Towards Cross-layer Monitoring of Cloud Workflows

Eric Kübler and Mirjam Minor
Institute of Informatics, Goethe University, Robert-Mayer-Str.10, Frankfurt am Main, Germany
{*ekuebler, minor*}*@informatik.uni-frankfurt.de*

Keywords: Cloud Applications Performance and Monitoring, Workflow, OpenShift.

Abstract: Prospective cloud management requires sophisticated monitoring capabilities. In this paper, we introduce a novel monitoring framework for cloud-based workflow systems called cWorkload. cWorkload integrates monitoring information from different layers of the cloud architecture. The paper puts its focus on the two-layer monitoring regarding the workflow layer and the PaaS layer. We present the layered monitoring architecture, an implementation of the two-layer cross-monitoring part, and an experimental evaluation with sample workflow data. Further, we discuss related work on cloud monitoring divided into one-layer, multi-layer, and cross-layer approaches. Our plans for future work on extending the implementation by further layers towards a cross-layer, prospective monitoring for prospective cloud management are described.

1 INTRODUCTION

Cloud management (CM) aims at an optimal resource and capacity planning. *Cloud monitoring* becomes essential to predict and keep track of the evolution of all the parameters involved in the process of assuring the *Quality of Service* (QoS) (Aceto et al., 2013). Monitoring capabilities facilitate cloud service providers to fulfill *service level agreements* (SLAs). Service consumers may use monitoring capabilities to audit whether SLAs have been violated. Today, monitoring services such as Amazon CloudWatch (Amazon, 2014b) provide data on the current state of particular cloud resources. However, they facilitate a rather reactive management of resources. The increasingly complex structure of cloud systems made of several layers requires more complex monitoring systems in future (Aceto et al., 2013).

We identified a research gap for prospective cloud monitoring capabilities. Such approaches that go beyond monitoring capabilities provided by cloud vendors might improve CM significantly. In this paper, we introduce the novel cross-layer monitoring framework cWorkload that integrates process monitoring and cloud monitoring capabilities. cWorkload is part of our process-oriented cloud management model whose fundamental ideas have been published in our previous work (Schulte-Zurhausen and Minor, 2014; Minor and Schulte-Zurhausen, 2014). The cloud management model has been inspired by the multi-tier model for cloud management introduced by Maurer et al. (Maurer et al., 2013), which we have extended by the business process perspective. An integrated monitoring solution that is aware of the ongoing business processes contributes to a better estimation of approaching workloads. Cloud monitoring information from existing tools is integrated with process monitoring. As a consequence, interventions can be planned in advance and executed in a timely manner. This improvement of the cloud management yields benefits for both, a cloud service provider and a cloud user perspective. The novel monitoring approach might reduce the amount of overprovisioned resources extremely, which is required to maintain compliance with SLAs.

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). Several instances of the same workflow can be executed at the same time. A *cloud workflow* is a workflow that is executed within a cloud environment (or within several cloud environments). For instance, a video surveillance process comprising of several analysis steps for recorded video sequences, such as identi-

fying image changes that may correspond to humans moving within the observed area, provides a sample scenario for a cloud workflow. In this paper, we make use of the workflow notion to define workload as follows: A *workload* is a task (of a workflow instance) with its input data, its number of users, and a task type (CPU intensive, network intensive, memory intensive, storage intensive). While the workload of an ongoing task is well-specified the future workloads might be known only approximately, i.e. the workloads are still underspecified. Underspecified workloads are refined as soon as all properties of their according tasks are specified, such as input data and number of users. The overall workload of a cloud system at a particular execution time is then characterized by the set of current workloads (from different tasks being executed). The remainder of this paper is organized as follows. In Section 2, we present related work. Section 3 summarizes our cloud management and cross-layer monitoring approach. Section 4 contains our hypotheses, the experimental setup, and results of our experiments. We draw a conclusion in Section 5 and point out future work.

