Towards Domain Model Optimized Deployment and Execution of Scientific Applications in Cloud Environments

Fabian Glaser
Institute of Computer Science, University of Göttingen,
Goldschmidtstraße 7, 37077 Göttingen, Germany

Existing solutions for automatic scaling of applications in the cloud focus on the requirements of web services. A number of application servers is deployed, a load balancer is utilized to distribute the requests to these application servers, and new application servers are launched and configured when the requests exceed a certain capacity. However, the requirements for scaling scientific applications in a cloud are different. Often, these applications are used by a single scientist and the computational load is defined by the complexity of the model to be computed rather than by the number of users. In this paper, we present an alternative approach to scale scientific applications in the cloud. Hereby, the deployment scaling is driven by a domain model defined by the scientist.

1 RESEARCH PROBLEM

Todays scientific research and simulations largely depend on computing resources. While grid computing offered the possibility to share and combine large computing resources already in the past, cloud computing offers more flexibility and automatic scaling features when deploying distributed applications. Large-scale international projects, e.g., Helix Nebula[1] and the EGI Federated Cloud[2] exist, that leverage the usage of inter-connected clouds for the scientific community. While cloud computing has penetrated many business domains, the adoption in the scientific community has not been that enthusiastic. Bunch at al. (Bunch et al., 2012) identify three major reasons for this observation:

1. Cloud systems are diverse and code written for one cloud platform can not easily be ported to another platform ("vendor lock-in").
2. Current cloud systems are designed for the execution of applications from the web service domain.
3. Using cloud computing requires user expertise, rendering it inaccessible for non-computer scientists.

[1]Helix Nebula: http://www.helix-nebula.eu

[2]EGI Federated Cloud: https://www.egi.eu/infrastructure/cloud/

The first issue has been addressed recently, by introducing techniques known from *Model-Driven Development* (MDD) to the design of cloud applications (see e.g. MODAClouds (Ardagna et al., 2012)) and also by the proliferation of standards like the *Topology Orchestration and Specification for Cloud Applications* (TOSCA) (OASIS, 2013) for Cloud Orchestration and the *Open Cloud Computing Interface* (OCCI) (Nyren et al., 2011). The two later points remain as open issues. To address these issues, we introduce a framework that uses information from the domain of the scientist to appropriately scale scientific applications in cloud environments. By leveraging the information encoded in the problem definition to scale the infrastructure, we enable scientists to focus on their domain rather than being distracted by complicated cloud internals.

The remainder of this paper is structured as follows. Section 2 summarizes the research objectives of this project. Section 3 introduces three applications from different scientific domains that motivate our research, while Section 4 summarizes the current state of the art in modeling and deploying (scientific) applications in cloud environments. In Section 5, we discuss the requirements and restrictions that are defined by our motivating applications on cloud deployments and in Section 6, we introduce a framework that aims to provide a foundation to solve the research questions defined in Section 2. We comment on drawbacks of the suggested solution in Section 7. Section 8 defines the expected outcome and finally in Section 9, we summarize the current state of our research.

2 OUTLINE OF OBJECTIVES

To be able to fully leverage the benefits of cloud computing in science, we identify the following research questions:

RQ1: How can we shield the individual scientist who is willing to deploy his application in a cloud from complicated cloud internals?

RQ2: Which parameters from the domain model of the scientist have an influence on the required scale of the cloud infrastructure?

RQ3: How can these parameters be utilized to scale the cloud infrastructure conveniently?

To answer these questions, we propose a framework that uses models to deploy applications in cloud environments and utilizes information from the domain of the scientist to appropriately scale the deployed infrastructure.

3 MOTIVATING EXAMPLE APPLICATIONS

In the scope of our project, we target three different scientific application that are candidates for being executed in a cloud environment. These applications utilize different software stacks, which will be introduced in the following.

