Experience reuse in a workflow-oriented cloud management framework

Eric Kübler and Mirjam Minor

Goethe University, Business Information Systems, Robert-Mayer-Str. 10,
60629 Frankfurt, Germany
ekuebler@cs.uni-frankfurt.de,
minor@cs.uni-frankfurt.de

Abstract. Using cloud resources for the execution of workflows is a recent approach. It provides new business concepts for selling the execution of workflows in the cloud. However, there is a lack of concepts for flexible integration of workflow management tools and clouds for resource usage optimization. While traditional methods such as running a workflow management tool monolithically on cloud resources lead to over- and under-provisioning problems, other concepts include a very deep integration, where the options for changing the involved workflow management tools and clouds are very limited. In this work, we present the architecture of *WFCF*, a connector-based integration framework for workflow management tools and clouds to optimize the resource utilization of cloud resources for workflow by Case-Based Reasoning. Experience reuse contributes to an optimized resource provisioning based on solutions for past resource provisioning problems. The approach is illustrated by a real sample workflow from the music mastering domain.

1 INTRODUCTION

Today, cloud computing receives significant attention and the variance of offered services is increasing. Novel business concepts emerge. One of these concepts is *workflow as a Service (WFaaS)* as introduced by [20, 12]. The Workflow Management Coalition [21] defines a *workflow* as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". A *task*, also called activity, is defined as "a description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resources(s) to support process execution" [21]. The idea of WFaaS is to execute activities within a cloud. A cloud vendor [5] is a company that offers services in the cloud, for example the execution of a workflow. However, the vendor is not always a cloud provider. Even if renting the required cloud resources by a third party provider, the vendor is responsible for maintaining the service level agreements (SLA) for the own costumers. An SLA defines agreements between the provider and the customer

about different aspects of the quality of service. For example, an SLA can be specified for the execution time of the workflow. To prevent an SLA violation, the vendor may rent more resources than required (over-provisioning) but this will reduce the profit. On the other hand, if the vendor rents less resources than required (under-provisioning) this can lead to violations of the SLA. Violations of an SLA create high costs and a loss of reputation [19]. Thus, the optimal management of resources is an important aspect for cloud computing [9] in general and, particularly, for WFaaS vendors. It is challenging to find a good balance between over- and under-provisioning of resources [6]. A straight-forward solution to provide resources is the static way. This means, the system does not adjust itself to a changing situation. Obviously, this will lead to under- or over-provisioning [19]. A more dynamic approach is preferable. Existing approaches range from rather simple, rule-based solutions, such as observing the number of open connections to a cloud resource [17] to sophisticated, algorithmic solutions [18].
Knowledge and experience management methods [10] provide an alternative solution approach focusing on the reuse of experience. In this paper, we investigate *Case-based reasoning (CBR)* as a method for optimizing the provisioning of cloud resources by experience reuse. The idea of CBR is that similar problems have similar solutions [4]. To retrieve similar problems (cases), a similarity function determines the similarity between two cases. CBR has two unique benefits. Due to the fact that CBR only requires the similarity function to receive other, similar problems and their similar solution, the time and computational effort should be relatively low. We will introduce the architecture of *WFCF* (Workflow Cloud Framework) a connector-based integration framework for workflow management tools and clouds that aims to optimize the resource utilization of cloud resources for workflows by means of CBR. WFCF follows a shallow integration approach, i.e. it is independent of the chosen workflow management tools and cloud systems. The idea is to have a set of WFCF components that are independent of the workflow management tools and cloud systems. A tool-specific set of connectors interacts with the actually used tools and system. The benefits of experience reuse is twofold namely the reduction of costs for the vendor by reducing over-provisioning and SLA violations and, second, a better cost estimation based on experience.

The remainder of the paper is organized as follows: Related work is discussed in Section 2. The WFCF architecture is presented in Section 3. A running sample from the music mastering domain illustrates the plausibility of the approach in Section 4. Finally, a conclusion is drawn in Section 5.

