Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

USING KUBERNETES TO IMPROVE DATA FARMING CAPABILITIES

Falk Stefan Pappert
Daniel Seufferth
Heiderose Stein
Oliver Rose

Department of Computer Science
University of the Bundeswehr Munich
Werner-Heisenberg Weg 39
85577 Neubiberg, GERMANY

ABSTRACT

Simulation can reach computational limits, especially when running large-scale experiments. One possibility to counter this issue is distributed simulation. Recent developments in containerization and container orchestration technologies, such as Kubernetes, provide a stable and scalable infrastructure, that can serve distributed simulation. Although these solutions exist, applications within the simulation community remain scarce. Thus, in this paper we present the general setup of such an infrastructure, and discuss the application on an example case. Adding to the existing literature, we present our path forward and insights with different versions, as well as the efforts needed to construct similar implementations. As a result, we showcase the speed-up of simulation experimentation. We aim to provide a helpful foundation for others in our community to weigh the effort and benefit of such a system for their own projects.

1 INTRODUCTION

There are many topics where simulation data can provide useful information or lay the foundation for modern methods like machine learning. Yet the increasing complexity of models, with various factors and sources of uncertainty (Sanchez et al. 2021), challenges the simulation environment. The workload required for complex simulation puts limits on the feasibility of performing simulation experiments, as the computational effort and time are constrained.

One way to confront the increased need for compute resources, is the usage of meta-models or proxy-models, as detailed in (Bandaru and Ng 2015). This method replaces expensive simulation with inexpensive approximations, which helps accelerating the usage of the simulation model. However, to create and train such models, data is required, which, if not already available, will need to be generated by simulation during resource-intensive data-farming runs. Although, this approach helps during the usage of the model, this mainly shifts the computational load from the usage phase to the development phase of a model, while typically increasing the number of simulations required. Another way to support these types of simulations, is to provide enough computing power to address the increased demand. Especially in data farming and optimization experiments, there is a benefit in distributing simulation workloads onto multiple machines to run in parallel (Lechler et al. 2021).

Amouzgar et al. (2018) give a detailed description of a framework for multi-objective optimization and knowledge discovery of machining process, in which they explain how the underlying simulation model gets distributed to multiple parallel workstations. For this, they utilize a cloud-base Windows script. This is a tailored method for a given use-case and has to be adjusted for other simulation experiments. Therefore, a more general approach would be beneficial. The research on high-performance computing infrastructures

is diverse – and has been growing with the rise of distributed computing and cloud services. For example, Kutzner et al. (2022) evaluated the economic effectiveness of clusters compared to cloud services and showed that cloud solutions became interesting in recent years.

As there are no off-the-shelf solutions readily available, some research papers focus on the creation of self-build solutions. Scalarm and DIRAC for example are two different computing infrastructures that distribute simulation workloads onto multiple computing nodes in heterogeneous environments (Król et al. 2013). Although some remains of Scalarm are still available, the creation of a computing infrastructure based on the public code is difficult.

A different way to compute scalable workloads is to use containerized applications managed inside a Kubernetes-Cluster. Kubernetes was introduced by Google in 2014, to orchestrate containerized applications at scale. From a high level view, a Kubernetes cluster consists of two main parts:

- the control plane, responsible for management and communication, and
- the worker nodes, responsible for running the application containers.

In their book, Poulton and Joglekar (2022) give a good first introduction to Kubernetes' architecture and the various Application Programming Interface (API) objects used for scalable workloads. The Kubernetes documentation (Kubernetes) is another valuable source for starting with Kubernetes.

These recent technological advances have been evaluated by Anagnostou et al. (2019) with their work on simulation experimentation frameworks applying a micro-services auto-scaling approach.

In this paper, we document the changes and efforts necessary to move from a simple simulation model to a heavily distributed version which allows running experiments on large server clusters. As an example, we use a data farming project, that was originally developed for Personal Computers (PCs) and grew with increasing availability of computing hardware. Now it runs in a self-hosted Kubernetes cluster environment. Our goal is to give our readers not only the reasoning for required changes but also a better idea of the effort required on this way, to help inform decisions on whether to move other projects along a similar way.

2 REFERENCE PROJECT

The goal of our simulation project example is to support planners of semiconductor fabs by providing them the maximum equipment utilization possible while still achieving their Key Performance Indicator (KPI) Flow Factor (FF). FF, also known as X-factor (Tsuruta et al. 2006), is commonly used in the semiconductor industry to assess the material flow. It is calculated as the ratio between the cycle time, i.e. the time a job actually stays in the system, and the raw process time, including only the time required to perform all value adding process steps.

$$FF = \frac{cycle\,time}{raw\,process\,time}.$$

Depending on the products manufactured, semiconductor companies typically aim for a FF between two to five. As the FF is dependent on the cycle time it is also dependent on the utilization of the system. Operating curves introduced by Aurand and Miller (1997) are a good way to illustrate this connection. The form and position of these curves depend on the underlying systems. As an example, Figure 1 shows operating curves of two systems. Both systems contain only a single equipment. While the system on the left is a basic system that processes whole lots when they arrive, the second system waits until a number of lots have gathered to be processed together. With a highly different behavior between equipment groups, it is difficult for a planner to decide on the target utilization of the equipment. In our work, we aim to support the planner by providing thresholds at which given FFs are still achievable, based on the configuration and characteristics of the equipment group. To find these thresholds one can simulate different utilization points of the equipment group until the thresholds are found. A problem with this approach is the time needed from the moment the planner needs the information until the required values can be provided. When

