12th International Workshop on Science Gateways (IWSG 2020), 10-12 June 2020

Industry Simulation Gateway on a Scalable Cloud

Jozsef Kovacs, Attila Farkas, Mark Emodi, Robert Lovas, Peter Kacsuk
Parallel and Distributed Systems Laboratory, SZTAKI, Budapest, Hungary,
{Jozsef.Kovacs, Attila.Farkas, Mark.Emodi, Robert.Lovas, Peter.Kacsuk}@sztaki.hu

Gary Pattison, Shane Kite, James Petry, George Snookes
Saker Solutions Ltd
Alcester, UK
{Gary.Pattison, Shane.Kite, James.Petry, George.Snookes}@sakersolutions.com

Tamas Kiss
Centre for Parallel Computing, University of Westminster, London, UK,
T.Kiss@westminster.ac.uk

Anastasia Anagnostou, Simon J. E. Taylor
Department of Computer Science
Brunel University London, London, UK,
{Anastasia.Anagnostou, Simon.Taylor}@brunel.ac.uk

***Abstract* — Large scale simulation experimentation typically requires significant computational resources due to an excessive number of simulation runs and replications to be performed. The traditional approach to provide such computational power, both in academic research and industry/business applications, was to use computing clusters or desktop grid resources. However, such resources not only require upfront capital investment but also lack the flexibility and scalability that is required to serve a variable number of clients/users efficiently. This paper presents how SakerGrid, a commercial desktop grid based simulation platform and its associated science gateway have been extended towards a scalable cloud computing solution. The integration of SakerGrid with the MiCADO automated deployment and autoscaling framework supports the execution of multiple simulation experiments by dynamically allocating virtual machines in the cloud in order to complete the experiment by a user-defined deadline.**

Keywords—simulation, cloud, orchestration, autoscaling, science gateway

I. INTRODUCTION

Computer simulation can be effectively used to analyse complex operational issues but in this paper we specifically looked at how it is used to support different evacuation strategies and train personnel in the context of emergency planning and response. The technique uses a model, an electronic representation of a real life system that can be used in a simulation to run an experiment to study the effects of different parameters and random number streams on the results. For example, a computer model representing a facility, its personnel, the nature of emergency, the medical facilities and the emergency resources can be produced that captures the interactions between people as they follow evacuation procedures. A computer simulation of such a situation allows different scenarios to be studied and evaluated under different conditions. These simulations can be "immersive" as they allow users to interact with the simulation via a sophisticated graphical user interface that can give them a first person point of view.

One of the key issues facing simulation modellers and analysts is the time it takes to run simulation models. As these models are stochastic (time and choice within models are typically represented by stochastic distributions that are sampled using random number streams as the model is executed), each experiment typically consists of multiple replications (the same model with the same experimental parameters but with different random number streams). Industrial models can take hours to execute a single run. A scenario consisting of multiple experiments (each with multiple replications) can take a significant amount of time. This is compounded by runs needed to test and validate a model. A typical simulation user will be running multiple simulation projects, each generating its own computational demands [1].

Saker Solutions Ltd [2] is an independent supplier of simulation solutions from the United Kingdom that underpins its simulation offers with the provision of innovative technologies which support users to gain the most from simulation projects. These offers encompass simulation related applications, operational and strategic planning applications and visualisation. An example of this is the SakerGrid Platform [3] which significantly reduces the timeframe for users to undertake model experimentation. SakerGrid is implemented as a desktop grid and its client application is a science gateway for users running simulation experiments. However, the capacity of SakerGrid is restricted by the number of available desktop grid workers at any time. As the number and complexity of the simulations are continuously growing, the need for on-demand computational resources became more and more natural and inevitable. Moreover, in many scenarios (such as evacuation simulations) executing a complex experiment by a given deadline is of paramount importance.

The huge computational requirements that simulations require and the desire to complete the experiments by given deadlines can be served by on-demand cloud resources. Utilising flexible, on-demand access to cloud computing resources and services can result in significant cost and time savings. Moreover, large, upfront capital investments can be replaced by day-to-day operational costs over a longer period of time. There are, however, definite barriers to entry for the scientific research community and smaller companies that lack the cloud-specific skills and knowledge necessary for shifting to the cloud. Additionally, organizations may struggle with achieving

maximum savings due to a lack of flexibility and scalability at the level of the application.

