SiaaS: Simulation as a Service

Felix Willnecker,Christian Vögele
fortiss GmbH
80805 München, Germany
{willnecker,voegele}@fortiss.org

Helmut Krcmar
Technical University of Munich (TUM)
85748 Garching, Germany
krcmar@in.tum.de

Abstract

One major advantage of performance models over tests using real systems is the ability to simulate design alternatives by simply modifying or exchanging parts of such models. However, the evaluation of numerous design alternatives can be time consuming depending on the number of alternatives and the complexity of the model. To overcome this challenge, this work presents a scalable simulation service for the Palladio Component Model (PCM) workbench based on a headless Eclipse instance, a Java EE application server, packaged in a docker container and run in kubernetes. The simulation service supports parallel simulation runs, multiple PCM instances in the same container and scales out automatically, when resources of one container instance exceed. Simulation jobs are triggered by a platform-independent REST interface and can be re-used by other applications. This allows to simulate a vast amount of model instances in parallel on cloud or on-premise installations.

1 Introduction

Performance models and corresponding simulation techniques are able to predict performance metrics (e.g., resource utilization, response times, throughput) of applications systems [4]. Simulations become time consuming and resource intensive when multiple simulations are executed, or the complexity of the simulated model, the number of simulated users or the simulation time increases [4]. Today, these simulations are computed on the workstation of the user of this model in a workbench application one after each other [1]. Parallel simulations are merely possible, only by running multiple workbench instances in parallel including the resource overhead for running multiple applications. Furthermore, the workstation is busy running the simulations and restrains normal usage. A job scheduler to run several simulations overnight is typically not included making the usage of these workbenches improper for large amount of simulation runs.

A large amount of simulation runs is necessary when the evaluation of a vast amount of design alternatives is required. Such alternatives concern among others granularity of components, deployment units, or deployment topologies [3, 9]. The number of possible simulations increases exponentially with the number of possible decisions [9]. We propose a distributed and scalable simulation service in order to cope with these increased resource requirements. This service runs on an application server, requires less overhead per simulation, and can schedule and execute a vast amount of jobs in parallel. Furthermore, the system is designed to auto-scale to the required number of application servers as long as resources are available. We call this service SiaaS: Simulation as a Service[1].

2 Related Work

Guo et al (2011) designed a service-oriented architecture for simulation services, named SimSaas [2]. Although, technology has evolved since 2011, the core architecture principles of SimSaaS can be applied to the here presented service.

Dlugi et al. (2015) presented a headless Palladio Component Model (PCM) simulation engine in order to integrate it in a continuous delivery pipeline [5]. Their work is based on a headless Eclipse[2] instance requiring no user interface. This headless simulator runs on a build server instance and is started via command line. Our work reuses an advanced version of this Command Line Simulator (CLS) to execute simulations on application servers.

The microservices architecture style became increasingly popular as a core architecture principle [8]. Small, manageable and independent services are easier to develop and to maintain [8]. Therefore, we adopt the service thought but implement it in a modern architecture style for simulating PCM instances.

3 Simulation Service

SiaaS is based on Java Enterprise Edition (EE) and is accessed using a RESTful service Application Programming Interface (API). We use the Java API for RESTful Web Services (JAX-RS) and Contexts and Dependency Injection (CDI) to provide the API either run on a standard Java EE application server or packaged as Wildfly-Swarm[3] Microservice [8]. Therefore, this service can be part of a larger service infrastructure as in the Performance Management Work (PMW) tools[4] architecture. An instance can be run

[1] http://pmw.fortiss.org/tools/siaas/
[2] http://eclipse.org/
[3] http://wildfly-swarm.io/
[4] http://pmw.fortiss.org