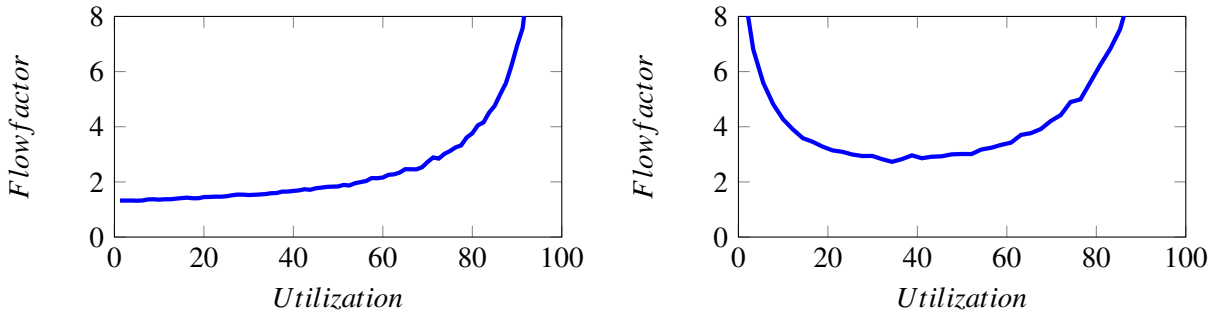

Figure 1: Operating curves of two systems. Both systems represent a single piece of equipment.

considering the stochastic nature of the model and the number of utilization points to be considered even simulations for small equipment groups require a couple of minutes of run time. As these thresholds are needed frequently, there is much time to be saved by reducing the response time of the system. In Pappert et al. (2017), we proposed a combination of data farming and machine learning to tackle this problem. Instead of running simulations for a specific equipment group configuration when the planner needs it, we simulate different possible equipment group configurations in an independent data farming experiment. Then, we use the performance data of these simulation runs to train a neural network. The trained network is then used during the application phase as a surrogate of the simulation model, to respond to the planner and provide utilization thresholds for the specific configuration almost instantly.

As a first step, we considered what would have an influence on FF thresholds. Starting from Robinson et al. (2003) and Hopp and Spearman (1996), we extended literature findings based on discussions with our industry partners and decided on the features and factors. We furthermore defined factor levels for our experimental design based on common values for these characteristics to achieve a representative selection of evaluated data points. A more detailed discussion of our factors is given in Pappert and Rose (2021).

Based on these factors we implemented a generic model of an equipment group, which is shown in Figure 2. Several sources release products based on the product mix given in the scenario. These products are released with lots of 24 wafers each and arrive in the equipment group. Then, a controller handles batching, dispatching, dedication, and setups. It assigns lots or batches to individual machines, when appropriate. Each machine processes the arriving material within a defined raw process time. If a lot or batch needs a different setup state than the previous one on this machine, the equipment will first perform this change and only then start processing. In our model, machines may breakdown or need maintenance from time to time, and will not be available for production then. After the process is finished, the material will be collected, and the original lots will be reformed. Once all wafers of an original lot have arrived, the lot heads on to the rework gate. Here, we decide based on a rework probability whether the lot is finished or needs to undergo another round of processing. If rework is required, we will move the lot back into the queue of the controller. Finished lots will be moved to the sink for data collection.

With this model, different configurations can be evaluated to achieve certain flow factors. For this purpose, we look for the utilization point with the lowest flow factor and from there start searching for the thresholds. As we are using a stochastic model, we simulate each utilization point several times. We start with five replications and dynamically decide whether enough simulation runs have been done based on the already achieved confidence interval comparable to Law and Kelton (2004). With higher utilization, the variance contributed by more factors becomes visible and we see a significant increase in the necessary simulation runs. In total, for all evaluated utilization points and replications, we run a single scenario on average about 850 times to find our points of interest.

A big challenge in our work is the sheer number of simulation runs to be done. First, we need a large number of simulation runs to properly evaluate a single scenario, i.e. equipment configuration. Second,

