An Intelligent Cloud Management Approach for the Workflow-cloud Framework WFCF

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

Keywords: Cloud Management, Workflows, WFaaS, BPMN Tools, Automated Techniques.

Abstract: Workflow as a service is a recent trend in cloud computing. The opportunity to execute a workflow in a cloud is very attractive for business. There is, however, a lack of concepts for an integration of clouds and workflow management systems. Today's solutions are often not very effective in terms of resource usage. Further, they are not flexible enough to exchange the workflow management system, the cloud or multi-cloud environment. In this work, we evaluate WFCF our connector based integration framework for workflow management tools and clouds. WFCF uses intelligent methods to manage cloud resources with respect to monitoring information from both, the workflow and the cloud system. We introduce the architecture of WFCF and test the prototypical implementation. The evaluation is based on workflows from the music mastering domain.

1 INTRODUCTION

Cloud Computing is a field that has grown over the last few years. More and more companies use clouds for their business. Cloud computing opens new fields of business solutions. One of the new fields 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). An example for a workflow is the step-ba-step editing of raw music files to improve the audio quality. An example task within this workflow could be to add a fading effect at the end of the song, so that the music slowly becomes quieter at the end. The basic idea of WFaaS is to execute such tasks within the cloud. The implementation of the example may use a web service which receives the music file, manipulates the file and returns it. The web service could be hosted on a PaaS container or on an IaaS virtual machine (VM).

However, it is not an easy task to provide cloud resources in a way that not more resources are started than necessary (over-provisioning), but also not less than necessary (under-provisioning). Over-provisioning leads to additional costs and should be avoided. On the other hand, under-provisioning may slow down workflow execution since not enough resources are provided. In the best case, this will frustrate the user, in the worst case it can lead to violations of the Service Level Agreements (SLA), under which the WFaaS is offered as a service to the costumer. A SLA defines agreements between the provider and the customer about different aspects of the quality of service. Violations of a SLA may cause high costs and a significant loss of reputation for the provider (Shoaib and Das, 2014). It is difficult to determine how many resources are required if the status of the currently ongoing workflows is unknown. Thus, a solution that monitors the ongoing workflows and manages the cloud resources is required. A suitable management strategy should consider both, cloud and workflows in an integrated manner.

Several challenges arise for the integration of workflows and clouds. To avoid the vendor lock-in problem, a solution should be capable of handling different clouds. A WFaaS provider may decide to switch the cloud provider to save costs or to include different clouds for different services. The multi-cloud problem as described by (Ferry et al., 2013), means that

Kübler, E. and Minor, M.
An Intelligent Cloud Management Approach for the Workflow-cloud Framework WFCF.
DOI: 10.5220/0006843408410848
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 841-848
ISBN: 978-989-758-320-9

a user interacts with several different clouds, each individually. Recently, each cloud has its own API and standards, which complicates the situation for a system that operates on multiple clouds. Further, a WFaaS provider may intend to change the used workflow management system for reasons of performance. An alternative scenario is that a company aims to run an integrated cloud and workflow solution but still has its own workflow management tool and does not want to substitute the legacy tool. In both cases, an integration of workflows and cloud should allow the exchange of the used workflow tool, without the need for re-coding the core management system.
In our previous work (Kübler and Minor, 2017), we introduced the architecture of our *Workflow Cloud Framework (WFCF)*, a connector-based integration framework for workflow management tools and clouds that aims to optimize the resource utilization of cloud resources for workflows. In this work, we have implemented a prototype of WFCF and evaluate how long it takes from the registration of a new workflow instance to the start of the required resources. The remainder of the work is organized as follows. In Section 2 we will discuss related work. In Section 3, we explain the core concepts of WFCF and the architecture. In Section 4, we will present our experimental setup and the test results together with a discussion of the results. In Section 5, we draw a conclusion.

2 RELATED WORK

In this section, we introduce and briefly discuss related work. The management of cloud resources is a difficult task, as mentioned above. There is several work in the literature that addresses the problem of resource provisioning in the cloud (Shoaib and Das, 2014; Pousty and Miller, 2014; Quiroz et al., 2009; Bala and Chana, 2011; Rodriguez and Buyya, 2017). These approaches have the problem that the provisioning is either very static, does not make use of the capabilities of a cloud, or that the approaches are not implemented yet and therefore rather theoretically.
Another alternative could be *case-based reasoning (CBR)*. The idea of CBR is that similar problems have similar solutions (Aamodt and Plaza, 1994). We discussed the feasibility of CBR for cloud management in our work (Kübler and Minor, 2016).
Other approaches aim at a deeper integration of clouds and workflows (Wang et al., 2014; Korambath et al., 2014; Liu et al., 2010). They deeply integrate workflow and cloud technology, reducing the occurrence of over-provisioning and under-provisioning.

