Scheduling Architectures for Scientific Workflows in the Cloud

Johannes Erbel(✉), Fabian Korte, and Jens Grabowski

University of Goettingen, Institute of Computer Science, Goettingen, Germany
{johannes.erbel,fkorte,grabowski}@cs.uni-goettingen.de

Abstract. Scientific workflows describe a sequence of tasks that together form a scientific experiment. When workflows are computation or data intensive, distributed systems are used. Especially, cloud computing has gained a lot of attention due to its flexible and scalable nature. However, most approaches set up a preconfigured computation clusters or schedule tasks to existing resources. In this paper, we propose the utilization of cloud runtime models and couple them with scientific workflows to create the required architecture of a workflow task at runtime. Hereby, we schedule the architecture state required by a workflow task in order to reduce the overall amount of data transfer and resources needed. Thus, we present an approach that does not schedule tasks to be executed on resources, but schedule architectures to be deployed at runtime for the execution of workflows.

Keywords: Workflow · Models at runtime · Cloud computing · OCCI

1 Introduction

For an easy and repeatable execution of experiments, scientists utilize workflows to model their experiments as a sequence of data processing tasks [3]. Hereby, the individual tasks are commonly scheduled for execution on a cluster of preconfigured computing resources which are for example provided by a cloud service. Cloud computing however, allows to provision compute, storage, and network resources on demand [9]. Combined, these resources serve as infrastructure for arbitrary applications to be deployed. Thus, cloud computing allows to dynamically deploy individual architectures for workflow tasks requiring specific application and infrastructure configurations. Meanwhile, a few workflow languages exist that annotate workflow models with architectural requirements. Even though some approaches dynamically deploy these required architectures at runtime [15], no runtime reflection is provided. Therefore, it is difficult to make use of runtime information and tune the workflow accordingly.

In this paper, we propose an approach that combines the benefits of design time and runtime models to dynamically schedule architectures in the cloud for

We thank the Simulationswissenschaftliches Zentrum Clausthal-Goettingen (SWZ) for financial support.

F. Khendek and R. Gotzhein (Eds.): SAM 2018, LNCS 11150, pp. 20–28, 2018.
https://doi.org/10.1007/978-3-030-01042-3_2

the execution of workflows. While, the design time model describes the order of workflow tasks and their architectural requirements, the runtime model reflects the state of the workflow and the running architecture. Based on these models, we propose a simulation procedure to derive optimized cloud architectures for successive tasks to reduce the amount of resources and data that has to be transferred.

The remainder of this paper is structured as follows. Section 2 provides an overview of basic concepts and related work. Section 3 summarizes problems to be solved in order to dynamically schedule cloud architectures for workflows. Section 4 presents the proposed approach and its different components, whereby the current state of the implementation is discussed in Sect. 5. Finally, Sect. 6 provides a summary and outlook into future work.

2 Background and Related Work

Workflows are typically expressed as *Directed Acyclic Graph* (DAG) or as *Directed Acyclic Graph* (DCG), if loops are supported [3]. Hereby, nodes represent tasks, whereas links describe either a control or dataflow between them. As workflows can be compute and data intensive, distributed architectures like grids and clouds are commonly utilized by workflow systems like Pegasus [4] and Taverna [17]. Hereby, the dynamic capabilities of such systems are often not utilized as tasks are scheduled on a preconfigured cluster of computing resources. However, cloud computing is a service that allows to rent virtualized resources on demand [9]. To provide a uniform access to such a service standards like the *Open Cloud Computing Interface* (OCCI) [12] and the *Topology and Orchestration Specification for Cloud Applications* (TOSCA) [11] emerged for which corresponding metamodels have been developed [10,11]. This trend led to models at runtime approaches [5,6] which deploy a modeled architecture and keep it in sync with the cloud.