3.1 Monte-Carlo Simulation and Data Analysis in High Energy Physics

Experimental particle collision data collected by the experiments at the *Large Hadron Collider* (LHC)[3] at CERN is produced at a data rate of $\sim$15 PB/year. To establish a ground truth, particle collisions data in the same order of magnitude is generated with help of Monte-Carlo methods according to the standard model and its variations. Currently, this data is stored and analyzed with the help of a multi-tiered grid[4], which spawns the whole globe. However, due to the update of the LHC in 2013, the data rate will increase dramatically. Hence, the high energy physics community is searching for new computing models and extending the resources with cloud resources is a viable option. Production of Monte-Carlo data by simulation and also analysis of data can be easily parallelized since each simulated event (a particle beam crossing) can be simulated and analyzed independently. Cloud computing is a valuable solution for scientist conducting analysis on smaller filtered datasets (up to a few TB in size). The parallel ROOT facility (PROOF) (Ganis et al., 2008) is a commonly used tool for conducting the analysis on computing clusters built from commodity hardware. It is both parallelized (using multiple threads on multiple kernels) and distributed (master/worker architecture).

3.2 Modeling and Optimization of Public Transport Networks

LinTim[5] is a software framework, developed by the optimization working group at the University of Göttingen. It aims to support the different planning stages in public transport networks. Hereby, planning and optimization of the networks is broken up into five stages which are network design, line planning, timetabling, vehicle scheduling, and delay management. Each stage involves modeling the problem as an optimization problem which is then either solved by a heuristic or by third-party solvers. LinTim supports the solvers Xpress[6], Gurobi[7], and Cplex[8]. It is used to steer the execution of the solving steps and feeding the output data from one step into the other. LinTim itself is implemented to run on a single-core machine. The external mathematical solvers are optimized for multi-core machines. The runtime of a solving step is largely influenced by the complexity of the optimization problem.

3.3 Material Science

OpenFOAM[9] is a software toolbox, which was primarily created for numerically solving systems of partial differential equations (PDEs) of continuum mechanics problems on a predefined geometry and domain. Its inter-process communication is based on Open MPI[10]. Thereby, solving the system of PDEs, typically involves the following steps: definition of the PDEs, definition of the geometry of the domain, definition of a mesh that covers that domain and decomposition of the domain for parallel computation. Different solvers can be utilized to then numerically

[3]The Large Hadron Collider: http://home.web.cern.ch/topics/large-hadron-collider

[4]The Wordwide LHC Computing Grid: http://wlcg.web.cern.ch/

[5]LinTim: http://lintim.math.uni-goettingen.de

[6]FICO Express Optimization: http://www.fico.com/en/products/fico-xpress-optimization-suite

[7]Gurobi: http://www.gurobi.com/

[8]Cplex: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[9]OpenFOAM: http://www.openfoam.org/

[10]OpenMPI: http://www.open-mpi.org/

solve the PDEs, including finite volume methods and finite elements methods.

4 STATE OF THE ART

In the following, we shortly sketch the state of the art in modeling, deployment, configuration management, and automatic scaling of cloud applications.

4.1 Cloud Application Modeling

MODAClouds (Ardagna et al., 2012) targets cloud-provider independent development of cloud applications. Thereby, it supports the design, implementation and deployment of software with a cloud-provider independent modeling approach. The application model undergoes different modeling refinement steps, starting with a *Cloud-Enabled Computation Independent Model* (CIM), which identifies the basic components of a system, to a *Cloud-Provider Independent Model* (CPIM), which incorporates general cloud concepts like elements from *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS), and *Infrastructure-as-a-Service* (IaaS) and finalizing with the *Cloud-Provider Specific Model* (CPSM), which includes the information on how to deploy the application on a specific cloud.

4.2 Automated Scaling in the Cloud

In general, there is a differentiation between *vertical scaling*, i.e., the resizing of virtual machines or containers and *horizontal scaling*, i.e., the multiplication of virtual machines or containers (compare, e.g., (Rajan et al., 2013)). Horizontal scaling of web-services is commonly implemented with the help of a load balancer, which distributes the load on the existing application servers. If an application needs to be scaled, is triggered by so called *user-defined rules*, which define actions, e.g., the launch of an additional application server in case a certain condition is met, e.g., the number of user requests exceeds a certain threshold.