However, they strongly depend on the used cloud and workflow management tools. Therefore, they are very limited in their options to exchange either the used cloud or workflow management tool or both. This leads tho a high risk of vendor lock-in. A solution for this problem should be a more flexible integration of different workflow management tools and clouds. This is the goal of WFCF.
Closely related to the concept of WFCF is the CloudSocket project(clo, 2018). CloudSocket is a tool for end users to design their workflows and deploy them to the cloud. This is very similar to WFCF, but the main focus of CloudSocket is the user aspect. The main concern is that a user can design own workflows and deploy them fast, while WFCF rather focuses on the aspect of flexible integration of different workflow tools and clouds, to provide more freedom in chosing the tools.

3 WFCF CONCEPTS AND ARCHITECTURE

In this section, we will explain the basic concepts of *WFCF*, followed by the overall architecture. First we will introduce in short the model we use for the representation of the cloud and the workflows. Then we will explain the core concepts of WFCF, followed by the architecture.

3.1 Placement Model

The first step towards an optimal usage of resources is a representation of the actually used cloud resources and the active tasks, or the task that will be active soon. We call this the placement model. This model contains the started virtual machines (VMs) on the IaaS layer, containers from the PaaS layer and dependencies between them. We call this sub-model the cloud model. The other part of the placement model is the workflow model. This includes the currently ongoing tasks, but also the tasks that will be started next. One benefit of workflows is that the shape of the workflow is known and defined in the workflow definition. Thus, it is possible to determine a set of tasks that are potentially following the currently active tasks. If the workflow comprises only a sequence of tasks, or only contains sections with parallel tasks, all tasks within the set of following tasks will be executed for sure. If the workflow contains OR or XOR sections, the set of following tasks may contain tasks, that will not be executed next. The cloud model also contains references to the workflow model

to determine the place (VM or container) where a particular task is executed. Figure 1 shows an illustration of a sample placement model. The entire figure depicts the placement model. The lower part is the cloud model and the upper part is the workflow model. The sample cloud model includes several layers. The workflow model shows the currently active tasks (task1 only) and which container executes the task (con2). Task 2 and task 3 will be executed after task 1 has been finished. The plus symbol in the gateway indicates that the gateway is a parallel split. This means that both, task 2 and 3 will be executed for sure. If this gateway was an XOR gateway, task 2 or task 3 will be executed alternatively. At the moment, we are satisfied by knowing all potenitally following tasks. Future work will include research on heuristic methods to determine in more detail which of the tasks will be executed most likely. The figure does not show the detailed information that is stored for each element (tasks, VM, container ...). A VM for example includes information about the available resources like disk space, memory, number of cpu cores, the used operating system and so on. It also includes the usage of this resources in percentage.

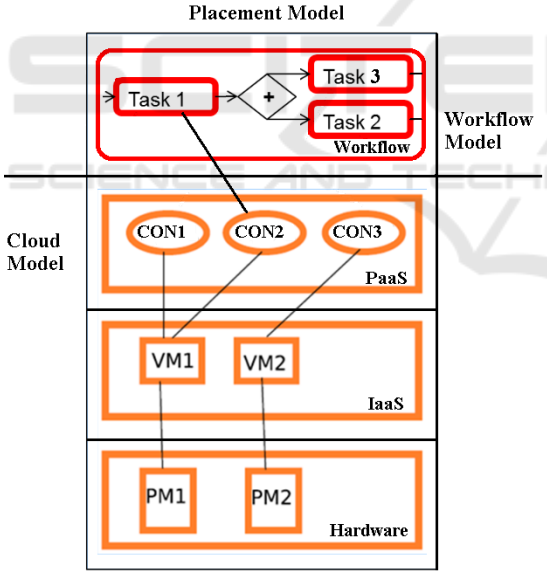

Figure 1: Illustration of a sample placement model.

3.2 The Connector Concept of WFCF

The goal of WFCF is to allow the usage of different clouds with different workflow management systems, in a way, that it can be extended later on and as easily as possible. An internal abstract representation that is independent from the actually used provider is a first necessary step. This may lead to the lost of some special ability of an actual cloud, but the abstract representation will help to manage different clouds without the need for adjustments within the model for each new cloud. However, even with an abstract representation of the cloud infrastructure there is the need to handle the actual systems. WFCF use the concept of connectors to overcome the gap between the abstract model and the actual system.