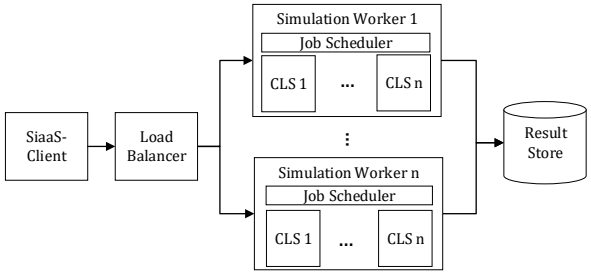

Figure 1: Simulation cluster architecture

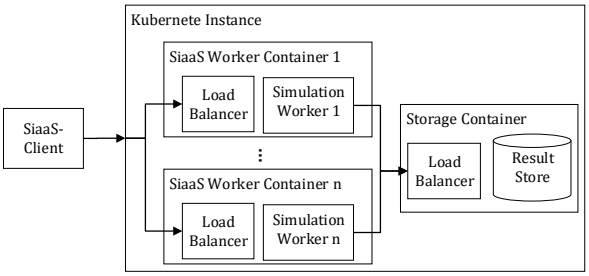

Figure 2: Simulation cluster in kubernetes instance

in a docker[5] container inside a kubernetes[6] instance to scale out automatically [6].

The complete SiaaS architecture as depicted in Figure 1 consists of three core components: (i) A load balancer distributing the jobs among the simulation worker instances, (ii) one or many worker executing simulation jobs, and (iii) a shared result store saving the simulation results. When SiaaS is run in a kubernetes cluster the load balancer is automatically part of each instance and becomes obsolete in such a setup as depicted in Figure 2 [6].

We designed SiaaS to support multiple PCM versions in parallel, so that older and newer versions can be run on the same service cluster and to test extensions not yet part of the PCM release in a larger scale. A SiaaS cluster consists of SiaaS worker instances. Each worker instance has a job scheduler, which controls a number of independent CLS instances thus control simulation jobs in separated processes. This architecture allows to separate simulation instances from each other on the same SiaaS instance without common dependencies.

The CLS uses a headless Eclipse instance started by a shell script [5]. As input the script requires a complete PCM model, the simulation time, the maximum available heap size for one simulation, and the name of the used simulation engine [5]. It currently supports the two main simulation engines of PCM: SimuCom and EventSim [1]. It is important that all SiaaS worker of a SiaaS cluster use the exact same CLSs. This can be achieved by storing the simulators in a common folder. The results of a simulation run are stored as a file archive to the disk and can be collected when a simulation job is finished. This archive is stored in the results store, which is a common file share between all instances. The same share can be used to store the CLSs.

The CLS is controlled by the job scheduler of the SiaaS. Each simulation worker controls a local job scheduler. When a new simulation is triggered, the job scheduler receives the necessary parameter to execute the simulation. If different versions of the CLS are available the correct version must be selected when starting a job. The job scheduler starts a new process using the selected CLS and assigns a unique job ID based on the session ID of the client. The scheduler continuously monitors the status of the job and provides this information via the SiaaS REST interface. Valid job status are *started*, *running*, *finished*, *failed* or *queued*.

The job scheduler is resource-aware and checks the available memory resources of the host before a new job is started. A job will be queued if less memory is available then necessary to execute a simulation. The queued job will be started as soon, as enough memory is available, after other simulation jobs have finished or failed. This allows clients to upload a batch of jobs which are automatically scheduled and executed based on the available resources.

SiaaS can run in a docker container inside a kubernetes instance [6]. New SiaaS workers can be spawned as soon as jobs need to be queued out of resource shortages. This allows SiaaS to adapt to the current load and scale out to the necessary number of instances. These two technologies are leveraged to deploy SiaaS as an elastic microservice. Figure 2 depicts the deployment structure when using docker and kubernetes [6].

SiaaS is controlled and monitored using a REST interface. This allows to integrate SiaaS into all sorts of applications independent from technology or plattform constraints. The REST interface of SiaaS provides four main interfaces: (i) schedule a new simulation job, (ii) get status of simulation job, (iii) abort a simulation job and, (iv) retrieve the results of a simulation job.

Scheduling simulation jobs requires to submit a model, the simulation time and the identifier of the used CLS. This initial request returns the unique job ID of the newly scheduled simulation job. The job ID is the only parameter necessary for the interfaces *ii-iv*. As long as a job is running, only the computing worker can provide the job status correctly. Therefore, it is important that the client is routed to the same worker instance where the simulation job was initially scheduled. Valid routing is enforced by session stickiness of the load balancer, while each new job creates a new session. The client must send its session ID with ev-

[5] http://www.docker.com/
[6] http://kubernetes.io/

ery status or abort request, in order to get a correct answer. Regular status polls must be executed by the client to avoid session timeouts. Session timeouts can occure, if a simulation runs longer without intermediate status polls. However, results are still stored in the *Result Store* after the job is finished and are available from all worker instances.