2 RELATED WORK

In order to compare cWorkload and other cloud monitoring platforms, we classify the related work into three types: single-layer monitoring, multi-layer monitoring, and cross-layer monitoring. A single-layer monitoring tool monitors only a single layer of the cloud architecture. This could be, for example, the application layer or the IaaS layer. A multi-layer monitoring platform is able to obtain and manage monitoring information from different layers at the same time. However, the collected information from each layer is not related to other layers. Thus, multi-layer platforms are per se not capable to observe workloads across layers in an integrated manner. A cross-layer monitoring platform receives monitoring information from several layers and is able to track the impact of a workload from layer to layer, for instance, to determine the resource utilization by a cloud workflow at lower layers. In the following, we present a selection of cloud monitoring platforms. The selection and grouping of work is an extension of the list of monitoring platforms discussed by Aceto et al. (Aceto et al., 2013).

Obviously, the related work on cross-layer monitoring forms the closest affinities to our approach. *AzureWatch* (Paraleap Technologies, 2014) in combination with other third-party monitoring services is capable of monitoring the application, PaaS and IaaS layer and of aggregating the information. Monitoring values from leading indicators for scaling, such as queue depths or rate of change in demand, are combined with values from trailing indicators, such as CPU utilization, requests per second, or bandwidth. Thus, AzureWatch is a cross-layer platform. However, AzureWatch is restricted to technical solutions based on Windows Azure. The workflow paradigm is not addressed. *vRealize Hyperic* (vmware, 2014) is a cross-layer monitoring platform from VMWare. It monitors several cloud layers and is able to point out resource utilization peaks across the layers. This is called metric drill down. In contrast to our work, vRealize Hyperic does not consider a workflow layer.

Multi-layer monitoring approaches are also of interest for our work since they collect monitoring information from different layers. Some of them visualize the acquired monitoring values in common dashboards. *CloudWatch* (Amazon, 2014b) uses metrics from the Amazon Web Services (AWS), which are distributed on the PaaS and IaaS layer. CloudWatch is also capable of obtaining and evaluating monitoring information from other applications. Thus, CloudWatch is a multi-layer monitoring platform. However, due to the lack of aggregation of the pieces of information, CloudWatch is not a cross-layer monitoring platform. *Monitis* (monitis, 2014) is an application based on an agent paradigm. It is mainly for AWS and, similar to CloudWatch, is a multi-layer platform with a common dashboard for monitoring information from different layers. Further multi-layer approaches for PaaS and IaaS are, for instance, *Lattice* (Palmieri et al., 2012), *Gmone* (Montes et al., 2013) or the monitoring component of *Aneka* (Manjrasoft, 2014). *Nimsoft Monitoring Solution* (ca technologies, 2014) is a multi-layer monitoring platform primarily for the hardware and IaaS layer but it is also able to obtain pieces of information from other monitoring platforms. However, it does not track events across the layers. A similar approach for multi-layer IaaS and hardware monitoring is *GroundWork* (GroundWork, 2014) which uses the tool Nagios (Nagios, 2014) for the hardware layer. The *CLAMS* framework (Alhamazani et al., 2014) is a multi-layer approach with a special focus on integrating monitoring information from cloud environments of different vendors. Please note that the authors use an alternative notion of cross-layer monitoring regarding multi-clouds. Maurer et al. (Maurer et al., 2013) present an approach to manage a cloud configuration at different layers. They divide the resource allocation into three levels for scaling called escalation levels. Low-level metrics from the cloud infrastructure, such as *free_disk* and *packets_sent*, are mapped to high-level

monitoring parameters with the aim to fulfill the Service Level Agreements (SLAs) by automated scaling. We consider this a multi-layer approach since the information from a lower layer is propagated towards higher layers. A cross-layer monitoring would require receiving monitoring information from different layers.

Single-layer monitoring approaches are slightly related to our work. Monitoring one layer is a prerequisite for multi-layer and cross-layer monitoring. *OpenShift*, for instance, can be monitored in a single-layer manner by a monitoring component for each particular PaaS container called monitoring cartridges (Pousty and Miller, 2014). For sample work on monitoring at a single IaaS layer, we refer to the literature (OpenNebula, 2014; rackspace, 2014; LogicMonitor, 2014; Kung et al., 2011; Alcaraz Calero and Gutierrez Aguado, 2014).