Figure 2 shows a simple example of the concept, with OpenShift [1] and Amazon Web Service (AWS)[2] as clouds in a UML like notation. The remaining components of WFCF use abstract connectors to interact with the multi-cloud. The connectors have methods to provide WFCF with information about the resource utilization (cpu usage, memory usage...), which kind of cloud is connected (IaaS or PaaS) and the ability to manage the clouds in an abstract way (start or stop VMs or containers, change resources...). Which cloud exactly is connected to WFCF, however, is not known for most parts of WFCF. The concrete connectors (in this example the OpenShift connector and the AWS EC2 connector), inherit from the abstract connector and implement the methods. We will explain later on, which parts of the WFCF framework create and configure the concrete connectors. We also use the same concept for monitoring the workflow engine, with an abstract workflow monitoring connector and concrete connectors which implement the required methods.

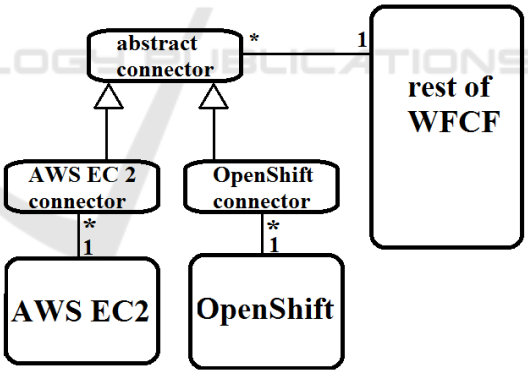

Figure 2: Simple illustration of the connector concept.

3.3 The Architecture of WFCF

Having introduced the core concepts of WFCF, we will now explain the architecture of WFCF. The WFCF framework can be divided into three parts. The monitoring part, the management part and the environment part, including the cloud and the workflow engine. Figure 3 shows an overview of the architecture of WFCF and its most important components.

[1] OpenShift: https://www.openshift.com
[2] AWS: https://aws.amazon.com

The monitoring aspect is managed by *CWorkload*. This component monitors the cloud resources and the workflow management tool. CWorkload uses the connector concept as explained above. The main tasks of CWorkload is to collect all necessary information and build the placement model. This includes the resource utilization and the tasks that are currently started or will be possibly started next. In addition, CWorkload also investigates the available workflow definitions. 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. We assume that the workflow definition is either available as file or within a database. Again, WFCF use a connector based concept to parse the information form the workflow definition file, using an abstract parser with concrete parser that implements the parsing methods, depending on the used workflow definition notation. Another task of CWorkload is the storage of execution information for tasks and the resource utilization. This includes, for example, the duration of a task. With this information, a task can be annotated with several characteristics. The characteristics describe the prospective behavior of a task. For example, the characteristic may describe whether the execution of the task usually needs more than 30 minutes or whether it is very disc intensive. In our previous works (Kübler and Minor, 2016; Kübler and Minor, 2017), we discuss our concept of tasks characteristics in more depth.

The management component of WFCF contains four parts: *CProblem, CSimu, Solver and Configurator*. The placement model which is created from CWorkload, will be investigated by CProblem. CProblem has all constraints and SLA's that are required. A constraint could be that if a task requires a web service then at least one instance of this web service is available in the cloud. Another constraint could be that the resource utilization of a VM is not higher than 90%. The constraints and SLA's have to be defined by the system administrator who manages WFCF. However, some problems may be hard to detect with only the placement model as reference. For example, to decide if the start of some very resource intensive tasks will lead to a SLA or constraint violation is not as easy as to recognize whetehr a web service has not yet been started. A simulation 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 a problem is found, for example a missing web service, CProblem notifies the *Solver*. This component has the placement model and receives from CProblem the problems with the placement. The solver then searches for a feasible solution that solves the problems that occurred. There are several possible solutions, how the Solver can solve the problems in the placement model. At the moment, we use a simple rule based implementation of the Solver. We will later on exchange it by a more intelligent method. emphCase-based Reasoning (CBR) (Aamodt and Plaza, 1994) could be such an intelligent alternative. The core idea is to retrieve similar situations and their solutions from the past in order to reuse them for the current situation. CBR has been considered for intelligent cloud management in the literature (Maurer et al., 2013). We think that the CBR approach works very well with the concept of the abstract representation of the environment. Future experiments will show whether this is correct. Similar to CProblem, the Solver has access to the simulation component to simulate the solution and also to trigger CProblem to check whether the solution has new Problems. If the solution is feasible, the *Configurator* will reconfigure the cloud. For instance, reconfiguration may include to start or stop VM's or containers. The Configurator is the only component in WFCF that knows the actually used clouds. The Configurator also has a list with all available images for containers and VM's and the necessary parameters. These parameters are currently saved as .txt files and are passed to a HashMap. This allows in a very flexible way to enter all parameters for an image without the need for re-writing programcode. For a more detailed discussion about the architecture of CWorkload, please take a look at our previous work (Kübler and Minor, 2017).