Recent approaches make use of these matured cloud orchestration techniques to directly couple cloud architectures with workflow tasks. Qasha et al. [15] extend the TOSCA standard to support workflows. Hereby, they couple tasks to cloud resources and ensure the workflows reproducibility by using containers. Even though, they deploy the containers at runtime their approach is not model-driven and only considers a design time representation. Another approach by Beni et al. [2] presents a middleware which monitors the running workflow over metamodel reflections. In addition to the generation of a workflow deployment plan, they gather runtime information to optimize the infrastructure for the workflow for future executions. However, their reflection of a running workflow does not conform to any cloud standard. Furthermore, it only considers the running cloud infrastructure and does not reflect deployed components and applications. Finally, Flowbster a workflow system presented by Kacsuk et al. [7], deploys scaling architectures for the execution of workflows. Nonetheless, the approach is not model-driven, does not conform to any cloud standard, and directly deploys the architecture for the whole workflow.

3 Problem Statement

The related work shows that there is a need for architecture aware workflows. However, most approaches only consider design time representations. Thus, the runtime state of a workflow and its underlying architecture can not be reflected preventing its architecture awareness. Furthermore, there is no process that merges cloud architectures taking the requirements of parallel executing and successive tasks into account. Therefore, current approaches require an explicit modelling of how a workflow is executed on an infrastructure making a modular design impossible. Finally, it has to be calculated when the deployment process for a task's architecture has to be triggered. However, no approach exists that combines the execution time of a task with the time needed to reconfigure a cloud architecture. Summed up, we identified the following problems:

- **P1:** Only design time models are used to execute workflows in the cloud.
- **P2:** Approaches are needed to merge cloud architectures for workflow tasks.
- **P3:** There is no connection between task execution and deployment times.

4 Approach

To ensure the reproducibility of the workflow while managing its underlying architecture at runtime, a design time as well as a runtime model is required. While the design time model provides a static representation of the workflow and its required architecture, as depicted in Fig. 1, the runtime model is used to reflect the actual state of the workflow and the cloud. Based on the design time information, the architecture scheduler assembles cloud architecture models required in different time steps by the workflow and checks their viability over a simulation approach. These models are then passed to a models at runtime engine which synchronizes the described architecture with the cloud. As soon as the architecture for a task is available the corresponding task is executed. Thus, the architecture serves as extra dependency for the task execution in addition to the input data. To loosely couple the architecture scheduler with the task execution engine, their communication is handled only over the runtime model. In the following the workflow runtime model and architecture scheduler concept are presented in more detail.

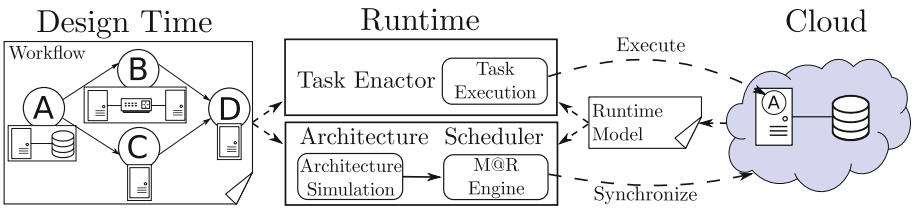

Fig. 1. Components to schedule architectures for workflow tasks

4.1 Workflow Runtime Model

To schedule cloud architectures for workflows, we propose the utilization of a design time as well as a runtime model. While the design time model is capable of storing all the information about the workflow and its architecture, the runtime model is used to reflect the actual state of the cloud and the workflow. To represent both views, we utilize OCCI, a cloud standard by the *Open Grid Forum* (OGF), which defines a uniform interface to manage cloud resources. OCCI defines an extensible datamodel for which Merle et al. [10] created a metamodel that got further enhanced in [18]. In addition to a classification and identification mechanism, OCCI defines three core base types: `Entity`, `Resource`, and `Link`, shown in Fig. 2. These core base types can be specialized to create new types which in general form OCCI extensions. Additionally, OCCI *Mixins* can be part of an extension which allow to dynamically add capabilities to specialized core types at runtime. In order to describe a task's cloud architecture the infrastructure [13], platform [14], and placement extension [8] can be used. These extensions define elements to describe how an `Application` consisting of multiple `Components` is deployed on a set of *Virtual Machines* (VMs) and how these machines are interconnected. On top of these extensions, we propose the extension shown in Fig. 2. This extension allows to model workflows running on top of architectures modeled with OCCI. Thus, already existing OCCI models can be used as underlying architecture for a sequence of tasks to be executed, as well as other extensions defined for OCCI.