2 Related work

The idea of using CBR for cloud management is not new. The work of Maurer et al. [16] applies CBR to implement automatic cloud management. A *case* in cloud management records a cloud configuration, that means the set of started VM's,

containers, and workflows, with current services and SLA's to be processed as a problem situation. A solution describes the optimal distribution of work on the optimal number and configuration of cloud resources while maintaining SLA's. Maurer et al. use a bag of workloads to schedule the work, which makes it difficult to predict future workloads and system behavior.
Another aspect of WFaaS is the integration of workflow management tools with clouds. In this case, integration means an exchange of information between the workflow management tool and the cloud for an optimized resource usage. Without any integration, it is difficult for any management approach to determine the required resources. This easily leads to over- or under-provisioning. In their work, Bala and Chana [8] present a survey of workflow scheduling algorithms for cloud computing. However, most of the approaches have not yet been implemented, three notable exceptions are [20, 12, 15]. They deeply integrate workflow and cloud technology, i.e., they strongly depend on the used cloud and workflow managent tools. Therefore, they are very limited in their options to exchange either the used cloud or workflow management tool or both. A solution for this problem should be a shallow integration for flexible integration of different workflow management tools and clouds while not being restricted to them, because of flexible connectors and an abstract representation of the used tools.

3 WFCF ARCHITECTURE

In this section, we will explain the architecture of WFCF and its components. Starting with the overall architecture, we show the details of the monitoring and management components and how they interact.

3.1 Overall architecture

Figure 1 shows the overall architecture of the WFCF, which we will explain in the following. The architecture can be divided roughly in three parts: the environment, the monitoring component and the management component. The environment are the cloud and the workflow management tool that is used by the customer. Ideally, WFCF will use the already offered information and management methods of the tools, so that additional changes are not necessary. Therefore, WFCF will use offered log files, databases and API's for monitoring the environment and to configure the cloud. *CWorkload* is the monitoring component. It collects information from the environment and combines data across the different layers (the cloud layer and the workflow layer) to one status model of the system. We had done initial tests for the cross layer monitoring aspect of CWorkload in [13]. The management component recognizes current or upcoming problems within the system. This could be for example violated SLA's, violated constraints or resource over-provisioning. If a problem occurs, the management component searches for a solution and reconfigures the cloud. We will explain this in more detail in section 3.3.

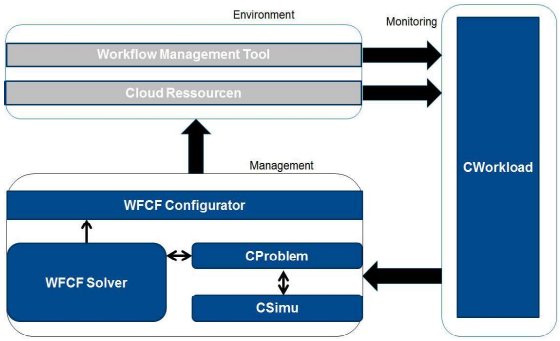

Fig. 1. Architecture of WFCF

3.2 Monitoring

The main components of WFCF work independent from the actually used environment. To work properly, WFCF needs different information about the status of the actually running workflow instances and the resource utilization of the cloud. Figure 2 shows in more detail the monitoring of WFCF.