The European funded COLA [4] (Cloud Orchestration at the Level of Application) project set out to address these issues, and designed and developed a generic framework to support the automated deployment and scalable execution of a wide range of applications on heterogeneous cloud resources. The proposed framework is called MiCADO [5] (Microservices-based Cloud Application-level Dynamic Orchestrator), a platform for the deployment and dynamic automated scaling of applications in the cloud. MiCADO is entirely open source and implements a microservices architecture in a modular way. The modular design supports varied implementations where components can easily be replaced with a different realization of the same functionality. At the time of writing, the current implementation of MiCADO uses widely applied open source technologies such as Kubernetes [6] (container orchestrator), Occopus [7] (cloud orchestrator) and Prometheus [8] (monitoring), and some additional custom implemented components.

The work described in this paper aimed to provide the Saker Simulation Manager, the central component of SakerGrid, with on-demand cloud resources for running simulations with the help of the MiCADO framework. The aim of this extension is to dynamically expand SakerGrid with cloud resources on demand in order to complete a complex simulation experimentation by a given deadline. The rest of the paper is organised as follows. First, related works are detailed in section II. Then an introduction is given on the SakerGrid Platform and the MiCADO framework in sections III and IV. In section V a more detailed description will introduce how the two systems have been integrated. Experiences and results are presented in section VI and conclusions are given in section VII.

II. RELATED WORK

The use of deadline constraints to minimize resource use in simulation has been described by several authors. Thai et al. [9] identified that the use of deadlines in this context is the most frequent requirement of Bag of Things applications, including simulation parameter sweep applications, essentially the type of experimentation performed by Saker.

Cai et al. [10] performed experiments with deadline-based workflow applications on cloud resources and demonstrated that when task execution time is stochastic, the cost of leased resources is increased and often the deadline constraint is violated. To address this they developed a simulator for evaluating deadline-based workflow applications with a stochastic task execution time that was used to evaluate infrastructure performance.

Mao et al. [11] propose an auto-scaling mechanism to schedule VM instances by considering the budget and the job deadline. In this approach, preliminary knowledge required to do the calculations of required VMs. They investigated different types of VM to achieve lower cost without extending the deadline. They implemented their mechanism in Azure.

Vecchiola et al. [12] discussed deadline-based cloud auto-scaling at the level of hardware virtualization. They developed an algorithm and implemented in Aneka platforms, respectively.

Candeia et al. [13] proposed a formalized model that schedules an application considering the deadline and costs into account. They simulated different scenarios by determining the number of public VM used. The scheduler selects the best acceptable result for the execution.

Bicer et al. [14] propose a mathematical model for predicting the execution time and the total cost of the hybrid cloud. They calculated with the communication overhead of the clusters, which is a crucial part of a hybrid platform. The model calculates the required number of public VM instances to satisfy the deadline or cost requirements.

Menache et al. [15] propose a framework which can determine on-demand resources when there is no spot instance available. It can adaptively allocate resources, helping to maintain a user-defined policy. This idea can ensure that the performance will always be available, and jobs can be finished before the deadline.

When compared to these previous efforts, the work presented in this paper introduces a production quality, generic, cloud platform and middleware independent deadline-based autoscaling solution that can be efficiently applied in both private and public cloud settings. The use of the MiCADO framework enables application developers to concentrate on the specificities of their application and the required scaling policies, freeing them from the low-level details of cloud deployment and autoscaling solutions.

III. SAKERGRID PLATFORM

The SakerGrid Platform [3] consists of three main components: the Saker Simulation Manager, the SakerGrid Client application and the SakerGrid Manager Service. The main functionalities of the Saker Simulation Manager are to provide input parameters for a simulation run and to get the results back for visualisation. An example of the Simulation Manager's visualisation component is depicted in Figure 1, including graphical output of the simulation results and the visualisation of the simulation (for this particular project the FlexSim [16] commercial simulation package was used for model development). The SakerGrid Client application is a web service that can be accessed by a web client using its API and provides a science gateway with a user-friendly interface for performing multiple simulation runs on SakerGrid. Finally, SakerGrid Manager Service is responsible for allocating jobs to worker nodes.