3 MONITORING ARCHITECTURE

The cloud management problem could be mapped into the MAPE cycle (Monitoring - Analysis - Planning - Execution) (I.B.M. Corporation, 2006). The first step is to monitor the resources. The analysis step is to detect an event. This could be the threatening violation of an SLA, for instance. Another example for an event is the occurrence of a high workload that provides an indicator for scaling resources in order to decrease the required execution time. Step three is the search for a solution in the planning step. A solution is a reconfiguration of the cloud configuration. This might include changes in the distribution of workloads for a better fit of the resources to the requirements. This could be achieved by using configuration activities across the layers while activities at higher levels are preferable. In addition, different problem solving strategies could be considered, such as meta-heuristics (Beloglazov et al., 2012), genetic algorithms (Hu et al., 2010), or case based reasoning (Richter and Weber, 2013).

In the following, we will briefly summarize our cloud management approach that has been published in our previous work (Minor and Schulte-Zurhausen, 2014). Based on this, we will introduce the novel monitoring architecture. A multi-tier model separates physical from virtual resources in different layers. It allows cascading configuration activities from layer to layer. The model of our monitoring architecture is inspired by the cloud management model from Maurer at al. (Maurer et al., 2013) which consists of three layers for hierarchical configuration activities. In our previous work (Minor and Schulte-Zurhausen, 2014), we have introduced a workflow tier on top of the three tiers from Maurer et al.'s model in order to achieve a process-oriented perspective (see left hand side of Figure 1). The physical machine tier at the bottom is manipulated by configuration activities like add and remove compute nodes. The virtual machine tier allows activities like to increase or decrease incoming and outgoing bandwidth of a virtual machine (VM), its memory, its CPU share, or to add or remove allocated storage by x%. Further, VMs can be migrated to a different physical machine or moved to and from other clouds in case of outsourcing/insourcing capabilities. The application tier is dedicated to management activities for individual applications. The same set of activities as for VMs can be applied but with an application-specific scope. This tier can also be a PaaS platform while the VM tier can be an IaaS platform. Obviously, the migration and insourcing/outsourcing activities refer to the placement of applications on VMs at the application tier. The workflow tier addresses the placement of workflow tasks on VMs. Tasks can be migrated to another VM or tailored, i.e. split into replicated tasks with a portion of the input data each.

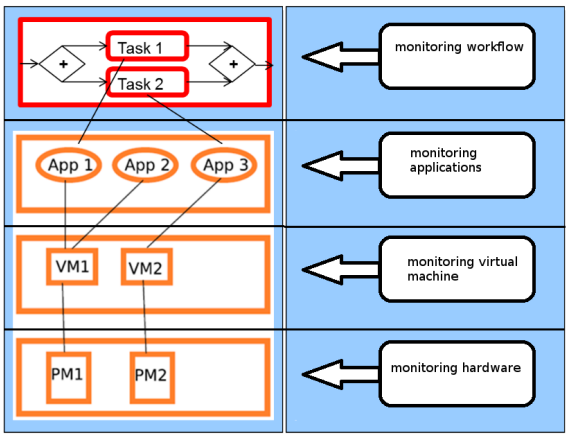

Figure 1: Process-oriented model for cloud management extending (Maurer et al., 2013).

cWorkload is a novel part of the cloud management architecture for cross-layer monitoring (see right hand side of Figure 1). In case of the workflow layer, it monitors the workflow state of instances and tasks. States reported frequently in the literature are for example "running", "active" or "suspended". We aim at using this information for a prediction of workloads as a core feature of cWorkloads. The monitoring results can be used to improve the management capabilities from a reactive towards a prospective management. To obtain the workload of an instance, we monitor the input and output data of the tasks. With