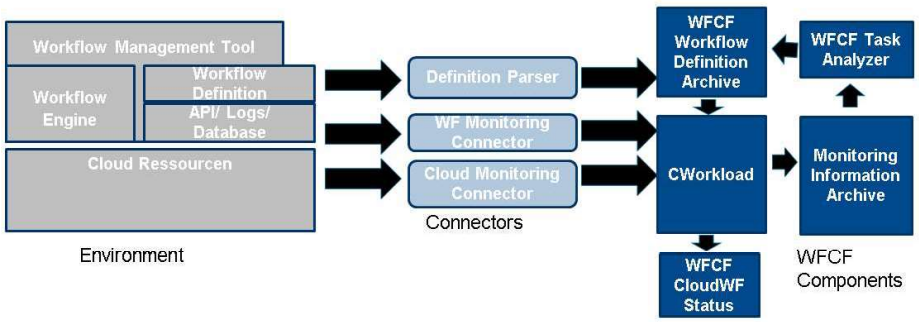

Fig. 2. Monitoring of WFCF

There are three connectors between the environment and WFCF. A workflow definition is very similar to a class in programming as that a workflow definition is the schema of a workflow where an workflow instance is an object of a workflow definition. A workflow definition contains all information about the structure of the workflow. For example, the name of the tasks and their order. There are several formats to define a workflow definition. These could be, for example, BPMN or acyclic directed graphs. The *Definition Parser* parses the workflow definition, transforms it into a standardized format and stores it in the *WFCF Workflow*

Definition Archive. The archive will also contain the service characterizations we introduced in [13]. In short, the service characterizations provide a hint as to how a web service (or the tasks which call the web service) will utilize the cloud resources. A characterization could be, for example, *long running*, which means the service will be executed for more than 30 minutes. Another example could be *compute intensive*, which means that the service has a high demand for CPU cycles. In addition, the archive also contains information about local SLA and other constraints, for example, which task requires which type of web service.
The *WF Monitoring Connector* gathers the information about the current workflow instances. This information could be from log files, databases or directly from the workflow engine via API. The information contains the name and start-time of the executed workflow and the start-, end-time and name of individual tasks, as well as the URL or IP of the called web service. This information should be offered in one form or another by all commercial workflow management tools and most of the open source tools. Because of the great variety of workflow management solutions, it is necessary to implement the WF Monitoring Connector and the other two individually for the used management tool. However, the workflow management tool itself has not to be changed when any kind of logging is enabled. So even when the connector has to be reimplemented, the company has not to change their already running systems.
The *Cloud Monitoring Connector* is the interface between WFCF and the used cloud. This connector monitors the resource utilization. For example, the CPU and memory usage. Similar to the WF Monitoring Connector, this connector can use log files, API's or databases for monitoring and has to be implemented individually for each different cloud.
CWorkload is the core of the monitoring component. It has two tasks. First, it builds the monitoring model. This model is the *WFCF CloudWF Status* and combines all information about the status of the cloud, the currently running workflow instances and the information about the workflow definitions of these instances. It also contains all information about local SLA and constraints. The management component of WFCF will use the WFCF CloudWF Status to identify current or upcoming problems. The second job of CWorkload is to maintain the *Monitoring Information Archive*. This archive stores information about the duration, run time behavior and resource-usage of the executed tasks. The *WFCF Task Analyzer* analyzes this information and updates the service characterizations of tasks in the WFCF Workflow Definition Archive. For example, if a task has been executed several times and each time its execution time was over 30 minutes, WFCF Task Analyzer will annotate this in the WFCF Workflow Definition Archive as long running.

3.3 Management

Whereas the monitoring component observes the environment, the management component configures it. This means, the management component starts and stops virtual machines or PaaS container, scales resources and migrates content. Figure 3shows the management component in more detail.