4.3 Cloud Deployments and Cloud Orchestration

Cloud orchestration frameworks, such as Amazon CloudFormation[11], OpenStack Heat[12], and Cloudify[13] utilize reusable templates that model the infrastructure and the component structure required by a cloud application and enable their orchestrated and reproducible launch. TOSCA (OASIS, 2013) aims to provide a standardized template format for orchestration. Cloudify is set to fully adopt the standard in future versions and there are ongoing implementation efforts to build a translator to convert TOSCA to the Heat Orchestration Template (HOT) format. Scalability is supported by defining *policies* that trigger a user-defined action, when a certain condition is met. Cloudinit.d (Bresnahan et al., 2011) is a tool to support the orchestrated launch, configuration and monitoring of services in an IaaS cloud. Thereby, it differentiates between different *run-levels* of the required services, where the levels define the dependencies of services to each other: services on the same run-level have no dependencies, while there might be dependencies between services running in different run-levels. The deployment and configuration of the Virtual Machines (VMs) is steered by three user defined scripts, which is submitted to the VMs at launch time: The *startup script*, which is run at start-up to install the necessary software packages and the required configuration, the *test script*, which is used to test the system after configuration and the *termination script*, which runs, when a service is shut down. In the startup script the user is free to also use Configuration Management tools like Puppet[14], Chef[15] or Ansible[16] (see Section 4.4). To offer scientific software frameworks in the cloud, Bunch et al. (Bunch et al., 2012) introduce a domain-specific language called *Neptune*. In contrast to Cloudinit.d, Neptune is specialized to steer the deployment of specific software frameworks on top of the PaaS framework AppScale[17]. Thereby, it supports the utilization of different *High Performance Computing* (HPC) software packets for distributed computation that are often used in science, including MPI, X10 and MapReduce.

4.4 Configuration Management

While configuration management tools like Puppet, Chef, and Ansible are not cloud deployment specific, they are often used to manage large scale deployments on cloud platforms. For this purpose, they use declarative text-based languages to define the de-

[11]Amazon CloudFormation http://aws.amazon.com/cloudformation/

[12]OpenStack Heat: https://wiki.openstack.org/wiki/

[13]Cloudify: http://getcloudify.org/

[14]Puppet: http://puppetlabs.com/

[15]Chef: https://www.chef.io/chef/

[16]Ansible: http://www.ansible.com/home

[17]AppScale: http://www.appscale.com

sired configuration of a (distributed) set of physical or virtual hosts. This includes, but is not limited to installed software, permissions, and network configurations. These descriptions are then used to automatically enforce and keep the hosts in the desired state. Hereby operating-system specific details, like the utilized packet manager are transparent to the user. Wettinger et al. (Wettinger et al., 2013) define different approaches to integrate Configuration Management with Cloud Orchestration.

5 DISCUSSION

While the applications represent heterogeneous approaches from different scientific domains, they share common key characteristics, which are given below:

C1. A fixed set of existing frameworks is used to evaluate or execute input models from the distinct scientific domain.

C2. These frameworks are highly specialized and encode a high amount of domain knowledge.

C3. The frameworks are not built for being executed on a scalable computing infrastructure.

C4. The utilized frameworks are responsible for distributing the computational load.

C5. The computational load to be handled largely depends on the input model provided to the framework.

When moving scientific applications to cloud environments, these characteristics have to be taken into account. While cloud computing offers great flexibility, when creating computing infrastructures, it poses the question on how these infrastructures need to be scaled to fit the computational demand. The characteristics of the frameworks described above restrict the solution space for this problem: C1 defines the software configuration of the cloud infrastructure, C2 renders it often impossible to switch to a framework which is optimized for being executed in a cloud environment, C3 does not allow the cloud environment to scale driven by the framework, and C4 restricts the infrastructure deployment to a certain architecture. While C1-C4 enforce a certain structure on the deployed compute infrastructure, C5 defines the computational load on this infrastructure and so it should present a main source for defining its scale. In addition to the requirements introduced by the characteristics defined above, another set of requirements is defined by the scientist. This might include limitations on the overall runtime in case a certain deadline needs to be met or a certain number of backups of the output data needed to be kept for reliability. Given the

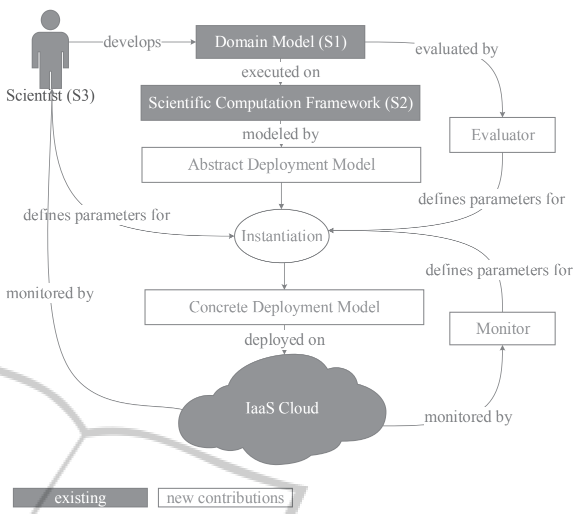

Figure 1: A framework for adapting scientific application deployments according to domain model demands.

characteristics above, we hence identify three main sources that define the requirements on an optimal deployment selection:

S1. the domain model to be computed,

S2. the scientific computation framework, which is utilized,

S3. the scientist.