the information on the workloads we will be able to estimate the effort. This estimation will serve in our model as a means to forecast the need for scaling. We are planning to run simulations to collect information about the efforts of domain specific workloads and build an empiric knowledge repository about typical efforts.
At the PaaS layer, we aim to monitor the resources that are offered by the run-time environment. Generally, there are different tools and ways to accomplish this task depending on the used PaaS platform. In our implementation, we use OpenShift (Pousty and Miller, 2014) as a cloud environment. The cloud workflow is executed by a workflow engine running within OpenShift. Each workflow task is performed by one web service (Singh and Huhns, 2005). The workflow engine and the web services are operated on different containers. Monitoring information from all containers that are involved is gathered by the linux tool top (Unix Top, 2014).
Similar to the PaaS layer, the IaaS layer offers different tools to monitor the health of the VMs. In future work, we will use Eucalyptus (Eucalyptus Systems, 2014) or the Amazon web services (Amazon, 2014a), probably with CloudWatch (Amazon, 2014b) as a monitoring tool.
At the hardware layer, there are monitoring metrics available such as for the cpu utilization, the energy consumption and the temperature.

The long-term goal of cWorkload is to reduce the overprovisioning of resources. To achieve this goal, we will integrate the monitoring information by a smart handling of resources depending on workload forecasts. We will combine the information from different monitoring layers to obtain a better understanding of the state of our cloud.
In the remainder of this paper, we will present on the integration of the workflow and the PaaS layer. One essential part of this concept is the task placement. We define a *task placement* as the assignment of automated tasks to VMs and furthermore the assignment of VMs to physical machines (PMs). Figure 2 illustrates an example for a task placement. The placement includes two PMs and three VMs. The figure illustrates which VM is executed on which PM, for example “VM3” is running on “PM2”. This schema is repeated for the runtime environments of the PaaS layer and for the web services at the workflow layer. For instance, “s3” is executed on the “runtime3”, which is executed on “VM2”. On the right hand side, an example workflow is depicted that uses “s1” and “s5”.

The integration of the PaaS and the workflow layer is performed by cross-linking the metrics at the PaaS

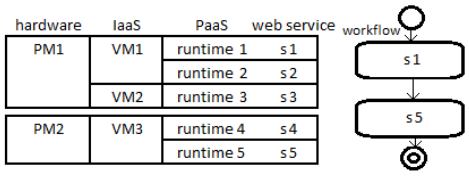

Figure 2: Sample task placement for a workflow.

layer with the monitored task states at the workflow layer. Each task in the state “running” is assigned to a measuring point for the cloud resource that has been assigned for execution. The measured values from both layers are aggregated into a multi-dimensional representation form based on time stamps from both, the workflow execution log and from the time of measuring recorded at the PaaS level. The aggregated information can be interpreted by an automated cloud management mechanism or visualized for a human being in a multi-dimensional presentation. For instance, in our preliminary implementation within OpenShift, each PaaS container comprises of exactly one web service that is assigned to a PaaS container via its URL. However, the same web service can be used by multiple workflow instances containing the same task that is performed by the web service. As a consequence, the measured metrics at the PaaS layer have to be partitioned in order to determine the portion of the measured values for each task. For example, if “s1” that is deployed on “runtime1” is used by three workflow instances at the same time and if the CPU utilization of “runtime1” is 100% we should not simply conclude that each instance uses 33% of the CPU because the CPU utilization for each instance depends on the input data. At the moment, we consider the measured values at the PaaS layer as an amalgam if multiple workflow task instances are running on the same PaaS container. A partition function is required to approximate the share of effort for each instance. We will address this in our future work in order to aggregate the values from the PaaS layer with the workflow layer more precisely than in our simplified experiments (see below).

4 EVALUATION

The cWorkload framework is implemented in a pilot system with a focus on the two-layer monitoring of the workflow and the PaaS layer. Integrating the monitoring capabilities of the workflow and the PaaS layer is an important step towards a cross-layer monitoring. A prerequisite for this is to check whether monitoring values from both layers can be aggregated in principle. The feasibility of the

two-layer approach has been investigated by running some experiments with sample workflows on the pilot system. Preliminary evaluation results have been achieved guided by the following three hypotheses:

H1: The Utilization of Cloud Resources Can Be Aligned with a Workflow Instance.
The core of the hypothesis is that it is possible to identify the PaaS resources used by a particular workflow task that has been executed. This includes recognizing the PaaS container that has been used, the utilization of the resources of the container and the duration of using the container. The hypothesis addresses a simple but critical point. Only with the capability to monitor the resources required by a task, a deep integration of workflow and cloud management can by achieved. Obviously, the PaaS resources can be monitored but for a workflow instance it is not straightforward to identify the resources (and shares of resources) that have been used by a particular workflow task during execution in a cloud environment.