The `Task` element represents a single workflow task and inherits from the `Resource` type. This element gives information about a task's current state and provides actions to start and stop its execution. Each `Task` is linked over an `ExecutionLink` to an executable `Component` which may be part of a larger `Application`. Hereby, the execution of the `Task` itself is triggered over the start action every `Component` has to implement besides other lifecycle operations. As

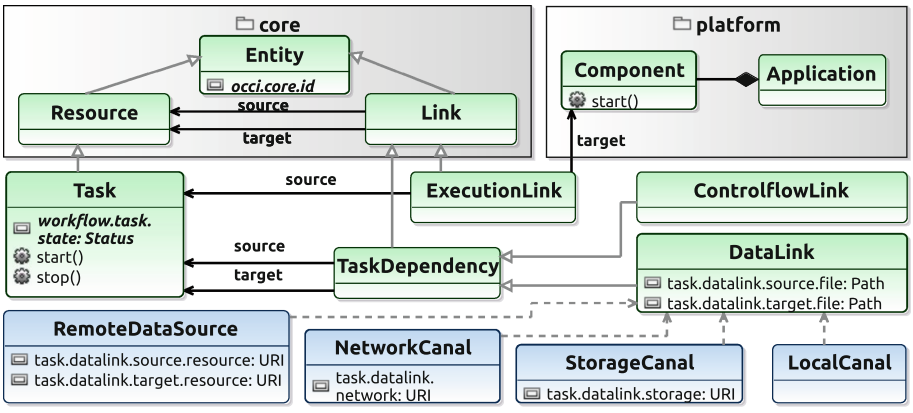

Fig. 2. OCCI workflow extension

the task execution order is predetermined, each `Task` can be directly reflected in the runtime model, whereas the architecture lying beneath the `Component` to be executed can be created at runtime. Thus, a loose coupling between the architecture and the workflow is provided. Additionally, this allows to update a workflow during its execution by adapting the task sequence or architecture requirements in the design time model. As a consequence, only the task sequence in the runtime model has to be updated as a task's architecture is extracted from the design time model shortly before it's execution.

To model a sequence of tasks, each `Task` can be linked over a `TaskDependency` to its successor. While `ControlflowLinks` only represent a simple control flow, `DataLinks` describe a control and dataflow. Because of that, a `DataLink` stores information about the source and target location of a file to be transferred which represents the in and output data of a `Task`. In addition to the workflow entities, we define three canal Mixins, a `NetworkCanal`, a `StorageCanal`, a `LocalCanal`. Using these, we can specify what kind of communication canal is used to transfer the data between two tasks. For example, the flow of data between two VMs using a network (`NetworkCanal`) or a storage (`StorageCanal`) or the flow of data within a VM (`LocalCanal`). Additionally, we specify the `RemoteDataSource` Mixin which is attached to a `DataLink` if the data is not directly located on the device hosting the executable `Component`.

4.2 Architecture Scheduler

The goal of the architecture scheduler is to compose an architecture model fitting the runtime needs of the workflow. The resulting model is then deployed over a models at runtime engine which compares the runtime state of the cloud to the desired state and performs required requests accordingly. To assemble such cloud architectures for different points in time, several questions need to be answered: *What* is required by the current and following tasks? *Where* can a task's application be deployed most efficiently? *When* does the new architecture configuration needs to be triggered?

The architecture requirement for each task is contained within the design time model. However, we adjust this static composition at runtime in order to combine architecture requirements of multiple tasks. This way a task's architecture can be defined more modular and nested workflows can be supported. To investigate possible configurations, we combine architectures of tasks which are executed in parallel based on their required workload and similarity. Hereby, we aim at utilizing each provisioned resource as much as possible to reduce the overall amount of resources required for the execution of the workflow.

In a next step, we elaborate how the assembled architecture can be combined with the running one. Hereby, we aim at reducing the amount of data that has to be transferred between successive tasks. Therefore, we fuse the merged architecture requirements with the information of the runtime model in order to deploy a task's application next to its input data. To check the viability of the resulting architecture model we perform a simulation which allows to evaluate performance metrics without an actual deployment.