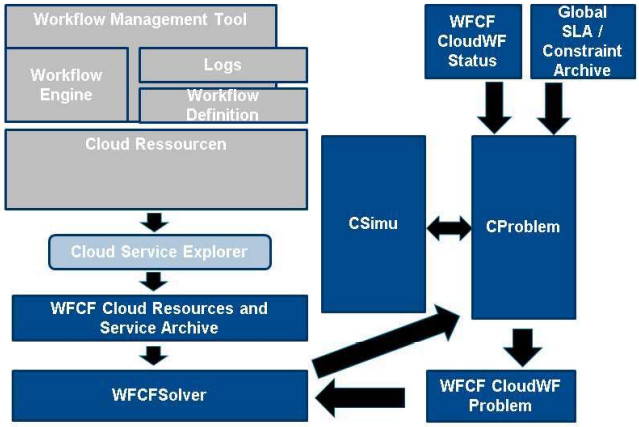

Fig. 3. Management of WFCF

After CWorkload has build the WFCF CloudWF Status, *CProblem* is the part of WFCF which interprets the current status of the environment that is recorded as the WFCF CloudWF Status. Besides the CloudWF status, there is another archive, the *Global SLA // Constraint Archive*, where global constraints and SLA's are stored. Other than the WFCF Workflow Definition Archive that only contains local constraints and SLA's for individual workflows, the *Global SLA // Constraint Archive* contains SLA's and constraints that are valid for all workflows of a user. There are several different problems that can occur and which CProblem will identify, e.g., violated SLA's. We are planning that CProblem does not only check the current situation, but also do a forecast to identify upcoming problems and over-provisioning. Through the workflow definitions, for example, CProblem can recognize if a certain web service is going to be used in the future by a currently running workflow instance. If not, WFCF can shut down the VM or container to save money. Another possible scenario could be that currently, there is no violated SLA, but in the near future, several tasks with high resource demand will be started, which can probably lead to a SLA violation, so WFCF should scale up the resources to avoid this problem. Forecasting SLA violations, however, could be a difficult task. To decide if the start of some resource intensive tasks lead to a SLA violation is not as easy as to recognize if a web service has not started yet. A simulations seems a proper way to identify these kind of problems. Therefore, CProblem interacts with *CSimu*. We are planning to use CloudSim [1] as the core of our simulation part. CSimu will simulate the execution of the tasks with the current cloud status and will show if this will lead to a SLA violation. If any problem is unidentified, CProblem extends the CloudWF status with annotations about the problems. This new annotated model is the *WFCF CloudWF Problem*. Such annotations could be, for example, *web service x is not longer needed* or *SLA y is currently violated*.

Whereas CWorkload is the core of the monitoring component, the *WFCF-Solver* is the core of the management. Similar to CWorkload, the solver has two jobs. First, the solver searches for a new cloud configuration that solves the current problems. Then it finds a reconfiguration path from the current cloud configuration to the new solution. In the last step, the solver sends the reconfiguration steps to the *WFCF Configurator* as shown in Figure 1. The reconfigurator then will do the reconfiguration job. There are several possible approaches to find a new cloud configuration. We will choose *Case-Based Reasoning* (CBR) as our solving strategy.

3.4 CBR for problem solving

In this section we take a closer look how the *WFCFSolver* will solve the cloud management problems with CBR methods. As mentioned in Section 1, the idea of CBR is that similar problems have similar solutions. If a problem situation occurs the system retrieves experience by searching a similar situation from the past. In our case a problem situation is a cloud configuration with a problem, such as violated SLA's. This is the retrieval step. The key to experience retrieval is a good notion when some kind of experience is relevant for a certain situation. This knowledge is captured in the similarity measure [10]. The reuse step of CBR is to use the solutions from the past for the current problem. In our case, the solution contains re-configuration steps. This for example could be the to start new VM's or to migrate containers to another VM.
A problem situation is recorded as *WFCF CloudWF Problem*. Figure 4 shows an example of a simple CloudWF Problem. This example contains one VM, two containers for the required web services and a bunch of workflow instances currently being executed. The image depicts not the entire workflows but the tasks that are currently active within the instances. Most of the workflow instances are derived from the same workflow definition and are in the same state of execution. At this point, the task *Task 1* uses the web service *web service 1* while *Task 2* uses *web service 2*. In addition, there is another workflow instance (in the bottom right corner). This instance is probably from a different workflow definition, or the instance is in a different state of execution. The current task of this instance is *task 217* and for its proper execution, a web service that has not yet started is required. This example also includes the constraint that the average resource utilization must not extend 75% for reasons of performance. The example CloudWF Problem includes also three problems. The resource utilization of the CPU and memory of VM1 is too high and a new web service must start for Task 217. More complex CloudWF Problems may involve several VM's, containers and workflow instances.