H2: The More Workflow Instances Are Running on a PaaS the Higher Is the Demand for Resource Scaling.
Each workflow instance within a PaaS should be independent from each other. In order to avoid interferences between instances the resources might be scaled. The need for scaling might increase with the number of ongoing workflow instances. We assume that the sharing of resources actually occurs and that scaling is performed by a heuristically method, for instance by rules observing the number of open I/O connections to a PaaS container. The confirmation of the hypothesis is not obvious because if tasks are running on different physical machines they might not have any impact on each other.

H3: The Higher the Workloads from Ongoing Workflow Tasks on a PaaS the Higher Is the Demand for Scaling.
In addition to the number of running workflow instances, the type of workload that is processed within an instance has an impact on the demand for scaling. Again, we assume that resources are shared heuristically on the PaaS. If the workload is distributed advantageously it is possible to run several workflow instances without any interference between them. On the other hand, disadvantageous distributions of workload might lead to obstructions. To observe such effects, knowledge about the type of workloads is required.

We have designed two different, artificial workflows for the experiments. A sample modelled in BPMN (Grosskopf et al., 2009) is depicted in Figure 3. The trigger for starting a workflow instance is a message from the user as indicated by the envelope symbol. Two sample web services are included. The fibo web service ("WS: Fibo") calculates the Fibonacci number for a given number. The prim web service ("WS: Prim") calculates all prime numbers from 2 to n for a given n. The web services are executed in parallel as indicated by the plus symbol. The workflows have been modelled using the workflow designer *Intalio—BPM* (Intalio, 2014). Our work considers the task types of the workloads such as CPU intensive, storage intensive, memory intensive or network intensive tasks. The prim and fibo web services are clearly of the type CPU intensive since they mainly require CPU as a resource, do not require any storage, and use only a tiny quantity of memory and network capacities. The web services are deployed on the *OpenShift online* platform (Pousty and Miller, 2014) that provides a public cloud solution of a PaaS. Each web service is running on an OpenShift small gear, i.e. a small-size container of the runtime environment providing 512MB RAM, 1GB hard disc and a single CPU with 2.5GHz. The prim web service is deployed on gear no. 1 and the fibo web service on gear no. 2. We use a WildFly 8 application server (redhat, 2014b) as a cartridge, i.e. as a preconfigured application on top of a gear. The system *Intalio—BPM 6.5* serves as a workflow engine running on a 4-core CPU with 3.4GHz and 16GB RAM. The instances are started by hand. The CPU utilization for each gear is logged via the Linux tool *top* (Unix Top, 2014). Our cWorkload framework records the terminal output at regular intervals. In addition, Intalio—BPM logs the progress of the workflows. We run the simulation with several workloads and compare the results of the different runs. In case of the prim web service, the input data is a given number, for example 10. The result for this example is the set of prime numbers $\{2,3,5,7\}$. The input data for the fibo web service is also a number. For example, the result is 55 for the number 10. The higher the input number the higher is the workload. Consequently, this requires a higher computational effort.

Within an initial test phase, we couldn't observe any interference between the web services. Most probably, this was due to the fact that two gears on different Amazon EC2 instances had been selected automatically by the load balancer used by OpenShift online. Thus, we will discuss the results for the particular web services separately. In order to investigate hypothesis H1, cWorkload monitors the CPU utilization of the gears for new workflow instances that