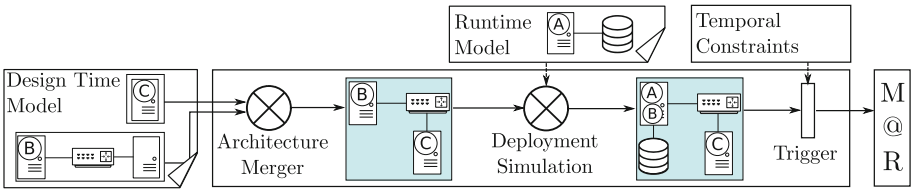

Fig. 3. Architecture Scheduler Example

To trigger the deployment of a new configuration, the task's execution time and the time to deploy the configuration is required. Therefore, we can estimate a task's execution time, e.g., based on the amount of input data [16]. Additionally, we can profile the required provisioning or deployment time of each element in a workflow's architecture. For example, we observe how long it takes to provision a VM, and the time to deploy an application on top of it. Using this information, we then estimate how long the execution of a complete deployment plan takes. Combined with the task's execution time we can then calculate when to deploy the new architecture or adapt the existing one.

An example of the described process is shown in Fig. 3. Here, task A is currently running, whereas B, C are followup tasks. Based on the architecture requirements of B and C, the scheduler assembles a suitable architecture for the following step. For example, the second virtual machine of task B is merged with the one of task C, as we historically recorded a low workload for both. Next the architecture proposal is simulated in the runtime environment context. Assume, task A and B both require a similar component and the data produced by A is required by B and stored locally. Thus, we reuse the machine that executed task A for B as no machine has to be started and no data has to be transferred. Finally, the reconfiguration process is triggered based on the estimated deployment time and the time for task A to be finished. In a next step, when A is finished, the storage and components not required anymore are deleted by removing them from the runtime model.

5 Current Status

As the proposed approach is work in progress, we discuss how the issues described in Sect. 3 are tackled. Therefore, we explain how the proposed approach is going to be implemented and validated.

P1: *Only design time models are used to execute workflows in the cloud.* To provide a runtime representation of a workflow, we propose an OCCI extension which allows to manage and reflect the workflow execution and the architecture over a standardized interface. To implement the proposed OCCI extension, we use the OCCIware tool chain [18]. This tool chain provides a graphical and textual editor to design OCCI extensions. Furthermore, it allows to automatically generate an implementation for an OCCI interface supporting the designed

extension. Then we deploy the interface implementation of the workflow extension on an OCCIware runtime server which also supports the infrastructure [13], platform [14], and placement extension [8]. This runtime server is directly connected to the cloud system on which the workflow is executed and the task specific architectures are deployed. Hereby, the server directly maintains an OCCI runtime model reflecting the state of the workflow and the cloud. To proof the concept of the workflow extension, we are going to develop a set of compute and data intensive workflows. Moreover, these workflows will be designed in such a manner that they require a sequence of complex infrastructure and application configurations for their execution. Finally, we plan to test the execution of these workflows on a private cloud using a prototypical implementation of the proposed approach.

P2: *Approaches are needed to merge cloud architectures for workflow tasks.* To lower the workflow execution time, we strive to reuse parts of a running architecture to reduce the amount of resources and the amount of data that has to be transferred. First, we identify resources that have enough capacity to handle components of parallel executing tasks. For this purpose, we are going to investigate suitable scheduling approaches to assign tasks to resources. It should be noted that software configurations of different applications may interfere with each other. Thus, every application has to be completely separated from each other. Therefor, either containerized virtualization or separated runtime environments can be used. Secondly, we fuse the resulting architecture with the runtime model to place a component next to its data. Finally, we validate the resulting configuration by deriving performance measurements using the OCCI simulation extension [1]. To actually deploy the simulated model, we utilize the OCCI compliant models at runtime engine presented in a former work [5]. To investigate the benefits and drawbacks of combining architectures of successive and parallel tasks, we plan to execute workflows with and without the proposed merging approach. Hereby, we will measure and compare the execution time required by each workflow, as well as the amount of provisioned resources and data that had to be transferred.