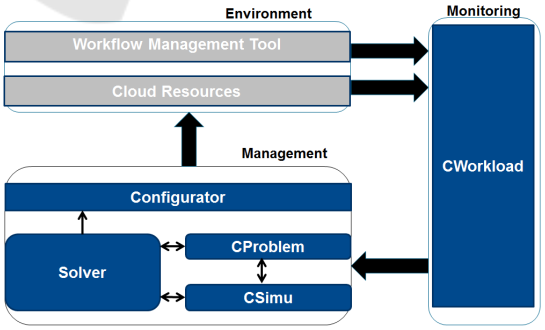

Figure 3: Overview of the architecture of WFCF.

4 EXPERIMENTAL SETUP

In this section, we present our evaluation. The goal of the evaluation is to test if our implementation

of WFCF is capable to detect and monitor new instances of workflows and handle the missing requirements correctly. We test WFCF with six workflows and compare the results with a setup without WFCF and dynamically started containers. The experimental workflows are based on the music mastering domain. A raw music file can not be used directly but must post processed. This can include several steps, ranging from normalizing the volume, change the sample rate, the playback speed or adding effects like a fading effect at the end of the song.

Four our first tests we used workflows with some initial tasks and a single processing step only. A sample workflow is shown in Figure 4. The workflow starts with the task "Init Workflow Parameters" as an initial task. In this task we define some parameters for example the url of the web service, which kind of music files we use (e.g. midi files) and so on. The next task, "Generate random variable set" randomizes some parameters. For the normalize task we randomize the value to which the volume of the song should be normalized. In the following task, the "Read data"' task, we load the file that will be processed. After that follows the actual "normalize" task. This is the processing task. The processing tasks use a web service that is not hosted on the same machine as the workflow engine. As said before, the task normalizes the volume. The last task, the "write data" task, saves the modified music file to the storage disk. During the

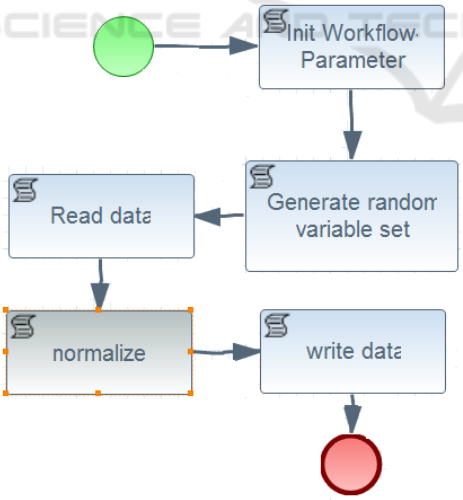

Figure 4: Example workflow with volume normalizing task.

experiment, we execute six different workflows. All of them have the same structure as described above but with different processing task. This includes the following processing task:

Normalize: normalize the volume

Limiter: limit the signal of the song by an upper and lower barrier

Table 1: Experiment 1 without WFCF and with pre-started containers.

Name	run 1	run 2	run 3	run 4	run 5	average
Limiter	33/34	17/18	17/18	17/18	17/18	20,2/21,2
Fading	19/20	17/18	19/20	17/18	17/18	17,8/18,8
Channels	13/14	12/13	12/13	13/14	12/13	12,4/13,4
Normalize	21/22	17/18	16/17	17/18	17/18	17,6/18,6
SampleRate	15/16	12/13	12/13	12/13	12/13	12,6/13,6
SampleSize	18/19	16/17	16/17	15/16	17/18	16,4/17,4

Channels: chose how many channels are used, for example mono or stereo

Fading: add a fading effect to the end of the file

Sample Rate: determine the sample rate. If the sample rate is high, the song will be played faster.

