Timo Korvola*, Jari Lappalainen, Jukka K. Nurminen

Simulation-based optimization in the cloud

Keywords: optimization, cloud computing, dynamic simulation, genetic algorithm

***Corresponding Author: Timo Korvola:** VTT, E-mail: Timo.Korvola@vtt.fi
Jari Lappalainen: VTT, E-mail: Jari.Lappalainen@vtt.fi
Jukka K. Nurminen: University of Helsinki, E-mail: Jukka.K.Nurminen@helsinki.fi

1 Background

A proper dynamic simulation model provides a tool for simultaneously evaluating process and control design. This is especially valuable for engineering complex industrial processes. While various modelling methodologies have already gained a lot of attention, effective means to exploit the developed models are equally crucial for increasing the use of simulation-aided engineering, and deserves more attention. This research is specially focusing at complex maritime problems involving energy systems and operative aspects [1, 2].

2 Aims

Our objective is to use dynamic simulators out of the box without significant model reformulation, harness cloud resources for simulation-based optimization and facilitate the work flow by developing a framework capable of coping with their complexity and exploiting their scalability.

3 Materials and methods

A schematic view of the optimization framework is presented in Figure 1. An optimizer executes dynamic system simulations to evaluate solution candidates. In such simulation-based optimization, the simulations are usually the dominant computational workload. Our framework provides parallel execution of these simulations in the cloud. This benefits optimization methods that are able to exploit parallel evaluation; we use a genetic algorithm [3], which can evaluate each generation of solution candidates in parallel.

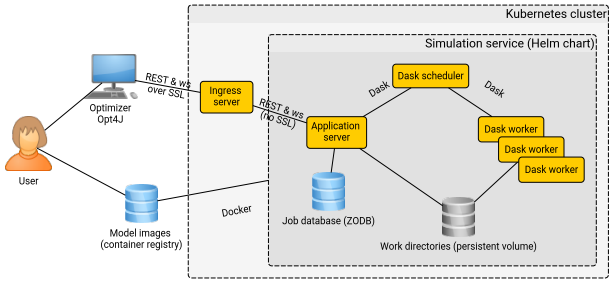

Figure 1. Architecture of the optimization framework.

In our framework the simulations are actually executed by a distributed simulation service. The optimizer posts simulation jobs to the service and retrieves results via a REST and WebSocket interface. Such a strict interface allows the optimizer and the simulation service to run on different hosts. It also allows them to be implemented in different languages: currently we use Opt4J [4], a Java-based open source optimization framework, whereas the simulation service is written in Python. The simulation service makes use of several open source libraries and frameworks: Flask for the web service, Dask for managing distributed computation and ZODB for data storage.

For cloud deployment the simulation server is packaged in a Docker image along with the simulator and whatever model data is required. It can then be installed in a Kubernetes cluster with Helm. It is possible to have multiple instances of the simulation service in the same cluster; each instance is specific to a particular model. An ingress server provides them with a common access point and handles security. The optimizer may execute in the cloud or locally on the user's computer. Its computational load is minimal but a reliable network connection to the simulation service is necessary.

4 Results

The framework has been implemented and initial tests have been conducted with a simple Matlab-based model, which was compiled into a Linux executable with the Matlab Compiler. It was then packaged with the simu-

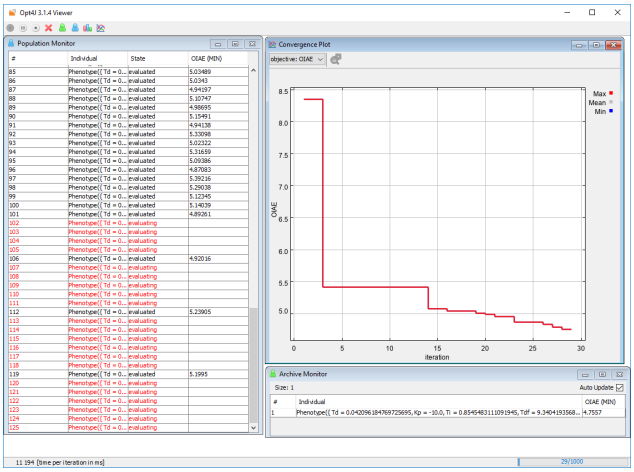

Figure 2. The Opt4J graphical user interface. The graph shows the best objective value by each generation.

lation server and the required Matlab runtime, and deployed to an Azure Kubernetes Service (AKS) instance. The optimizer was executed locally, allowing us to monitor progress with the graphical interface provided by Opt4J (Figure 2).

We initially hoped to run the simulation service on Azure Container Instances (ACI), without managing virtual machines. Virtual Kubelet would dispatch our containers from AKS to ACI. This technology turned out to have significant restrictions and reliability problems. Eventually we had to settle for regular AKS nodes, i.e., virtual machines. That seemed to work fairly reliably but requires scaling the cluster to an appropriate number of nodes before the optimization run and scaling it down to a minimal size afterwards (to avoid paying for unused capacity).

5 Future work

Our test model was so simple that the simulation effort largely consisted of starting the Matlab runtime. The next step is to work with large-scale dynamic simulators. The models origin from different simulation platforms, including Apros and Matlab/Simulink. As the main results so far, we are sharing our experiences on the platform implementation and its performance. A more comprehensive report is presented in [5].

In the later phase of the project, we continue with industrial simulation cases. We want to find out the type and scope of optimization problems that best fit this approach. We also investigate which optimization algorithms promote the optimization with the framework, and how they should be tuned and modified to deliver the best performance in the cloud environment.

References

[1] Lepistö V, Lappalainen J, Sillanpää K, Ahtila P. Dynamic process simulation promotes energy efficient ship design. *Ocean Engineering*. 2016;111:43–55. URL http://www.sciencedirect.com/science/article/pii/S0029801815005909

[2] Zou G, Kinnunen A, Tervo K, Elg M, Tammi K, Kovanen P. Modeling ship energy flow with multi-domain simulation. In: *27th CIMAC World Congress*. Shanghai. 2013; .

[3] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*. 2002;6(2):182–197.

[4] Lukasiewycz M, Glaß M, Reimann F, Teich J. Opt4J - A Modular Framework for Meta-heuristic Optimization. In: *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011)*. Dublin, Ireland. 2011; pp. 1723–1730.

[5] Lappalainen J, Korvola T, Nurminen JK. Cloud-based framework for simulation-based optimization of ship energy systems. In: *Proceedings of MOSES2019, the 2nd international conference on modelling and optimisation of ship energy systems*. Glasgow, UK. 2019; In press.

Acknowledgements

This work belongs to the Task 2.3 in the INTENS research project, jointly funded by Business Finland's Arctic Seas program and the INTENS consortium (Wärtsilä Finland Oy, NAPA Oy, Meyer Turku Oy, Dinex Ecocat Oy, Deltamarin Oy, Vahterus Oy, Protacon Technologies Oy, Parker Hannifin Manufacturing Finland Oy, JTK Power Oy, 3D Studio Blomberg Ab, Jeppo Biogas Ab, Visorc Oy, Tallink Silja Oy, NLC Ferry Ab Oy, Aalto University, Lappeenranta University of Technology, University of Vaasa, Åbo Akademi University and VTT Technical Research Centre of Finland Ltd), which are gratefully acknowledged.

The authors would like to thank Zsolt Homorodi at VTT for assistance with cloud technology.