A *case base* is an archive of previous problems and their solutions. The case base is not depicted in Figure 3, because it is part of the solving strategy and not part of WFCF itself. The solver will search the case base for similar problems in the past. In our previous work [14], we have introduced the idea of a similarity function for cloud configurations. For the similarity of a cloud configuration, we consider the following aspects as important.

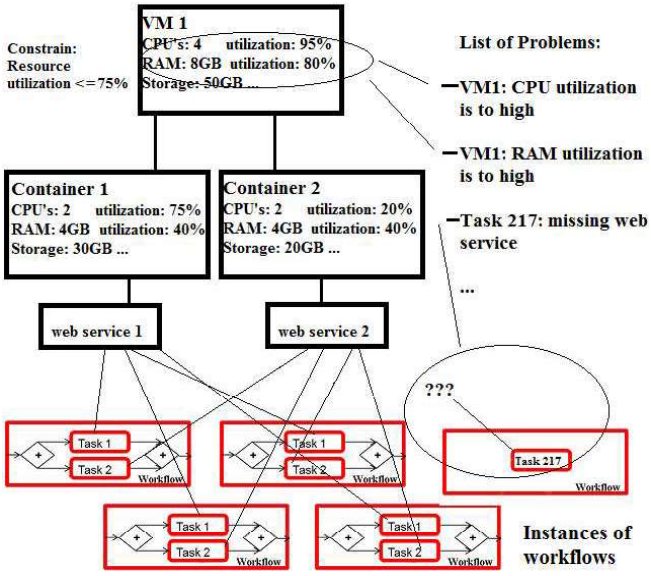

Fig. 4. Example representation of a case

The provided resources. Two VM's are similar, if they have a similar set of resources available. For example, two VM's with a quad core processor should be more similar than a VM with a dual core processor and a VM with a quad core. The idea is, that VM's with a similar set of resources should handle general workload similar, where VM's with a different set of resources maybe lead to other results, for example you can not migrate a container that requires a quad core, if the VM only have a dual core. The same applies to containers.
The resource utilization. VM's with a similar resource utilization, for example average CPU usage, should be considered as similar. If the utilization differs significantly, a solution that is valid for one case could be invalid for the recent case. For example if the disk space utilization for a VM $vm_1$is 20% and for an another vm_2 100%, the system can not migrate a container to vm_2, because of the lack of free disk space, while a migration to vm_1 is feasible. The same applies to containers.
The assigned SLA's and whether they are violated or not. If two cloud configurations have a similar set of SLA's, the configurations should be considered as similar. Different SLA's or the violation of different SLA's can lead to a situation, where a problem of the one case is not a problem in another case. For example if a cloud configuration includes an SLA on availability and the other doesn't, the availability can be a problem in the first case while it is not in the second. That leads to the situation, that a solution that mends the availability problem for one case is not applicable for the other case.

The executed workflow instances and their workflow definitions. The number of the started instances and the structure of the workflow definitions can have a high impact on the requirements for resources and for started web services. For example, if an instance of a workflow is started that requires a certain web service, every solution that does not include this web service is not valid. The structure of the workflow definitions also specifies which tasks will be started next.

To determine the similarity of two cases, we use a composite, distance-based similarity function based on the aspects introduced before. The similarity of each aspect in two cases is computed by a particular local similarity function. The local similarity values are aggregated by means of a sum of weighted aspects. For example, the similarity function of the resources provided for a VM is based on a taxonomy, and analog for containers. For the size of the provided resources, we have been inspired by Amazon EC2 instances [7] for nodes and OpenShift [3] for containers. Figure 5 shows our taxonomy for Amazon EC2 VMs. For other aspects, we use mainly standard distance functions. For example to determine the distance between the resource utilization for VMs vm_uti, we use the Euclidean distance for the resource vectors of CPU, memory, storage, network traffic, and so on. The utilization values are provided in percentage. The distance of the resource utilization vm_util is calculated by the Euclidean distance $vm_{util}(p,q) = \sqrt{\sum_{i=1}^{n}(q_i^{vm} - p_i^{vm})^2}$, where p is the vector of n utilization values for the first case and q for the second case. For example, $q_1^{vm} = 50$ is the utilization of the CPU q_1^{vm} with a value of 50%. p_2 is the utilization of the memory and so on.