4 Evaluation

SiaaS is part of the PMW tools. It has been used and evaluated as a simulation cluster for two other PMW tools projects: (i) Deployment Unit Optimizer (DUO) and Load Test Selector (LTS) [7, 9].

The DUO project automatically selects an optimal deployment topology for an Enterprise Application (EA) for a given set of resource containers [9]. The number of potential topology grows exponentially with the number of deployable components and the number of resource containers available [9]. Therefore, a vast amount of simulations is required to detect an optimal solution [9]. SiaaS computed about 1200 simulations on 4 nodes in 8 hours using the SimuCom engine [9]. SiaaS was deployed in a fixed setup, with 4 application servers and one Apache Webserver[7] as load balancer. DUO acted as a client and queued new simulation runs based on the results from previous runs. The results were collected and analyzed and evaluated by DUO.

LTS searches for load tests design candidates matching given performance objectives like finding a minimum test set with a good component test coverage and/or high resource utilization. LTS uses a single SiaaS instance as simulation service. As with DUO new simulations are triggered by the results from simulations computed previously, which are analyzed and evaluated by LTS. LTS uses the SimCom engine with another CLS as DUO as different PCM versions are required [7].

First tests using EventSim instead of SimuCom finish already in about 30% of the time a SimuCom simulation took for the same models. Thus, future usage of SiaaS will leverage the speed increase to compute more simulation runs in a shorter period of time or with less resources.

5 Conclusions

We showed a scalable simulation service called SiaaS as part of the PMW tool chain. SiaaS can compute PCM simulations as a distributed service that is resource aware and auto scales to necessary size when run in a kubernetes instance. SiaaS is controlled via a simple REST interface allowing developers to easily integrate SiaaS into their tool chain. We demonstrated this with two PMW tools: DUO and LTS, both using SiaaS as distributed simulation service.

SiaaS is able to support multiple versions of PCM CLSs. Therefore, SiaaS is a multi-tenant application, which allows to use the same service instance for multiple purposes as demonstrated by DUO and LTS. Furthermore, this feature allows to run SiaaS as a Software as a Service (SaaS) application for multiple institutes or to run tests of multiple PCM instances. The general architecture allows developers to integrate other performance simulations or solving techniques for performance models.

Future work mainly concerns integrating SiaaS into other tools and increase the stability of the service and its infrastructure components. Furthermore, the resource-awareness of SiaaS can be extended to consider CPU utilization instead of only memory consumption for scheduling jobs and spawning new simulation instances.

[7] http://httpd.apache.org/

References

[1] S. Becker, H. Koziolek, and R. Reussner. "The Palladio Component Model for Model-Driven Performance Prediction". In: *Journal of Systems and Software* 82.1 (2009), pp. 3–22.

[2] S. Guo, F. Bai, and X. Hu. "Simulation Software as a Service and Service-Oriented Simulation E xperiment". In: *Proc. IRI '11*. 2011, pp. 113–116.

[3] A. Koziolek, H. Koziolek, and R. Reussner. "PerOpteryx: Automated Application of Tactics in Multi-objective Software Architecture Optimization". In: *Proc. QoSA '11*. ACM. 2011, pp. 33–42.

[4] A. Brunnert, A. van Hoorn, F. Willnecker, et al. *Performance-oriented DevOps: A Research Agenda*. Tech. rep. SPEC-RG-2015-01. SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), 2015.

[5] M. Dlugi, A. Brunnert, and H. Krcmar. "Model-based Performance Evaluations in Continuous Delivery Pipelines". In: *Proc. Qudos '15*. ACM. 2015, pp. 25–26.

[6] A. Verma et al. "Large-scale Cluster Management at Google with Borg". In: *Proc. EuroSys '15*. Bordeaux, France: ACM, 2015, 18:1–18:17.

[7] C. Vögele et al. "Modeling Complex User Behavior with the Palladio Component Model". In: *Proc. SSP '15*. GI - Softwaretechnik-Trends, 2015.

[8] A. Balalaie, A. Heydarnoori, and P. Jamshidi. "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". In: *IEEE Software* 33.3 (2016), pp. 42–52.

[9] F. Willnecker and H. Krcmar. "Optimization of Deployment Topologies for Distributed Enterprise Applications". In: *Proc. QoSA '16*. ACM. Venice, Italy, 2016.