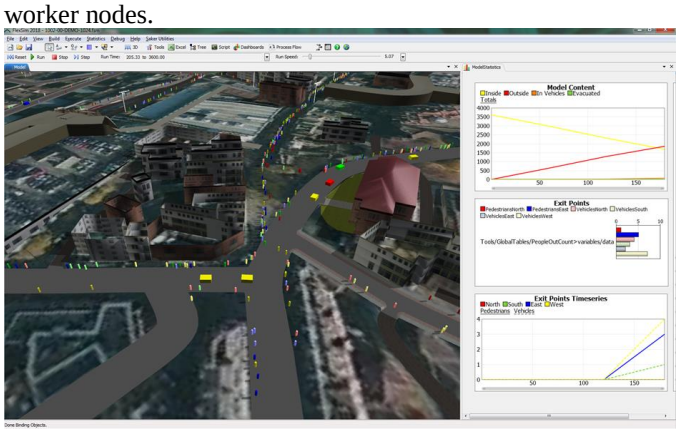

Figure 1 Visualisation in Saker Simulation Manager

The top-level architecture of SakerGrid is presented in Figure 2. The Saker Simulation Manager and the SakerGrid Client and Manager Services share a database server. This database server is an SQL server that hosts four databases. The Simulation Manager Database stores information about all simulation projects' experimentation. The Project Database contains data relevant to a specific simulation project experimentation. The Blob Store acts as a file repository for

models and collateral (e.g. DLL files). Finally, the Grid Manager Database stores configuration information for the sources (e.g. an Excel spreadsheet), and the models, scenarios and data associated with a given set of experiments.

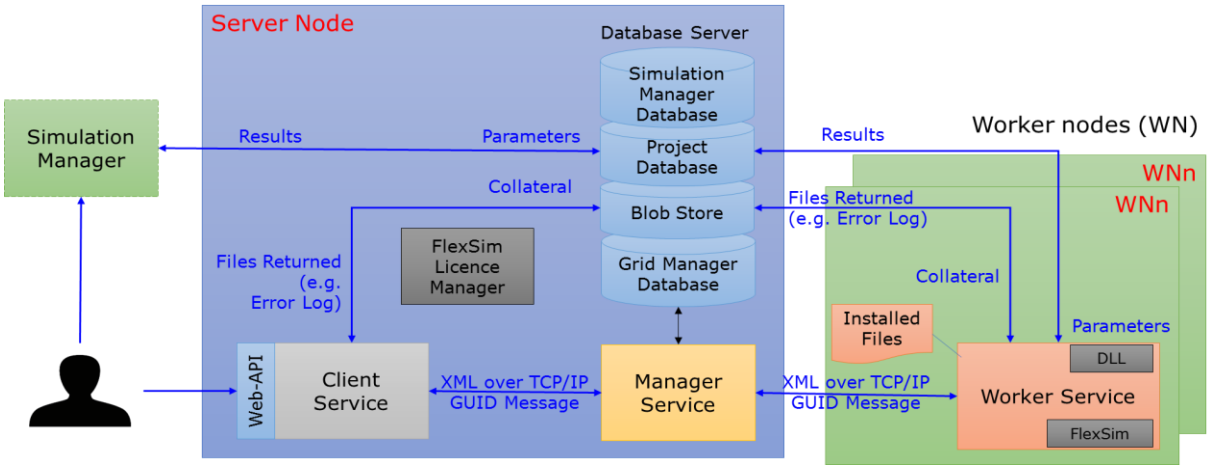

Figure 2 SakerGrid Platform top level architecture

desktop grid.

The Database Server and the SakerGrid Client and manager services reside on the same node, denoted here as Server Node. The simulation software is pre-installed on every worker node, denoted here as WNn, and the software license is managed by a license server. For this project, the worker nodes have the FlexSim software pre-installed and these run the jobs that are submitted to them by the SakerGrid Manager Service running on the server Node. The FlexSim License Manager has a 100 seat network licence installed and FlexSim also requires Windows and its relevant license to run.