Sample Size: determine the number of bits that are used, for example 8bit or 16bit

All of the processing tasks are hosted as web services within a docker container with a tomcat 9 as web server and are managed by an OpenShift 3 private cloud (ope, 2018). OpenShift was executed on a PC with 4 x2GHz CPUs and 8GB RAM. None of the containers had resource restrictions activated.

First, we tested what is the average run time of the workflows. In preparation for this, all necessary web services are pre-started on OpenShift. All tests were conducted using the same music file with a size of 35kb. The workflows were executed by a jBPM 6 workflow engine (jbp, 2018).

During the tests, we executed every workflow five times and measured the overall run time of the workflow and the execution time of the processing task. Table 1 shows the results. All results are described in seconds. The first value in every cell is the processing time of the web service, i.e. how long it takes to build the connection to the web service, transfer the music file, process the music file, send the file back and close the connection. The second value is the overall time of the entire workflow. The latter includes the parameter initialization and the loading and saving of the file.

As we can see, the overall run time was in every run, one second higher than the processing time. The overhead could be expected, because the processing task was not the only task that was executed. But with always the same file to process it is not surprising that the effort was constant. The processing time differs slightly from run to run. Even with the same file to process, it took not always the same execution time. A slight difference could be explained by the network. However, if we take a look at run 1 for every workflow, we can see it was always higher than the following runs. We explain this effect by the used web server. If a connection to a Apache web server is recently established, following connections are done faster than for the first time.

In order to test WFCF we repeated the experiment with the only difference that the necessary web services are not pre-started. Instead, WFCF monitors if a new jBPM workflow instance is started. To do so, its searches for new log files, created by jBPM and determines which kind of workflow has been logged. The structure of the six used workflows is similar, however, it makes a difference which workflow exactly is started because of the different web services that are needed. WFCF scans the log file to determine which task is currently running. WFCF is able to determine the requirements for these tasks (if there are any), or whether the task after the currently active task has any requirements. For our tests, the processing tasks have the requirement for an individual web service, based on the processing task. The normalize task needs the normalize web service, the fading task the web service for fading and so on. WFCF scans every 5 seconds for a new log file. If a new file is found, a monitoring connector checks every 5 seconds if a new task is started. If a new task was detected, CProblem checks if there is any unsolved requirement. In our tests this will be the missing web services. The solver then will solve the problem with a simple rule based approach. It looks which web service is missing and adds a solution to the WFCF Configurator which web service should be started. In contrast to the Solver, the Configurator knows which cloud is used and which container images are available. Each image is tagged with information, including the available web services on the image. The Configurator finally has for every image a config file that specifies information that is required for starting a container in OpenShift3. This specification could be extended to further clouds than OpenShift in the future. The results of our tests are depicted in Table 2. As previously, all results are provided in seconds. The first number in each cell is the time how long it takes for WFCF to detect a new workflow instance. The second number is the time how long it needs to deploy the OpenShift container with the Apache server. The third number is the time to process the request to the web service, just like in the first experiment. The last number is the overall execution time of the workflow. We can see that the minimum time and the maximum time before a new workflow instance as noticed by WFCF vary from one to five seconds. Since WFCF searches for new log files all five seconds the result is not surprising. The time it takes to deploy the OpenShift container also varies from web service to web service, but also from run to run for the same web service. This might be caused by small inaccuracies in the time measurement (we only measured in seconds, not in milliseconds). Another aspect is OpenShift. We observed that deploying a docker container from OpenShift with the same conditions varies in its duration. A very interesting aspect is the time to process the request. We excepted that the time is nearly the same as in the previous experiment. However, the time was always a bit longer. The reason for this is partly based on the side-effect that an Apache server builds a connection faster if a connection was established recently. Since the containers are always freshly started, this effect could not occur and, thus, the time was increased. However, even with this effect in mind, the processing time is higher than excepted. The processing time for Normalize, for instance, took in the worst case (the first connection) 22 seconds. The average time for the second experiment for Normalize was 26,4. We assume this additional time comes from the fact that the container is freshly started. Even if the Apache server is working it seems that either OpenShift or the docker container needs some additional time to work with full capacity, after having started a new container. The overall run time of the workflows is nearly the same time it takes to detect the new workflow instance plus the deployment time plus the request processing time. This result was excepted. Compared to the overall run time of the workflows in experiment one, the time to execute the whole workflow is more than twice in average. However, if the processing time will be increased the additional time of the detection and deploying will have less impact in comparison.