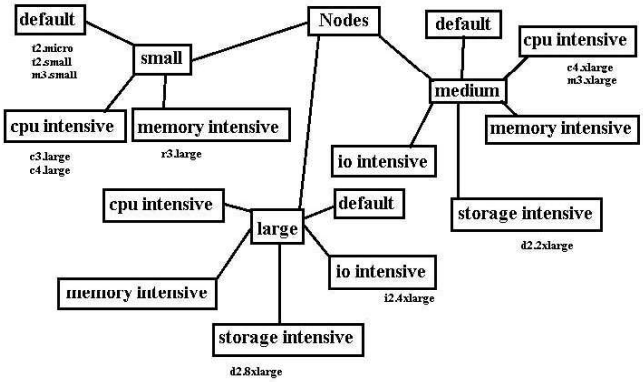

Fig. 5. Snippet of the taxonomy of provided resources for nodes.

The similarity function for the workflow aspect of our approach is ongoing work. We are planning to consider the currently active tasks as well as the tasks that are to be started in the near future. The similarity of two individual tasks is determined by its service characterization and the size of its input data. Two tasks are similar if they have the same characterization (for example CPU intensive) and if the size of the input data are similar. Each workflow instance has 0 to n active tasks. These are the tasks that are currently executed. The set of active tasks is the set of all active tasks from all workflow instances. To determine the similarity of two sets of active tasks, we are planning to implement one of the functions introduced in the literature [10].
In addition, the knowledge of the workflow definitions allows to build another set of tasks that will be active in the near future. Figure 6 shows an example workflow. If Task 1 is the currently active task, we can say for sure, that task 2 will be executed as soon as Task 1 is finished. In some cases, for example after a conditional fork, the next task to executed can be unclear. However, in this case we can make an assumption, based on the empirical knowledge of the workflow, stored in the Monitoring Information Archive, we introduced earlier. This can be done for every active workflow instance to estimate the tasks approaching soon. The result is a set of possible future task. Due to the information stored in the Monitoring Information Archive and the service characteristics, WFCF should be able to identify near future problems and bottlenecks. Thus, we will also consider the similarity of the future tasks in the problem part.

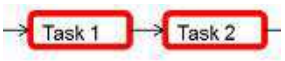

Fig. 6. Sample workflow with two tasks

For the reuse step, a solution is a cloud configuration without problems. The solver will search for a similar problem and use the solution for this old problem or the solution can serve as a starting point for a new solution. Anyways, the solver will send the solution back to CProblem to check if the solution comes up with new problems. CProblem will check and simulate the solution and give feedback to the solver. This will be repeated until a solution is found or another condition is fulfilled. This could be, for example, a time limit. In this case, the solution with the least significant problem will be chosen. The usage of CBR also opens the opportunity for post-mortem analysis and improvement of the stored solution, while WFCF is otherwise idle. In addition to the case base, there is the *WFCF Cloud Resources and Service Archive*. This archive contains information about the available type of containers, VM's, web services and so on. This archive helps the solver to find valid solutions. Similar to the connectors in the monitoring part, the *Cloud Service Explorer* is a connector to the cloud to

discover available sizes and services and store them in the Resources and Service Archive.