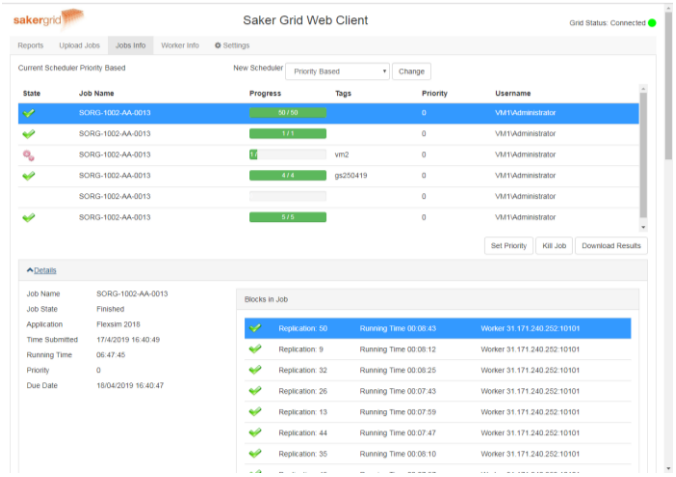

Figure 3 SakerGrid Client – jobs info view

The SakerGrid Client application provides a science gateway for users running simulations on the SakerGrid. Users can specify various parameters when uploading a job via a graphical user interface, such as the name of the job, the number of replications which the job will run (all of which will run an experiment using the same parameters and different random number stream each time), a priority value used to determine which jobs are given to the worker nodes first, a timeout value that represents the maximum time in minutes that a single replication of the job is allowed before the SakerGrid stops running it and returns an error, the simulation application to be executed on SakerGrid (e.g. FlexSim [16]), the input data

These initial input parameters of the SakerGrid have been extended, as result of the work described in Section V of this paper, according to the requirements of the scalable cloud solution, such as the deadline by that the job needs to be completed, the maximum number of processor cores that a worker can use whilst running the job, and the estimated runtime for a single replication. Once a job is submitted, the *jobs info view* allows users to monitor the progress of the job (Figure 3). An overview of the running worker nodes are shown on the top while details of the job's individual replications can be viewed by selecting the targeted job (bottom right).

IV. MiCADO Framework

When extending SakerGrid towards scalable cloud resources with deadline-based execution policy, the MiCADO framework was applied. MiCADO is an application-level multi-cloud orchestration and auto-scaling framework. The concept of MiCADO is described in detail in [5]. In this section a high-level overview of the framework is provided only to explain its architecture, building blocks and implementation.

The high-level architecture of MiCADO is presented in Figure 4. MiCADO consists of two main logical components: Master Node and Worker Node.

Master Node is the head of the cluster performing the collection of information on microservices, the calculation of optimized resource usage, the decision making, and the realization of decisions related to handling resources and scheduling microservices. Worker Nodes are volatile components representing execution environments for the microservices. These nodes are continuously allocated/released based on the dynamically changing requirements of the running microservices. Once a new Worker Node is allocated and attached to the cluster, the Master Node utilises its resources by allocating microservices on it. The input to MiCADO is a TOSCA-based [17] Application Description Template (ADT) [18] detailing the applications topology and the required scaling and security policies.

The MiCADO Master Node includes six components. MiCADO Submitter is the primary service endpoint for creating an infrastructure to run an application, and managing this infrastructure and the application itself. The incoming ADT is interpreted by the MiCADO Submitter and related parts are

forwarded to other key components. Creating new MiCADO Worker Nodes and deploying application containers on these Worker Nodes are the responsibility of the Cloud Orchestrator and Container Orchestrator components, respectively. The Cloud Orchestrator is responsible for communication with the Cloud API to allocate and release resources, and create and shut down MiCADO Worker Nodes when necessary. The Container Orchestrator allocates new microservices (realized by containers) on the Worker Nodes, keeps track of their execution and destroys them if necessary. The Monitoring System collects metrics on worker node resources and on resource usage of the container services, and makes this information available for the Policy Keeper component. It also provides alerting functionality in relation to the measured attributes to detect values that require reaction and sends these alerts to the Policy Keeper. Based on the metrics and alerts provided by the Monitoring System, the Policy Keeper applies the implemented scaling policies to make scaling decisions and call the components (Cloud and Container Orchestrators) responsible for allocating/releasing cloud resources and scheduling container services among the Worker Nodes. Moreover, this component makes sure that the Cloud and the Container Orchestrators are instructed in a synchronized way during the operation of the entire system. Lastly, the Execution Optimizer is a background microservice performing long-running calculations on demand for finding optimized setup of both cloud resources and container infrastructures.