Table 2: Experiment 2 with WFCF and without started containers.

Name	run 1	run 2	run 3	run 4	run 5	average
Limiter	5/11/28/44	5/11/26/42	5/12/28/46	5/11/26/42	3/11/26/41	4,6/11,2/26,8/43,0
Fading	5/12/26/44	3/10/26/40	5/10/25/42	4/11/28/43	3/11/26/41	4,0/10,8/26,2/42,0
Channels	2/16/21/39	3/16/22/41	5/15/21/43	3/16/21/41	3/15/27/37	3,2/15,6/21,2/40,8
Normalize	5/13/26/45	3/11/24/41	5/11/28/44	1/11/26/40	4/11/27/37	3,6/11,4/26,6/41,4
SampleRate	5/15/20/42	4/11/20/38	3/16/21/40	4/11/20/36	5/12/21/38	4,2/13,0/20,4/38,8
SampleSize	2/16/24/43	2/10/24/37	2/16/25/44	1/16/24/43	5/16/25/46	2,4/14,8/24,4/42,6

The results indicate that WFCF may not be the best choice in cases where the workload of the system is very stable or does not change over time. If for example a particular web service is needed all the time, without break, is this not an optimal situation for WFCF. Alternatively, if the time between two requests to the web service is so small, that it is not preferable to stop and restart a container or VM, the overhead that WFCF generates is probably not appropriate. However, if a web service is not needed all the time, the costs for running a, for example, EC2 instance is much higher, than just starting the service

if necessary. In this case, the additional time that is consumed to start the container or VM is acceptable if it reduces the overall costs. Another problem could arise, if the timing of an request is critical. In this case the additional time to start a container or VM could be a problem. Therefore WFCF is an solution for situations where the resources and especially the costs should be optimized without time critical operations.

5 CONCLUSION

In this paper, we introduce and test WFCF, a connector-based integration framework for workflow management tools and clouds. The behavior of the prototype of WFCF is promising and provides first evidence that we can accomplish the goal of WFCF to provide a way to integrate different workflow tools and clouds. We introduce different concepts of WFCF and describe their implementation. The connector concept works well with OpenShift. We did not include a recent version of the Eucalyptus cloud (euc, 2018) in our current experiment. However, WFCF works with an older version of the Eucalyptus cloud. We will include further clouds in the near future. The monitoring of the workflows works very well. WFCF detects and monitors different workflows and identifies the status of the active instances. WFCF uses the status information to start the required docker containers via OpenShift. The results of the experiments show that WFCF requires some time to detect and start a new web service. This latency could be a problem if the timing is important. In case the overall workload of the system does not change much, the overhead caused by WFCF could be a waste of time and effort. In other cases, WFCF could reduce the over provisioning and the costs for the owner. We will conduct further experiments considering the volatility of the system's workload in the future. There are other aspects of WFCF we have not implemented yet. Currently, the solver works with a simple rule based approach. In our future work, we aim to exchange it by a CBR approach. Another aspect is the simulation of the solution. The simulation component is not included yet. The analysis of the run time behavior of tasks and the automated annotation with task characteristics is also ongoing work. The experimental results highlight the feasibility of the tool-independent, connector-based approach. The work makes a contribution to automate the monitoring and management of cloud workflows towards intelligent cloud management in WFCF.

REFERENCES

The CLOUDS lab: Flagship projects - gridbus and cloudbus.

Cloudsocket project. https://site.cloudsocket.eu, 2018-04-08.

Eucalyptus iaas cloud. https://github.com/ eucalyptus/ eucalyptus/ wiki, 2018-04-08.

jBPM. https://www.jbpm.org, 2018-04-08.

OpenShift. https://www.openshift.com/, 2018-04-08.

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

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.

Ferry, N., Rossini, A., Chauvel, F., Morin, B., and Solberg, A. (2013). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 887–894. IEEE.

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. (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.

Kübler, E. and Minor, M. (2017). WFCF - a workflow cloud framework. In Ferguson, D., Muñoz, V. M., Cardoso, J. S., Helfert, M., and Pahl, C., editors, *CLOSER 2017 - Proceedings of the 7th International Conference on Cloud Computing and Services Science, Porto, Portugal, April 24-26, 2017*, pages 518–523. 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.

Rodriguez, M. A. and Buyya, R. (2017). A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments: Workflow scheduling algorithms for clouds. 29(8):e4041.

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.