4 EXAMPLE

To demonstrate the idea of WFCF, we will give a running example. As our example domain, we chose music workflows to mastering music. The purpose of such a workflow is to transform and process a music file. This includes to normalize and limit the volume of the sound, increase or reduce the sample rate, convert from mono to stereo or reverse and adding special effects like fading and compressing the size of the music file. Figure 7 shows an example workflow. The workflow is modelled in BPMN [11]. To simplify the image, figure 7 does not show the input and output files of the web services. The workflow starts with the *Init Workflow Parameter* tasks to initialize the workflow by a human. The user chooses some parameter for the later mastering. The following two tasks are also human tasks require along with the first one no cloud resources. The

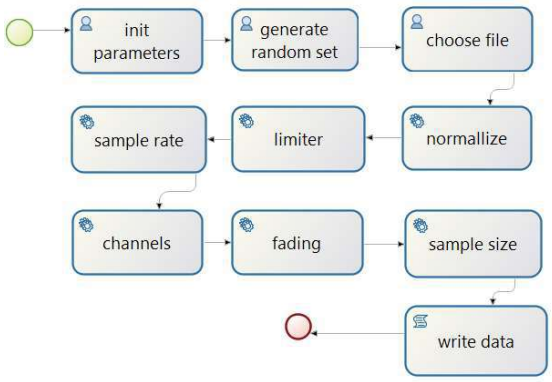

Fig. 7. Sample workflow of mastering music

following tasks are all based on web services and alter the music file each time. For example, the task *normalize* normalizes the volume of the music file, while the task *fading* adds a fade-out effect to the end of the music. Let us assume we are a user who runs jBPM [2] as a workflow management tool and OpenShift [3] (PaaS) and recently created the introduced workflow. Before an instance of this workflow is started, the *Definition Parser* detects a new workflow definition and stores this new definition in the *WFCF Workflow Definition Archive*. The Information will be stored as XML or JSON and will include, besides other information, the following: *Workflow-Definition = "master music", task name = normalize, requires = "normalize web service", service characterization = none.*

This means that the name of the workflow definition is master music and it has (among others) one task with the name normalize. This task requires a normalize web service and has no service characterization. When the user starts an instance of this workflow, the *Workflow Monitoring connector* registers the start and sends a message along with pieces of information to *CWorkload*. The information *CWorkload* receives is that an instance of the master music workflow is started along with the *Init Workflow Parameter* task. CWorkload will store the start-time of the first task in the *Monitoring Information Archive* and will prepare a *WFCF CloudWF Status* for *CProblem*. The WFCF Cloud Status contains the information about the freshly started workflow and the information about the current situation of the OpenShift. Because the user has not executed any Workflow at the moment, no container was started and WFCF includes this information. Because the first three tasks do not require any cloud resources, there is currently no problem. However, CProblem realizes that in the near future, the task *normalize* will start, due to the shape of the workflow. This task requires the web service *normalize web service* that is not available at the moment and this is a problem. *CProblem* prepares the *WFCF CloudWF Problem* and annotates that this web service is required. Because of the simple cloud configuration and because no SLA's are involved, no simulation from *CSimu* is needed. The *WFCFSolver* searches its case base for a case where a web service is required and no container is currently started. Let us assume that the WFCF-Solver finds such a solution and this solution includes to start a container with the needed web service. The solver will send this solution back to *CProblem* to check if the solution includes new problems. This, however, is not the case. The solver can now start to plan the reconfiguration. After the solver is done, the *WFCF Configurator* starts a container with the web service.

5 CONCLUSION

In this paper, we introduced the architecture of *WFCF*, a connector-based integration framework for workflow management tools and clouds. The goal of WFCF is to provide a way to integrate different workflow management tools and clouds, while also optimizing the resource utilization of the used cloud resources. To achieve this goal, WFCF uses multiple concepts. The connector's concept allows in a modular way to integrate workflow tools and clouds by using their usual management and monitoring concepts and without the need for special requirements to the used tools. The monitoring component of WFCF analyzes the run time behavior and resource usage of tasks for a better understanding of their needs and also combines information of the workflow management tool and the cloud to a status model for future analysis and forecast of problems. The management component analyzes this status model for problems by using a combination of simulation and static methods. When a problem occurred or can be forecasted, the management component uses CBR to find a similar problem in the past and solve the problem based on the past solution. WFCF aims at a shallow integration of cloud and workflow management tools for flexible combi-