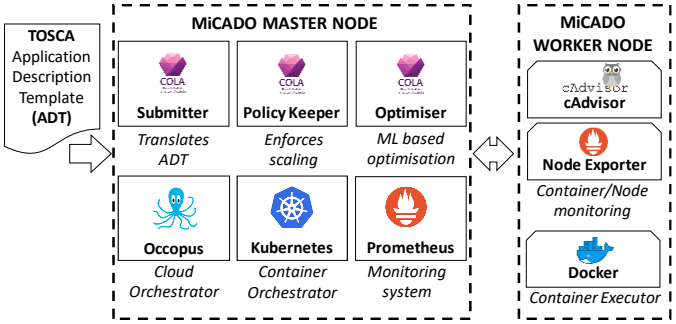

Figure 4 High-level architecture of MiCADO

MiCADO Worker Nodes contain the Node/container monitor that is responsible for measuring the load of the resources and the resource usage of the container services. The measured attributes are then provided to the Monitoring System running on the Master Node. The Container Executor starts, executes and destroys containers upon request from the Container Orchestrator. Container components are realising the user services defined in the (container) infrastructure description submitted through the MiCADO Submitter on the Master Node.

The current implementation of MiCADO utilises Occopus [7], an open source multi-cloud orchestration solution as Cloud Orchestrator that is capable of launching virtual machines (VMs) on various private (e.g. OpenStack or OpenNebula-based) or public (e.g. Amazon Web Services or CloudSigma [19]) cloud infrastructures, and also via the CloudBroker Platform [20]. For Container Orchestration, MiCADO uses Kubernetes [6]. The monitoring component is based on Prometheus [8], a lightweight, low resource consuming, but powerful monitoring tool. The MiCADO Submitter [21], Policy Keeper [22] and Optimiser components were custom implemented during the COLA Project.

V. Integration

In order to extend the execution of SakerGrid jobs to cloud computing resources, the existing SakerGrid architecture has been integrated with MiCADO. This integration enables Simulation Manager to execute simulation experiments on-demand using cloud-based virtual machines that are dynamically created and destroyed based on the specified scaling policy. When designing and implementing this integration, two major requirements were taken into consideration. First, Simulation Manager needed to be capable of executing multiple independent simulation experiments in the cloud and scaling and managing them independently from each other. Such a solution allows Saker Solutions to serve multiple clients and their requirements simultaneously. Second, the execution strategy requires a deadline-based scaling policy where a certain simulation experiment, including numerous replications, needs to be completed by a user-defined deadline while utilising only the required number of resources (virtual machines).

The integrated SakerGrid-MiCADO architecture that satisfies these initial requirements is illustrated in Figure 5. Beyond the deployment of SakerGrid Manager and Worker nodes, enhancements in both SakerGrid and MiCADO were required in order to support multiple experiments through contextualisation of the Worker nodes and the delivery of several dynamically changing parameters to MiCADO for the implementation of the desired deadline-based scaling policy.

Identification of the experiment through contextualisation was required in order to associate a Worker Service to a given simulation experimentation. When worker nodes are instantiated, MiCADO must make sure that the new Worker Service does not process replications belonging to another experiment. During the integration work, the Worker node was updated to take the experiment ID through contextualisation and the Worker Service instructs the Manager Service to send jobs belonging only to the given experiment.

An important update in the Manager Service was performed regarding exposing the necessary information for MiCADO to perform the desired deadline-based scaling of jobs. The Manager Service has been extended to make some parts of its internal database visible (see DB in Figure 5). An SQL table has been created and maintained by the Manager Service to provide information to MiCADO a) on the number of running and waiting replications, b) on the average execution time of a replication, c) on the deadline of the experiment, and d) on the current number of idle and busy workers. These parameters are necessary for MiCADO to implement deadline-based scaling and they are continuously monitored by an SQL exporter [23] (see SQL exporter in Figure 5) which is then scraped (collected) by Prometheus.