P3: *There is no connection between task execution and deployment times.* To reduce the time between the execution of two successive tasks, it has to be calculated when the reconfiguration of the cloud architecture has to be triggered. Therefore, we combine estimates about a task's execution time with the time required to deploy or reconfigure components within the cloud architecture. Therefore, we combine existing approaches that estimate a task's execution time [16] with historical data about the deployment and provisioning time of each single element in the cloud architecture. Thus, the architecture required for the execution of each task is deployed as soon as the previous task finishes. In order to evaluate the estimated time to trigger a cloud reconfiguration, we are going to measure the amount of time a workflow is idle because of a task which is waiting for its architecture to be deployed.

6 Summary and Outlook

In this position paper, we propose an approach that dynamically schedules cloud architecture states for the execution of workflow tasks using runtime models in order to reduce the overall amount of data transfer and resources needed. Hereby, we identified that in addition to a design time a runtime model is required to manage and reflect the state of the workflow and its underlying architecture. In the future, we will concentrate on implementing the OCCI workflow extension. Thereafter, we focus on the deployment time for cloud architectures and fuse them with the execution time of workflow tasks. Then we test multiple scheduling approaches to combine architecture models and elaborate how the resulting model can be fused with the runtime model. Finally, we are going to assemble these components into the proposed approach and proof the concept based on a set of compute and data intensive workflows.

References

1. Ahmed-Nacer, M., Gaaloul, W., Tata, S.: Occi-compliant cloud configuration simulation. In: 2017 IEEE International Conference on Edge Computing (EDGE), pp. 73–81 (June 2017)
2. Beni, E.H., Lagaisse, B., Joosen, W.: Adaptive and reflective middleware for the cloudification of simulation & optimization workflows. In: Proceedings of the 16th Workshop on Adaptive and Reflective Middleware, ARM '17, pp. 2:1–2:6. ACM (2017)
3. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: an overview of workflow system features and capabilities. Futur. Gener. Comput. Syst. **25**(5), 528–540 (2009)
4. Deelman, E., et al.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. J. **13**(3), 219–237 (2005)
5. Erbel, J., Korte, F., Grabowski, J.: Comparison and runtime adaptation of cloud application topologies based on occi. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER, vol. 1, pp. 517–525. INSTICC, SciTePress (2018)
6. Ferry, N., Chauvel, F., Song, H., Rossini, A., Lushpenko, M., Solberg, A.: Cloudmf: Model-driven management of multi-cloud applications. ACM Trans. Internet Technol. **18**(2), 16:1–16:24 (2018)
7. Kacsuk, P., Kovács, J., Farkas, Z.: The flowbster cloud-oriented workflow system to process large scientific data sets. J. Grid Comput. **16**(1), 55–83 (2018)
8. Korte, F., Challita, S., Zalila, F., Merle, P., Grabowski, J.: Model-driven configuration management of cloud applications with occi. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER, vol. 1, pp. 100–111. INSTICC, SciTePress (2018)
9. Mell, P., Grance, T.: The NIST Definition of Cloud Computing (2011)
10. Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., Tata, S.: A precise metamodel for open cloud computing interface. In: 2015 IEEE 8th International Conference on Cloud Computing, pp. 852–859 (June 2015)
11. OASIS: Topology and Orchestration Specification for Cloud Applications (2013). http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html. Accessed 27 July 2018

12. OGF: Open Cloud Computing Interface - Core (2016). https://www.ogf.org/documents/GFD.221.pdf. Accessed 27 July 2018
13. OGF: Open Cloud Computing Interface - Infrastructure (2016). https://www.ogf.org/documents/GFD.224.pdf. Accessed 27 July 2018
14. OGF: Open Cloud Computing Interface - Platform (2016). https://www.ogf.org/documents/GFD.227.pdf. Accessed 27 July 2018
15. Qasha, R., Cala, J., Watson, P.: Dynamic deployment of scientific workflows in the cloud using container virtualization. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 269–276 (Dec 2016)
16. da Silva, R.F., et al.: Toward fine-grained online task characteristics estimation in scientific workflows. In: Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science, WORKS '13, pp. 58–67. ACM (2013)
17. Wolstencroft, K., et al.: The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucl. Acids Res. **41**(W1), W557–W561 (2013)
18. Zalila, F., Challita, S., Merle, P.: A model-driven tool chain for OCCI. In: Panetto, H. (ed.) OTM 2017. LNCS, vol. 10573, pp. 389–409. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_26