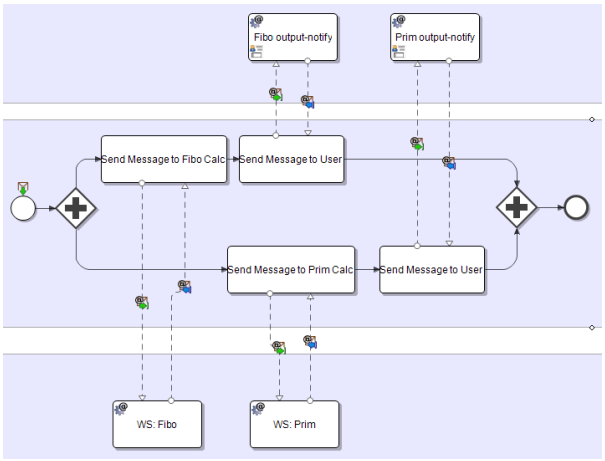

Figure 3: Workflow with a parallel call of web services.

have been started at irregular frequencies. Figure 4 and Figure 5 depict two sample sequences of values measured for the CPU utilization of gear 1 and gear 2. The sample input data was 45 for the fibo web service and 100,000 for the prim web service. There are some recognizable peaks within the time series. We have highlighted the areas where a web service is supposed to run according to the logs of the workflow engine. It can be observed that the log information matches the recognizable peaks. We conclude from the investigated samples that the utilization of cloud resources measured by the Top tool can be aligned with the CPU utilization of the gears consumed by the particular web service tasks. However, there are further single peaks in the time series. We assume that they are artifacts originating either from the Top tool, from the WildFly cartridges or from competing gears running on the EC2 instance. Hypothesis H1 has been confirmed by the measured samples.

For hypothesis H2, we have started a variable number of workflow instances simultaneously. The logs of the workflow engine have been analyzed to determine the throughput time of the web services, i.e. the duration of executing the workflow task. We have repeated this test for 5 samples with input 100,000 for the prim web service and 43 for the fibo web service. The average values are depicted in Figure 6 and Figure 7. The results show that with an increasing number of instances calling the same web service in parallel the throughput time increases significantly. At this point, our preliminary results are not yet a proof but a promising indicator that our hypothesis 2 is valid. In order to investigate hypothesis H3, we have logged the number of successfully executed requests for the fibo web services. A request is considered successful if the service has returned the result before receiving a time out. For instance, the Intalio workflow engine

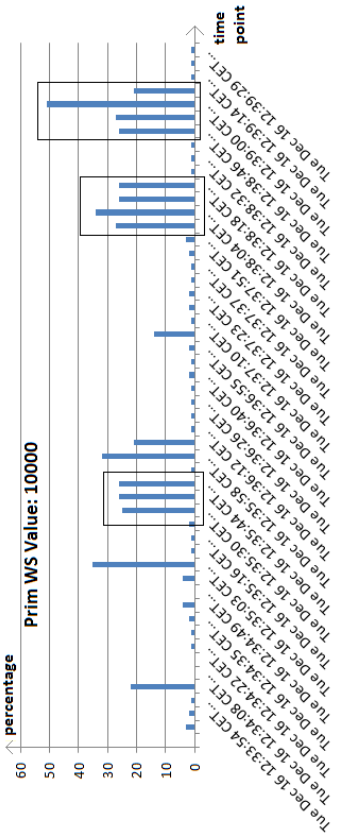

Figure 4: CPU utilization of prim.

throws such time outs after 60 seconds. We created different workloads from the same web service task by varying the input data. We have chosen an input value between 30 and 50. Each sample was repeated between 3 and 20 times. For input values of 43 and below, all executions have been successful. For input values of 48 and above, the execution was not successful. For values between, the success of the execution depends on the number of instances and on the system state. According to our experience, the duration of execution increases with the input value. Having executed 5 instances with input value 43 successfully makes it promising to run 5 instances with value 42 successfully as well. A systematic experimental proof of this experience as well as the development of a quantification function for workloads will be part of our future work. The preliminary results provide a first hint that the demand for scaling resources can be predicted approximately by means of the input values that should be calculated and, thus, by the workloads to be expected. Hypothesis H3 is not yet finally confirmed. Rather, the experiments play the role of a plausibility check. Based on the promising results, a partition function can be elaborated and hypothesis H1 can be investigated for multiple workflow in-

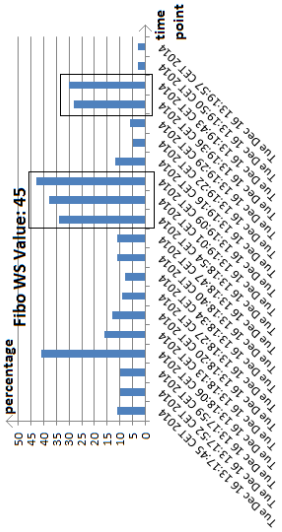

Figure 5: CPU utilization of fibo.