In the current implementation of MiCADO, the Master component supports the deployment, execution and scaling of only one application at a time. To support parallel execution of multiple experiments on the SakerGrid platform, a dedicated MiCADO instance is needed to be deployed for the lifetime of every experiment. For this purpose, a launcher component (see MiCADO launcher in Figure 5) has been designed and implemented, which instantiates a new MiCADO master whenever the SakerGrid Manager Service requires. When a user creates a new experiment through the SakerGrid Client Service, the SakerGrid Manager invokes a REST call of the MiCADO Launcher component interface and passes the experiment ID to

be delivered to the Worker nodes through contextualisation. The Launcher a) instantiates the MiCADO master, b) generates the TOSCA-based ADT according to a template by inserting the in our case FlexSim, to process simulation replications belonging to the experiment. Finally, Prometheus starts monitoring by collecting periodically the necessary parameters

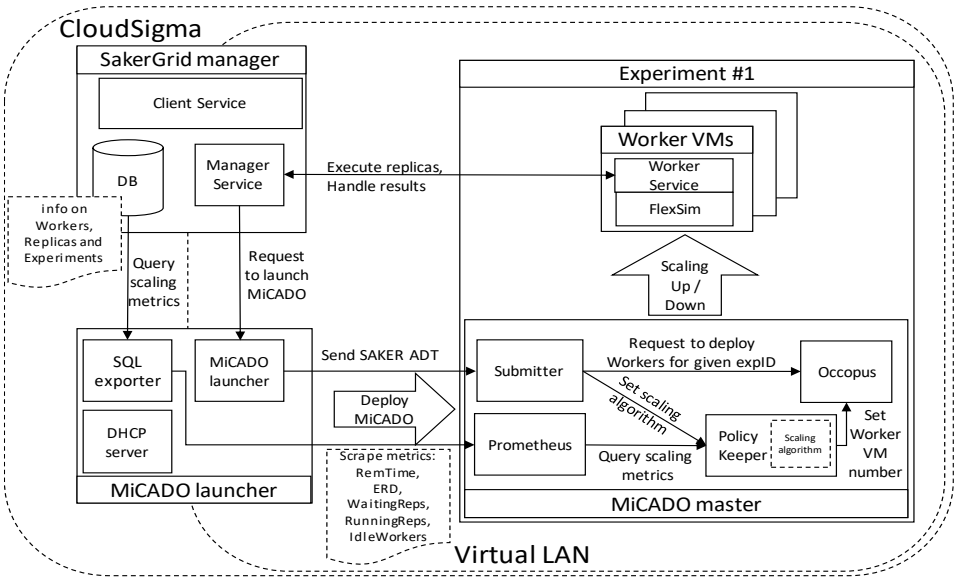

Figure 5 Integrated SakerGrid-MiCADO architecture

experiment ID and c) submits the ADT to the newly created MiCADO master instance. The ADT contains blocks to describe the components to be deployed and the related scaling policy. The policy has sections to describe parameters with assigned Prometheus queries and the scaling algorithm in Python that may refer to these parameters. The ADT is finalised by the Launcher component by adding experiment ID to the contextualisation of the Worker nodes as well as inserting experiment ID into the Prometheus queries to make the parameters contain the experiment-related information.

Although MiCADO supports various private and public cloud resources, the current implementation of the integrated solution specifically utilises the CloudSigma commercial cloud. By default, CloudSigma provides a public IP address for every created virtual machine. However, in our case the Worker VMs do not need public IP addresses and must be hidden from the outside world for safety reasons. Therefore, a private VLAN has been created for the Worker nodes together with a gateway and a DHCP server (see DHCP server in Figure 5) as well. In this private VLAN the is no to apply firewall. For the sake of efficient resource usage, the SQL exporter, the DHCP server and the Launcher components are all deployed on one single VM called MiCADO launcher.

Once the MiCADO master is alive and the launcher has generated and submitted the ADT, the deployment is performed automatically and the scaling activity is started by the MiCADO master. Based on the ADT, the Submitter generates the necessary descriptors for Policy Keeper and Occopus (see MiCADO master in Figure 5). Occopus descriptor contains the experiment ID in the contextualisation of the Worker VMs in order to specify the associated experiment. Policy Keeper descriptor includes the scaling policy and the specific Prometheus queries for the current experiment. With these descriptors, the Submitter initiates the Worker VM creation through Occopus with the initial number of VMs. Next, each newly created Worker VM starts the Worker Service which joins to the Manager Service and executes the simulation application, and delivering them to the Policy Keeper that makes the necessary decisions on the number of required Worker nodes.