A framework for automatic scalability of scientific cloud applications needs to be able to leverage requirements from all three sources. In the next section, we will introduce a framework that allows to deploy and scale the application according to these observations.

6 METHODOLOGY

Figure 1 depicts the overall framework which is used to address the research questions defined in Section 2. To leverage the full flexibility of cloud computing, the framework builds on IaaS. The components shaded in grey are already existing and the research in this project is focused on the white components.
In the following, we discuss the different components and their interaction. Thereby, we exemplify the steps with help of the example application from material science defined in Section 3.3.

6.1 Domain Models

Domain models appear in very different formats. Often only a limited set of parameters defined in the domain model have an impact on the computational de-

mand e.g., the selection of a certain solving strategy might require a certain amount of RAM in the infrastructure.
In OpenFOAM the domain model is defined with the help of a dictionary file that defines the geometry, a file that defines the boundary and initial conditions of the system of PDEs, and a properties file that defines the physical properties such as the system of PDEs to be solved. An additional file is used to decompose the domain into subdomains for parallel execution. The number of subdomains thereby should match the number of processors available in the infrastructure. While in traditional compute infrastructures, this is determined by the number of available cores, in cloud environments it should be defined according to the domain model.

6.2 Scientific Computation Frameworks

In our example, OpenFOAM represents the scientific computation framework (SCF). The SCF is the most restrictive component for the cloud deployment. It encodes how the computational load is distributed on the underlying infrastructure. As described in Section 3, OpenFOAM is parallelized using Open MPI, hence it requires an MPI cluster to run and distribute the computational load.

6.3 Abstract/Concrete Deployment Model

The deployment of the SCF is modeled with help of standardized modeling languages. Using models for describing a cloud application has the advantage that the description of the application deployment becomes cloud-provider agnostic and hence avoids vendor lock-in. It enhances comprehensibility by increased abstraction, and it fosters re-usability, since a cloud-provider agnostic model can be (semi-) automatically transformed to suit the requirements of a certain cloud-provider. By using models, we address **RQ1**.
The aforementioned TOSCA standard allows to define input parameters for application models. These parameters can include the virtual machine type to use or the number of instances of a certain type to launch. We call a model with unset parameters *abstract* and a model with instantiated parameters *concrete*. The transformation from an abstract model into a concrete model is called *instantiation*. For each domain model it must be evaluated which domain model parameters have an influence on the performance in the cloud, and these parameters then need to be mapped to and reflected by the input parameters of the abstract deployment model.
The optimal setting for the parameters needed for the instantiation is determined by three sources: the scientist, an *evaluator* and a *monitor*.

6.4 Evaluator/Monitor

To find suitable parameters settings for the requirements of a specific domain model, different strategies are possible. Hereby, we distinguish between *static evaluation*, whereby the applications is not executed in the cloud, and *dynamic evaluation*, whereby the applications are executed and monitored. Static evaluations are implemented with help of a domain specific model *evaluator*. This evaluator transforms values of parameters of interest of the domain model into suitable settings of parameters for the concrete deployment model. The evaluator is domain specific and the parameters of interest in the domain model need to be carefully selected (**RQ2**). Static evaluations are done before the application is deployed on the cloud infrastructure. The concept of the evaluator addresses **RQ3**. Dynamic evaluations can be done by manual observation of the application execution in the cloud, or automatically with help of a *monitor*. According to the outcome of the deployment evaluation, the parameters that have been used for the initial deployment are readjusted and a new instantiation of the abstract deployment model is initiated. Hereby, either a new concrete deployment model is created and deployed or the existing concrete deployment model is readjusted. The second approach is similar to the Models@Runtime approach, suggested by Ferry et al. (Ferry et al., 2014).

