WFCF - A Workflow Cloud Framework

Eric Kübler and Mirjam Minor
Institute of Informatics, Goethe University, Robert-Mayer-Str.10, Frankfurt am Main, Germany

Keywords: Cloud Management, Case-based Reasoning, Workflow.

Abstract: Using cloud resources for execution of workflows is common nowadays. 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. Case-based reasoning is used to optimize resource provisioning based on solutions for past resource provisioning problems. The approach is illustrated by real sample workflow's from the music mastering domain.

1 INTRODUCTION

Nowadays, cloud computing is more and more popular and the variance of offered services is increasing. One of these services is *workflow as a Service (WFaaS)* as introduced by (Wang et al., 2014; Korambath et al., 2014). The Workflow Management Coalition (Workflow Management Coalition, 1999) 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" (Workflow Management Coalition, 1999).

The idea of WFaaS is to execute automated activities (also called tasks) within a cloud. For the user or provider of workflows, this could be beneficial, because a cloud offers nearly infinite resources and is often cheaper than buying own infrastructure. However, it is still difficult to use the offered cloud resources properly. If the user or the WFaaS provider rents more resources than required (over-provisioning), he has to pay more than necessary. On the other hand, especially for WFaaS provider, if he rents less resources than required (under-provisioning), this can lead to violations of the service level agreement (SLA). A SLA defines agreements between the provider and the customer about different aspects of the quality of service. Violations of a SLA can lead to high costs and loss of reputation for the provider (Shoaib and Das, 2014). Therefore, the management of resources is an important aspect for cloud computing (Baun et al., 2011) in general and also for WFaaS. It is required to find a good balance between over- and under-provisioning of resources (Armbrust et al., 2010). To find such a balance is generally a problem. The simplest method 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 (Shoaib and Das, 2014). A more dynamic approach is required. The range for such approaches is great and spans from rather simple, rule-based approaches such as observations on the number of open connections (Pousty and Miller, 2014) to complex algorithms such as(Quiroz et al., 2009). Another alternative could be *case-based reasoning (CBR)*. The idea of CBR is that similar problems have similar solutions (Aamodt and Plaza, 1994). 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. In ad-

Kübler, E. and Minor, M.
WFCF - A Workflow Cloud Framework.
DOI: 10.5220/0006346305460551
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 518-523
ISBN: 978-989-758-243-1

dition, because of the stored solutions, when WFCF is otherwise idle, it can do a post-mortem analysis of the stored solutions and improve them. The idea of using CBR for cloud management is not new. The work of Maurer et al. (Maurer et al., 2013) applies CBR to implement automatic cloud management. A *case* in cloud management records a cloud configuration 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 (Bala and Chana, 2011) present a survey of workflow scheduling algorithms for cloud computing. However, most of the approaches have not yet been implemented, three notable exceptions are (Wang et al., 2014; Korambath et al., 2014; Liu et al., 2010). 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.

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. This follows a shallow integration approach. The goal of the WFCF framework is to reduce over- and under-provisioning via CBR and to allow the customer or WFaaS provider to integrate workflow management tools and cloud provider as wished. The basic idea of WFCF is to have a set of never-changing components, which work with a standardized sight of workflow management tools and cloud layers and a individual set of connectors to interact with the actually used tools.

2 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.

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 (Kübler and Minor, 2015). 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 2.2.

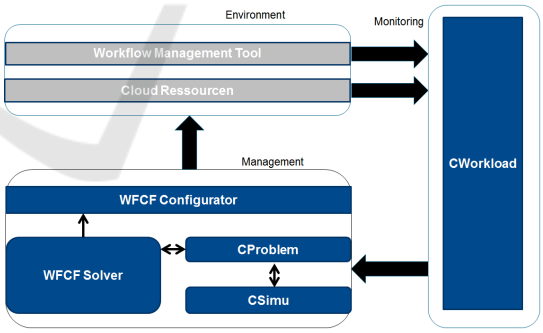

Figure 1: Architecture of WFCF.

2.1 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.

There are three concentrators between the environment and WFCF. A workflow definition is very

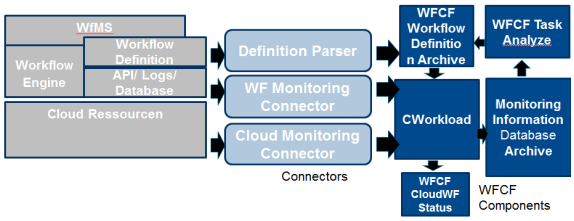

Figure 2: Monitoring of WFCF.

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 (Kübler and Minor, 2015). 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.

2.2 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.

After CWorkload has build the WFCF CloudWF Status, *CProblem* is the part of WFCF which investigates the current status of the environment that is modeled 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 (clo, 2016) 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*.

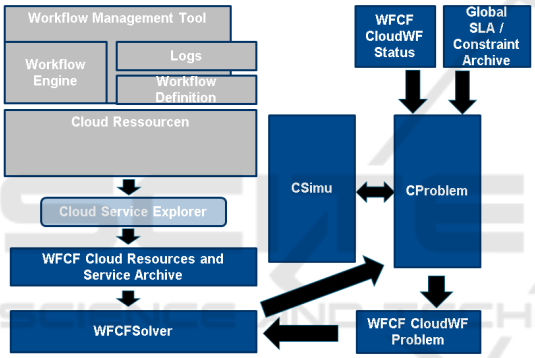

Figure 3: Management of WFCF.