The scaling algorithm is a short Python code developed for this particular environment. In MiCADO the Policy Keeper component is executing the scaling algorithm periodically which returns the number of Worker VMs to be kept for the experiment. The algorithm takes the following metrics as input: 1) remaining time until deadline (RemTime), 2) estimated running time of one replication (ERD), 3) number of replications (jobs) which has not yet started (WaitingReps), 4) number of replications being processed (RunningReps) and 5) number of idle workers (IdleWorkers). These variables are periodically queried from Prometheus and used by the scaling algorithm during execution.

The deadline-based scaling policy consists of two stages. In Stage 1 when there are waiting replications, the number of Worker VMs are periodically calculated based on the max(WaitingReps*ERD/RemTime) formula. At this stage, only upscale is enabled, but limited to the sum of running and waiting replications and to the maximum worker nodes defined by the user. With this limitation we can avoid to create too many Worker VMs causing the jobs finish much before the deadline or causing unutilised/empty VMs. When there are no more waiting replications but we are still having running replications, we step into Stage 2 and start downscaling the idle worker nodes. The algorithm queries the list of idle nodes and passes their IDs to Occopus for dropping. This is repeated until all idle worker nodes are dropped and the number of running replications reach zero. Please, note that the deadline-based policy algorithm is also an input for the system, so it can be further tuned at any time in case the policy developer decides so.

When the execution of the experiment finished (i.e. all replications successfully completed), the SakerGrid manager may instruct the MiCADO launcher to shut down the MiCADO master belonging to the given experiment to release all resources associated with the experiment. Otherwise it is possible to generate new replications in this experiment, override the

deadline and start new Worker nodes to process replications again.

VI. Experiences

Multiple simulation experiments were conducted in order to test the autoscaling behaviour of the SakerGrid-MiCADO integration. All tests used cloud resources provided by CloudSigma. The model used for the initial experimentation is an evacuation simulation developed in FlexSim where a single replication of the model is executed in 25 minutes (ERD = 25).

Figure 6 shows two simulation experiments executed one after the other. The upper part of the figure illustrates the number of VMs actively running, while the lower part is the number of replications still to be executed.

In case of the first experiment the user estimated the expected run-time of a replication correctly. As it can be seen from the chart MiCADO scales up the number of VMs based on this estimate and this number stays stabile until the end of the experiment. Once all replications are assigned to workers, MiCADO starts scaling down the VMs as the last replications on the VMs finish.

Figure 6 Evacuation simulation experiments with FlexSim using the integrated SakeGrid – MiCADO

The second experiment shows an example for a situation where MiCADO needed to perform adjustments regarding the number of VMs during execution in order to complete the experiment by the set deadline. It is visible from the chart that MiCADO increases the number of VMs several times during execution in order to assure completion before the deadline. This experiment illustrates how MiCADO can adjust in case of incorrect initial user estimation or when this estimation is almost impossible due to varying length of replications.

VII. Conclusion

As simulation experimentation requires large computational power and the execution of experiments is typically deadline-based, cloud computing offers a viable option to provide scalable resources on demand for such tasks. This paper described how a desktop grid based commercial simulation gateway was integrated with a cloud-based automated deployment and autoscaling framework. The resulting solution enables the deadline-based execution of multiple simultaneous simulation experiments from a high-level user interface on dynamically provisioned cloud resources.

It must also be noted that, at the time of writing, the UK is in lockdown in response to the COVID-19 pandemic and many simulation modellers and analysts are working from home. The transition of SakerGrid to cloud provides a capability to run simulations remotely, an additional business benefit from moving to cloud that was not foreseen at the outset of the project.

Work is currently ongoing to port the solution to Microsoft Azure (a cloud resource most preferred by Saker clients) and to roll it out and offer it in production for perspective clients.

Acknowledgment

This work was funded by the COLA Cloud Orchestration at the level of Applications (Project No. 731574) and ASCLEPIOS Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare (Project No. 826093).

References

[1] Taylor, S.J.E. (2019) Distributed simulation: state-of-the-art and potential for operational research, European Journal of Operational Research, 273(1):1-19.

[2] “Saker Slolutions Ltd” [Online]. Available: https://www.sakersolutions.com/. [Accessed: 1-Mar-2019].

[3] S. Kite, C. Wood, S. J. E. Taylor and N. Mustafee, "Sakergrid: Simulation experimentation using grid enabled simulation software," Proceedings of the 2011 Winter Simulation Conference (WSC), Phoenix, AZ, 2011, pp. 2278-2288. doi: 10.1109/WSC.2011.6147939

[4] “COLA – Cloud Orchestration at the Level of Application.” [Online]. Available: https://project-cola.eu/. [Accessed: 1-Mar-2019].