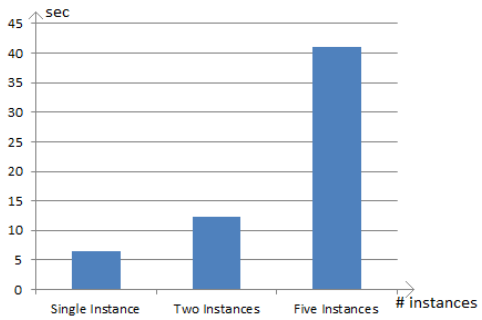

Figure 6: Average execution times for instances of fibo.

stances running in parallel in future.

5 CONCLUSION AND FUTURE WORK

In this paper, we have introduced cWorkload a novel monitoring framework for cross-layer monitoring of cloud workflows. The layer architecture of the cWorkload framework is designed in accordance with the layers of the cloud-based workflow system to be monitored. A workflow execution layer on top of the PaaS cloud system is monitored by means of the monitoring capabilities of the underlying PaaS layer. To achieve this two-layer integration, the resource utilization of the workflow tasks is determined by measuring values from the cloud resources and assigning the measured values according to the particular workloads of the workflow tasks. A preliminary evaluation of cWorkload is based on experiments with artificial workflow samples. The execution of CPU intensive

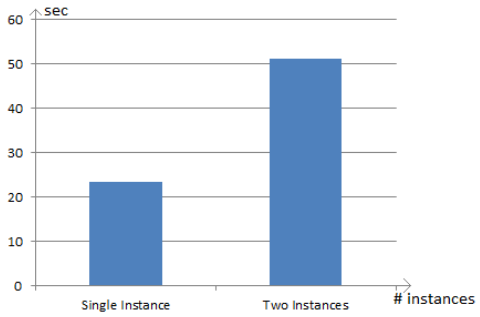

Figure 7: Average execution times for instances of prim.

workflow samples has provided that the CPU utilization of the cloud resources can be aligned with the workflow instances comprising of computationally intensive tasks. Further, we observed that the more workflow instances are running the higher are the utilization values of the according cloud resources. We conclude from the samples that the demand for scaling resources is increased by this. In addition, the higher the workloads from ongoing workflow tasks are the higher is the demand for scaling resources. The benefit of a cross-layer monitoring solution is that it provides deeper insights on the state of the system than single-layer monitoring approaches. In case of our two-layer approach that integrates the workflow layer with the PaaS layer, this results in a better understanding of the placement of workloads on PaaS containers such as OpenShift gears. Consequently, the assignment of workloads to containers can be improved, for instance by grouping tasks of different types on one gear or by selecting containers with an optimal size for a workload. Thus, the overprovisioning of resources might be reduced. The contribution of this paper and especially the preliminary experiments on the two-layer monitoring integrating the workflow and the PaaS layer provide quite promising results. The work is an important step towards the cross-layer monitoring of cloud workflows. However, it raises several novel research questions as follows.

In our future work, we will address mainly two issues. First, the experimental platform will be improved by a different technical environment. The workflow engine Intalio will be replaced by a workflow engine such as JBPM (redhat, 2014a) that facilitates the automated triggering of workflows and provides machine-readable logging information. Both will support us to conduct further experiments with a broader scope and with a deeper, automated analytics. The public cloud PaaS solution OpenShift Online will be substituted by a private installation of OpenShift in order to achieve better control of further layers below the PaaS layer. The two-layer integration will be ex-

tended to a cross-layer monitoring with multiple layers. Second, we will investigate how to predict future demands for scaling from expected workloads. The workflow notion will provide this information either based on the set of tasks to be triggered next or by a clocked simulation of further workflow execution. As a first step, a partition function for measured values on PaaS containers hosting multiple workflow task instances has to be implemented. A comparison study of the prospective solution with non-predictive cloud management approaches will highlight the importance of our approach.

