service interface that receives all service calls from the OPAL process. The simulator contains all components of the ODP middleware as introduced in section II and captures additional data for analysis (e.g. cost data). The simulator also provides a graphical user interface allowing live insight into all of its components as well as exporting the captured analysis data. The simulator application, the BPEL process, as well as all data gathered during the evaluation are available online at www.odp-approach.com.

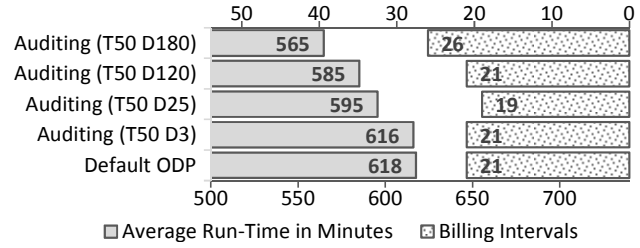

Fig. 11. Evaluation results (2)

All presented evaluation results are mean values of at least ten runs of the OPAL process. An overview is given in Fig. 10 showing the average run-time and cost for the OPAL process and different optimization strategies. The best (i.e. shortest) run-time is achieved by two strategies, the *cost model* strategy, and the *annotated CCPM* strategy. Both strategies reduce the run-time by 38 minutes (6.15%). The *CCPM structure* strategy shows only little improvement compared to default ODP. We set the look-ahead distance to a path length of two, in this case the CCPM analysis identifies only the first service (RM) as to be used again. For all other services, it is not guaranteed that they will be used again, which is mainly caused by the loops (i.e. the loop may be entered again or it may be left). Regarding costs all strategies perform better (i.e. cheaper) than the default ODP behavior, with the *cost model* strategy achieving the highest saving of five billing intervals (23.8%).

For the *auditing* strategy, Fig. 10 contains two different results; both have been achieved with different parameters (threshold T, look ahead distance D). Additional results for the auditing strategy based on more and different parameter settings are shown in Fig. 11. The threshold is fixed to 50% but the look ahead distance (in minutes) varies. With increasing look-ahead distance the average process run time decreases. Considering a greater duration for analysis increases the probability of identifying another call to the same service. In contrast, the costs show a quite different behavior. Considering the look ahead increase from 3 to 25 minutes the cost first decreases, but with further increased look ahead the cost increases again and in the end shows a much higher value than any other strategy. This demonstrates that a suitable parametrization is a non-trivial task that highly depends on the specific use case scenario. In addition, multiple parameters may influence each other and in general require deciding for a suitable tradeoff between cost and time.

VI. Quantitatve Desicison Support

Most of the presented optimization strategies require parametrization and their performance highly depends on these parameters. In addition, the effect of the optimization strategies with a given set of parameters varies depending on the usage context, e.g. the workflow that is executed. Considering time, an inappropriate parametrization can yield in no improvement at all, but considering cost, it may even lead to deterioration. This raises the need for a decision support mechanism that supports the user in selecting appropriate parameters for specific use case scenarios. The ODP simulator introduced in the previous section targets this challenge. It allows users to execute their specific use cases, i.e. workflows, in much shorter time and without any cloud-related costs. The data captured by the simulator can be analyzed to evaluate the effect of a certain parametrization on execution time as well as cost. It allows the easy execution of parameter studies, similar to what we presented in Fig. 11, in order to find suitable parameter sets.

VII. Related Work

Resource provisioning and its optimization is an established and well-studied topic in the context of cloud computing. An algorithm for the cost-optimal allocation of cloud resources from the *point of view of a cloud consumer* is presented in [11]. It is able to handle uncertainty in demand and price and covers on-demand resources as well as reservation plans. Resource allocation is however mostly considered from the *point of view of the cloud provider*. In [12] an approach for the QoS-aware dynamic resource allocation for VMs is presented. Based on workload and QoS metrics, CPU and RAM resources are dynamically (re-)allocated between multiple VMs. The work presented in [13] targets simulation workflows that run on grid environments but that are in parts also executed on cloud resources. This process comprises several decisions to be made, e.g. when to move from grid to cloud resources or when to move back and release cloud resources. All these decisions are made with the goal of optimizing the ratio between cost and the timesaving achieved by moving workload into the cloud. The decision about the point in time when cloud resources are released corresponds to the cost model strategy presented in our paper. The evaluation of [13] is, similarly as in our work, performed based on a simulator application.

VIII. Summary

The work presented in this paper continues our previous work on the on-demand provisioning of workflow middleware and services (ODP). It tackles situations when the default ODP behavior shows a not optimal behavior, e.g. regarding repeatedly used services. We presented an architecture for the seamless and loosely coupled integration of optimization strategies in ODP and introduced five different optimization strategies. We discussed the strategies providing qualitative decision support for the selection of an appropriate strategy and proposed an ODP simulator application as a means for quantitative decision support. The general optimization approach as well as all five strategies have been evaluated based on a real world use case from the eScience domain.

Acknowledgment

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC310/1) at the University of Stuttgart.

References

[1] Yang, X., Wang, L., & Jie, W. (Eds.). (2011). Guide to e-Science: next generation scientific research and discovery. Springer.

[2] I. J. Taylor, et al. (Eds.). Workflows for e-Science: Scientific Workflows for Grids. Springer-Verlag New York, 2006.

[3] Vukojevic-Haupt, K., Karastoyanova, D., & Leymann, F. (2013). On-demand provisioning of infrastructure, middleware and services for simulation workflows. In SOCA 2013. IEEE.

[4] Vukojevic-Haupt, K., Haupt, F., Karastoyanova, D., & Leymann, F. (2014). Service Selection for On-demand Provisioned Services. In EDOC 2014. IEEE.

[5] Gomez Saez, S., Andrikopoulos, V., Leymann, F., & Strauch, S. (2014). Towards Dynamic Application Distribution Support for Performance Optimization in the Cloud. In CLOUD 2014. IEEE.

[6] Schikuta, E., Wanek, H., & Ul Haq, I. (2008). Grid workflow optimization regarding dynamically changing resources and conditions. In: Concurrency and Computation: Practice and Experience, 20(15).

[7] Andrikopoulos, V.; Song, Z. & Leymann, F. (2013) Supporting the Migration of Applications to the Cloud through a Decision Support System. In: CLOUD 2013. IEEE.

[8] Sonntag, M., Hotta, S., Karastoyanova, D., Molnar, D., & Schmauder, S. (2011). Using services and service compositions to enable the distributed execution of legacy simulation applications. In Towards a Service-Based Internet (pp. 242-253). Springer Berlin Heidelberg.

[9] Haupt, F., Fischer, M., Karastoyanova, D., Leymann, F., & Vukojevic-Haupt, K. (2014). Service composition for REST. In EDOC 2014. IEEE.

[10] Vukojevic-Haupt, K.; Haupt, F.; Leymann, F. & Reinfurt, L. (2015) Bootstrapping Complex Workflow Middleware Systems into the Cloud. In: e-Science 2015. IEEE.

[11] Chaisiri, S., Lee, B. S., & Niyato, D. (2012). Optimization of resource provisioning cost in cloud computing. In: IEEE TSC, 5(2).

[12] Dawoud, W., Takouna, I., & Meinel, C. (2011). Elastic VM for cloud resources provisioning optimization. In ACC 2011. Springer.

[13] Ostermann, S., Prodan, R., & Fahringer, T. (2010). Dynamic cloud provisioning for scientific grid workflows. In GRID 2010. IEEE.

[14] Papazoglou, M.P.:Service-oriented computing: concepts, characteristics and directions. In: Proceedings of WISE 2003

All links were last followed on 14.09.2015.