[5] T. Kiss, et al., “MiCADO - Microservices-based Cloud Application-level Dynamic Orchestrator”, Future Generation Computer Systems, Vol 95, pp 937 – 946, May 2019. DOI: https://doi.org/10.1016/j.future.2017.09.050.

[6] Kubernetes, “Production-Grade Container Orchestration.” [Online]. Available: https://kubernetes.io/. [Accessed: 1-Mar-2019].

[7] J. Kovacs, P. Kacsuk, “Occopus: a Multi-Cloud Orchestrator to Deploy and Manage Complex Scientific Infrastructures”, Journal of Grid Computing, vol 16, issue 1, pp 19-37, 2018.

[8] “Prometheus,” [Online]. Available: https://prometheus.io/. [Accessed: 1-Mar-2019].

[9] Thai, L., B. Varghese, and A. Barker. 2018. “A Survey and Taxonomy of Resource Optimisation for Executing Bag-of-Task Applications on Public Clouds”. *Future Generation Computer Systems* 82(May 2018):1-11.

[10] Cai, Z., Q. Li, and X. Li. 2017. “A Toolkit for Simulating Workflows with Cloud Resource Runtime Auto-Scaling and Stochastic Task Execution Times”. *Journal of Grid Computing* 15(2):257-272.

[11] Mao, M., J. Li, and M. Humphrey. 2010. “Cloud Auto-Scaling with Deadline and Budget Constraints”. In *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing*, Brussels, 2010, pp. 41-48. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. doi: 10.1109/GRID.2010.5697966

[12] Vecchiola, C., R. N. Calheiros, D. Karunamoorthy, and R. Buyya. 2012. “Deadline-driven Provisioning of Resources for Scientific Applications in Hybrid Clouds with Aneka”. *Future Generation Computer Systems* 28(1):58-65.

[13] D. Candeia, R. Araujo, R. Lopes and F. Brasileiro, "Investigating Business-Driven Cloudburst Schedulers for E-Science Bag-of-Tasks Applications," 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, 2010, pp. 343-350, doi: 10.1109/CloudCom.2010.67.

[14] T. Bicer, D. Chiu and G. Agrawal, "Time and Cost Sensitive Data-Intensive Computing on Hybrid Clouds," 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, 2012, pp. 636-643, doi: 10.1109/CCGrid.2012.95.

[15] Menache, I., Shamir, O., & Jain, N. (2014). On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In 11th International Conference on Autonomic Computing ({ICAC} 14) (pp. 177-187).

[16] “FlexSim Simulation Software” [Online]. Available: https://www.flexsim.com/. [Accessed: 1-Mar-2019].

[17] Oasis, “TOSCA Simple Profile in YAML Version 1.2.” [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/TOSCA-Simple-Profile-YAML-v1.2.html. [Accessed: 5-Mar-2019].

[18] J. Deslauriers, T. Kiss, G. Pierantoni, G. Gesmier, G. Terstyanszky: Enabling Modular Design of an Application-Level Auto-Scaling and Orchestration Framework using TOSCA-based Application Description Templates, 11th International Workshop on Science Gateways, IWSG 2019. Ljubljana, Slovenia 12 - 14 Jun 2019.

[19] Cloudsigma Holding AG. “Cloud servers & Hosting”. [Online]. Available: https://www.cloudsigma.com/. [Accessed: 5-Mar-2019].

[20] CloudBroker GmbH., “Compute-intensive applications in the cloud.” [Online]. Available: http://cloudbroker.com/. [Accessed: 5-Mar-2019].

[21] G. Pierantoni, T. Kiss, G. Terstyanszky, J. Deslauriers, G. Gesmier, H. Dang: Describing and Processing Topology and Quality of Service Parameters of Applications in the Cloud. Journal of Grid Computing, 2020.

[22] J. Kovacs, “Supporting Programmable Autoscaling Rules for Containers and Virtual Machines on Clouds”, Journal of Grid Computing (2019) 17: 813. https://doi.org/10.1007/s10723-019-09488-w

[23] Prometheus SQL exporter, https://github.com/free/sql_exporter [Accessed: 5-Mar-2020]