REFERENCES

Aceto, G., Botta, A., de Donato, W., and Pescap, A. (2013). Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115.

Alcaraz Calero, J. and Gutierrez Aguado, J. (2014). MonPaaS: An adaptive monitoring platform as a service for cloud computing infrastructures and services. *IEEE Trans. on Services Computing*, 8(1):65–78.

Alhamazani, K., Ranjan, R., Mitra, K., Jayaraman, P., Huang, Z., Wang, L., and Rabhi, F. (2014). CLAMS: Cross-layer Multi-cloud Application Monitoring-as-a-Service Framework. In *2014 IEEE Int. Conference on Services Computing (SCC)*, pages 283–290.

Amazon (2014a). Amazon web services. http://aws.amazon.com/, 12-19-14.

Amazon (2014b). Cloudwatch. http://aws.amazon.com/cloudwatch/, 12-18-14.

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5).

ca technologies (2014). Nimsoft. http://www.ca.com/us/opscenter/ca-unified-infrastructure-management.aspx, 12-11-14.

Eucalyptus Systems (2014). https://www.eucalyptus.com/, 12-19-14.

Grosskopf, A., Decker, G., and Weske, M. (2009). *The Process: Business Process Modeling Using BPMN*. Meghan Kiffer Pr.

GroundWork (2014). http://www.gwos.com/features/, 12-19-14.

Hu, J., Gu, J., Sun, G., and Zhao, T. (2010). A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *2010 Third Int. Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, pages 89–96.

I.B.M. Corporation (2006). An architectural blueprint for autonomic computing. http://www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper V7.pdf, 11-01-14.

Intalio (2014). http://www.intalio.com/, 12-18-14.

Kung, H. T., Lin, C.-K., and Vlah, D. (2011). CloudSense: continuous fine-grain cloud monitoring with compressive sensing. *USENIX Hot-Cloud*.

LogicMonitor (2014). http://www.logicmonitor.com/, 12-19-14.

Manjrasoft (2014). Aneka. http://www.manjrasoft.com/, 12-18-14.

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

Minor, M. and Schulte-Zurhausen, E. (2014). Towards process-oriented cloud management with case-based reasoning. In *Proc. ICCBR 2014*, LNCS 8766, pages 303 – 312. Springer.

monitis (2014). http://www.monitis.com/, 12-19-14.

Montes, J., Sánchez, A., Memishi, B., Pérez, M. S., and Antoniu, G. (2013). Gmone: A complete approach to cloud monitoring. *Future Generation Computer Systems*, 29(8):2026 – 2040.

Nagios (2014). http://www.nagios.org/, 12-19-14.

OpenNebula (2014). http://docs.opennebula.org/, 12-19-14.

Palmieri, R., di Sanzo, P., Quaglia, F., Romano, P., Peluso, S., and Didona, D. (2012). Integrated monitoring of infrastructures and applications in cloud environments. In *Euro-Par 2011*, pages 45–53. Springer.

Paraleap Technologies (2014). Azurewatch. http://www.paraleap.com/azurewatch, 12-19-14.

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

rackspace (2014). Cloudkick. http://www.rackspace.com/cloud/monitoring/, 12-15-14.

redhat (2014a). jbpm. http://www.jbpm.org/, 11-26-14.

redhat (2014b). WildFly. http://www.wildfly.org/, 12-18-14.

Richter, M. M. and Weber, R. (2013). *Case-Based Reasoning: A Textbook*. Springer.

Schulte-Zurhausen, E. and Minor, M. (2014). Task placement in a cloud with case based reasoning. In *CLOSER 2014*, pages 323 – 328, Barcelona, Spain. SciTePress.

Singh, M. P. and Huhns, M. N. (2005). *Service-oriented computing - semantics, processes, agents*. Wiley.

Unix Top (2014). http://www.unixtop.org/, 12-18-14.

vmware (2014). http://www.vmware.com/products/vrealize-hyperic/, 12-19-14.

Workflow Management Coalition (1999). Glossary & terminology. 5-23-14.