Whereas CWorkload is the core of the monitoring component, the *WFCFSolver* 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. As in section 1 mentioned, the idea of CBR is that similar problems have similar solutions. In our case, a problem is a *WFCF CloudWF Problem*. A *case base* is an archive of previous problems and their solutions. The case base is not included 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 work (Kübler and Minor, 2016), we had introduced the idea of a similarity function for cloud configurations. Figure 4 shows an example configuration. It includes the container (CON1 to CON3), with the web services for the tasks (Task1 and Task2), the placement of the container to the virtual machines (VM1 and VM2) and the placement of the VM's to the hardware (PM1 and PM2). Future tests with human experts as reference were promising, but did not include workflow similarities yet.

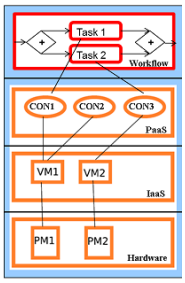

Figure 4: Example of a cloud configuration.

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 (Bergmann, 2002).

In addition, the knowledge of the workflow definitions allows to build another set of tasks that will be active in the near future. Figure 5 shows an example workflow. If Task *normalize* is the currently active task, we can say for sure, that task *limiter* will be executed as soon as the normalization 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 tasks. Since the size of the input data of the next task can be approximated from the ongoing task, the required amount of resources can be roughly estimated. Due to the run time information, from the past 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.

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 reached. This could be, for example, a time limit. In this case, the solution with the last significant problem will be chosen. The usage of CBR also opens the possibility for post-mortem analysis and improvement of the stored solution, while WFCF is otherwise idle. Beside the case base, there is the *WFCF Cloud Resources and Service Archive*. This archive contains information about the possibility of the cloud. For example, possible sizes of containers, available images and web services. 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 possible sizes and services and store them in the Resources and Service Archive.

3 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 5 shows an example workflow. The workflow is modelled in BPMN (Chinosi and Trombetta, 2012). To simplify the image, figure 5 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 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 (jbp, 2016) as a workflow management

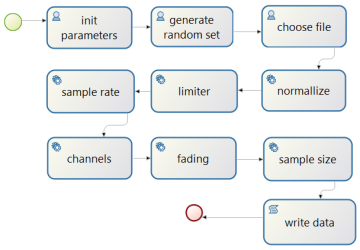

Figure 5: Sample workflow of mastering music.

tool and OpenShift (ope, 2016) (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. 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 WFCFSolver 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 prob-

lems. 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.

4 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 old solution. The goal of WFCF is a shallow integration of cloud and workflow management tools for flexible combination of tools and the optimization of resource usage. Currently, we are working on a prototype of the architecture to evaluate the concept in the future to the point where the prototype offers reconfiguration solutions for recognized problems. 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

(2016). The CLOUDS lab: Flagship projects - gridbus and cloudbus.

(2016). jBPM. https://www.jbpm.org, 2016-12-08.

(2016). OpenShift. https://www.openshift.com/, 2016-12-08.

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. 7(1):39–59.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. 53(4):50–58.

Bala, A. and Chana, I. (2011). A survey of various workflow scheduling algorithms in cloud environment. In *2nd National Conference on Information and Communication Technology (NCICT)*, pages 26–30. sn.

Baun, C., Kunze, M., Nimis, J., and Tai, S. (2011). *Cloud Computing - Web-Based Dynamic IT Services*. Springer.

Bergmann, R. (2002). *Experience management: Foundations, development methodology, and Internet-based applications*. Springer Verlag.

Chinosi, M. and Trombetta, A. (2012). BPMN: An introduction to the standard. 34(1):124–134.

Korambath, P., Wang, J., Kumar, A., Hochstein, L., Schott, B., Graybill, R., Baldea, M., and Davis, J. (2014). Deploying kepler workflows as services on a cloud infrastructure for smart manufacturing. 29:2254–2259.

Kübler, E. and Minor, M. (2015). Towards cross-layer monitoring of cloud workflows. In Helfert, M., Ferguson, D., and Muoz, V. M., editors, *CLOSER 2015 - Proceedings of the 5th International Conference on Cloud Computing and Services Science, Lisbon, Portugal, 20-22 May, 2015*, pages 389–396. SciTePress.

Kübler, E. and Minor, M. (2016). 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*, volume 2, pages 290–295. SciTePress.

Liu, X., Yuan, D., Zhang, G., Chen, J., and Yang, Y. (2010). SwinDeW-c: A peer-to-peer based cloud workflow system. In Furht, B. and Escalante, A., editors, *Handbook of Cloud Computing*, pages 309–332. Springer US.

Maurer, M., Brandic, I., and Sakellariou, R. (2013). Adaptive resource configuration for cloud infrastructure management. 29(2):472–487.

Pousty, S. and Miller, K. (2014). *Getting Started with OpenShift*. "O'Reilly Media, Inc.".

Quiroz, A., Kim, H., Parashar, M., Gnanasambandam, N., and Sharma, N. (2009). Towards autonomic workload provisioning for enterprise grids and clouds. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 50–57. IEEE.

Shoaib, Y. and Das, O. (2014). Performance-oriented cloud provisioning: Taxonomy and survey. abs/1411.5077.

Wang, J., Korambath, P., Altintas, I., Davis, J., and Crawl, D. (2014). Workflow as a service in the cloud: Architecture and scheduling algorithms. 29:546–556.

Workflow Management Coalition (1999). Workflow management coalition glossary & terminology. http://www.wfmc.org/resources 2016-12-15.