nation of tools and the optimization of resource usage. We believe that the reuse of experience will lead to the reduction of costs for the vendor by reducing over-provisioning and SLA violations and, second, offer the opportunity for a better cost estimation due to experience, while the approach should not so compute intensive and therefore faster as other solutions. Currently, we are working on a prototypical implementation of the of the architecture to evaluate the concept in future. An open issue is to design the *WFCF CloudWF Status* model in a universal way without dependencies of the actually used tools. Another future task is the acquisition of a larger set of problems that should be recognized and solved.

References

1. The CLOUDS lab: Flagship projects - gridbus and cloudbus (2016), http://www.cloudbus.org, 2016-12-08
2. jBPM (2016), https://www.jbpm.org, 2016-12-08
3. OpenShift (2016), https://www.openshift.com/, 2016-12-08
4. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches 7(1), 39–59 (1994)
5. Antonopoulos, N., Gillam, L.: Cloud computing. Springer (2010)
6. 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 53(4), 50–58 (2010)
7. AWS: Amazon web services (AWS) - cloud computing services (2016), http://aws.amazon.com/, 2016-12-08
8. Bala, A., Chana, I.: A survey of various workflow scheduling algorithms in cloud environment. In: 2nd National Conference on Information and Communication Technology (NCICT). pp. 26–30. sn (2011)
9. Baun, C., Kunze, M., Nimis, J., Tai, S.: Cloud Computing - Web-Based Dynamic IT Services. Springer (2011)
10. Bergmann, R.: Experience management: Foundations, development methodology, and Internet-based applications. Springer Verlag (2002)
11. Chinosi, M., Trombetta, A.: BPMN: An introduction to the standard 34(1), 124–134 (2012)
12. Korambath, P., Wang, J., Kumar, A., Hochstein, L., Schott, B., Graybill, R., Baldea, M., Davis, J.: Deploying kepler workflows as services on a cloud infrastructure for smart manufacturing 29, 2254–2259 (2014)
13. Kübler, E., Minor, M.: Towards cross-layer monitoring of cloud workflows. In: Helfert, M., Ferguson, D., Muoz, V.M. (eds.) CLOSER 2015 - Proceedings of the 5th International Conference on Cloud Computing and Services Science, Lisbon, Portugal, 20-22 May, 2015. pp. 389–396. SciTePress (2015)
14. Kübler, E., Minor, M.: Towards a case-based reasoning approach for cloud provisioning. In: CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science, Rome, Italy 23-25 April, 2016. vol. 2, pp. 290–295. SciTePress (2016)
15. Liu, X., Yuan, D., Zhang, G., Chen, J., Yang, Y.: SwinDeW-c: A peer-to-peer based cloud workflow system. In: Furht, B., Escalante, A. (eds.) Handbook of Cloud Computing, pp. 309–332. Springer US (2010)

16. Maurer, M., Brandic, I., Sakellariou, R.: Adaptive resource configuration for cloud infrastructure management 29(2), 472–487 (2013)
17. Pousty, S., Miller, K.: Getting Started with OpenShift. "O'Reilly Media, Inc." (2014)
18. Quiroz, A., Kim, H., Parashar, M., Gnanasambandam, N., Sharma, N.: Towards autonomic workload provisioning for enterprise grids and clouds. In: Grid Computing, 2009 10th IEEE/ACM International Conference on. pp. 50–57. IEEE (2009)
19. Shoaib, Y., Das, O.: Performance-oriented cloud provisioning: Taxonomy and survey abs/1411.5077 (2014)
20. Wang, J., Korambath, P., Altintas, I., Davis, J., Crawl, D.: Workflow as a service in the cloud: Architecture and scheduling algorithms 29, 546–556 (2014)
21. Workflow Management Coalition: Workflow management coalition glossary & terminology (1999), http://www.wfmc.org/resources 2016-12-15