6.5 Deployment

The deployments of the applications are fully automated to avoid manual interaction with the cloud and enable transparent application deployment for the scientist. A cloud orchestration framework is used for the orchestrated launch of the infrastructure and a configuration management tool is utilized to automatically configure the launched infrastructure.

7 LIMITATIONS OF OUR APPROACH

While the proposed framework offers the methodology to adapt application deployments to the needs of a specific domain model, it has one limitation: Domain models have very heterogeneous formats and the relation between a domain model and the deployment model has to be defined manually for each scientific computation framework to enable automatic scaling. Hence a domain specific *evaluator* needs to be implemented for each domain. Once this mapping is done, our framework enables the scientist to focus on the development of the domain model for his problem definition and is freed from deploying the applications accordingly.

8 EXPECTED OUTCOME

We expect the following outcome of this research project:

- Contributions to the state of the art in modeling of scientific applications for the cloud,
- a novel method to leverage domain model information of the scientist to scale scientific applications in the cloud,
- a prototypical implementation of the proposed framework to demonstrate its feasibility.

9 STATE OF THE RESEARCH

In this paper, we proposed a framework for automatically scaling scientific applications in a cloud. We argue, that the traditional way of scaling applications in cloud environments does not suit the frameworks for scientific computation, since it does not take the scientific domain into account. By evaluating the domain models defined by the scientist and mapping certain key characteristics of this model to the deployment model, we are able to shield the scientist from complex cloud internals.
In the first phase of this project, we were setting up an infrastructure, to support the different steps defined by our framework. We deployed the example application defined in Section 3 in a prototypical IaaS cloud based on OpenStack[18]. A modified version of Cloudify was used for the orchestrated deployment of the applications, whereby the application models are based on Cloudify's current support for TOSCA. The configuration of the cloud applications was automated with help of the configuration management tool Ansible. Unfortunately, it became clear that current implementations of the TOSCA language are very limited when it comes to defining and launching scalable components. If TOSCA is able to properly support the scalability demands, defined in our framework is currently under evaluation.

[18]OpenStack: https://www.openstack.org/

ACKNOWLEDGEMENTS

I would like to thank my supervisor Jens Grabowski for his support and fruitful comments. This work is partially funded by the Joint Centre of Simulation Technology (SWZ) of the University of Göttingen and the Technical University of Clausthal (project 11.4.1).

REFERENCES

Ardagna, D., Di Nitto, E., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C.-S., Petcu, D., et al. (2012). Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 50–56. IEEE.

Bresnahan, J., Freeman, T., LaBissoniere, D., and Keahey, K. (2011). Managing appliance launches in infrastructure clouds. In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, page 12. ACM.

Bunch, C., Drawert, B., Chohan, N., Krintz, C., Petzold, L., and Shams, K. (2012). Language and runtime support for automatic configuration and deployment of scientific computing software over cloud fabrics. *Journal of Grid Computing*, 10(1):23–46.

Ferry, N., Brataas, G., Rossini, A., Chauvel, F., and Solberg, A. (2014). Towards Bridging the Gap Between Scalability and Elasticity. In *Proceedings of the 4th International Conference on Cloud Computing and Services Science*, pages 746–751.

Ganis, G., Iwaszkiewicz, J., and Rademakers, F. (2008). Data Analysis with PROOF. In *Proceedings of XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, number PoS(ACAT08)007 in Proceedings of Science (PoS).

Nyren, R., Edmonds, A., Papaspyrou, A., and Metsch, T. (2011). Open Cloud Computing Interface - Core. [Available online: http://ogf.org/documents/GDF.183.pdf].

OASIS (2013). Topology and Orchestration Specification for Cloud Applications (TOSCA) 1.0. [Available online; http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html fetched on 03/12/2015].

Rajan, D., Thrasher, A., Abdul-Wahid, B., Izaguirre, J. A., Emrich, S. J., and Thain, D. (2013). Case Studies in Designing Elastic Applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 466–473. IEEE Computer Society.

Wettinger, J., Behrendt, M., Binz, T., Breitenbücher, U., Breiter, G., Leymann, F., Moser, S., Schwertle, I., Spatzier, T., et al. (2013). Integrating Configuration Management with Model-driven Cloud Management based on TOSCA. In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science*, pages 437–446.