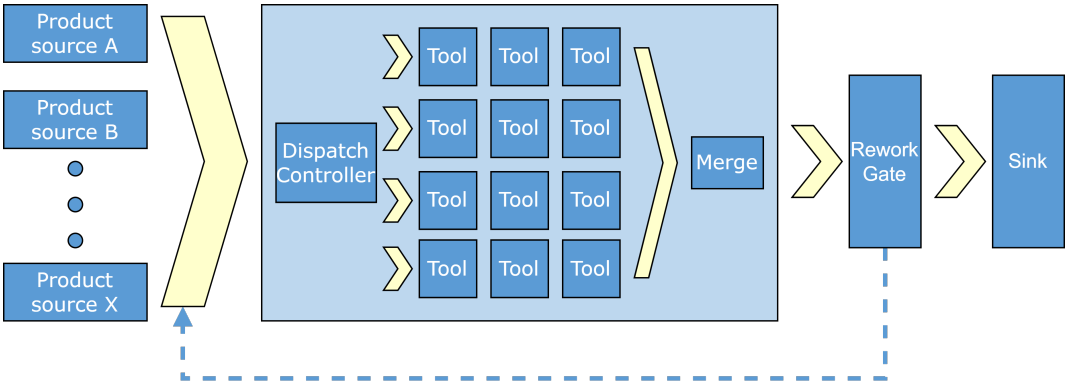

Figure 2: Overview of the simulation model.

in order to provide correct responses for a broad number of possible equipment group configurations, the number of scenarios we intended to simulate was significant from the start and has grown since then.

3 HISTORY OF THE ARCHITECTURE

In this section, we will give a brief history of our data farming implementations from the ground up. We start with the initial prototype and move through different architectures that are based on the hardware being available to us during different phases of the project. We will point out the change in scenarios run per day between versions. The performance gains we show can be mainly attributed to the increased hardware available. The figures shown throughout this section show the progress and change of the architecture. In blue we show parts of the architecture that is continued to be used from previous iterations and green indicates the changes done.

3.1 Simple Monolith

The initial version of data farming we implemented was created as a monolith (shown in Figure 3). The focus was to set up all components required to automatically run models in a tracer bullet approach (Hunt and Thomas 2003). Therefore the components required to evaluate a single scenario, i.e. model generation, release calibration, data collection, and the search for the utilization thresholds, needed to be developed. The goal was to be able to validate the automatically created simulation models and their results with industry colleagues. The scenario management was implemented as a slim shell around the previously mentioned components and targeted at running on a typical PC. This came with the following limitations: Simulation runs for a single scenario were parallelized, but different scenarios needed to be run sequentially. The factors and their levels were limited and hard-coded into the system. Results had been written to text files.

The implementation effort just for the scenario management part is comparable to about 1-2 working days, which resulted in a single office PC being able to evaluate between 20 and 50 scenarios per day, depending on the scenarios evaluated. When looking at the number of scenarios we intended to run, running all scenarios within the time frame of our project was infeasible, but we were able to provide sufficient amounts of data for validation.

3.2 Monolith on Large Virtual Machine

The next major step was to allow for evaluating several scenarios at the same time. This was done using a Virtual Machine (VM) with a large computational capacity of about 40 cores (see Figure 4). We usually run five replications in parallel for a single scenario, and based on our search strategy evaluating more than one utilization point at a time may be a waste of computing power. Running more replications in

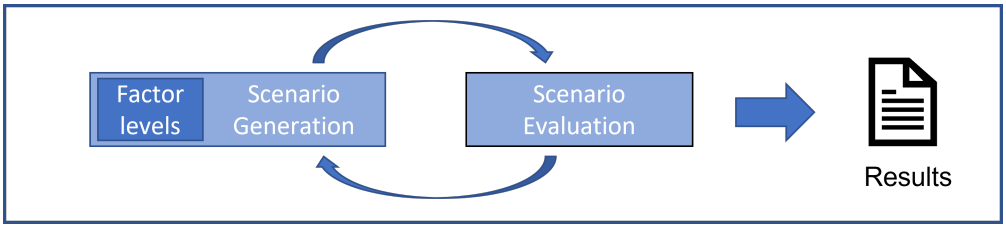

Figure 3: Initial approach of data farming as a monolith.

parallel for a single utilization point can also be quite wasteful, as a large number of utilization points can be estimated with five replications quite well. This led us to parallelizing scenario evaluations to utilize the available hardware.

A limiting factor for us, with regard to this VM, was memory usage. An initial approach was putting all scenarios in a *Collection* and running them using Java's basic *foreach*-loop. This led to memory issues frequently. On detailed analysis, we found this construct to have some disadvantages when multiple processes are parallelized: As each scenario evaluation ran up to five threads, Central Processing Units (CPUs) were rather contested. This led to scenarios being abandoned for some time while still having a significant memory footprint. To combat this, we needed to reduce the number of scenarios running in parallel by running batches of scenarios. An initial idea to reduce the number of parallel runs was running batches of scenarios, each provided with a certain number of cores for performing calculations. As evaluation time depends on a scenario's size and the variance of results causing us to increase the number of replications, this led to very busy CPUs when a new batch is started, and idle CPUs towards the end of batch processing when only the last scenario of the batch is processed. Simply limiting the number of parallel processed scenarios to handle this was no option because some combinations of parallelized scenario runs could lead to memory issues due to their size and the amounts of collected data. Therefore, we analyzed our factors and built overloaded batches based on the factors which have the highest impact on memory consumption. These overloaded batches encompassed slightly more scenarios as can in theory be run in parallel but did not yet run into memory issues. Thereby, scenarios with small and large memory footprints are grouped for overall average memory usage. We overload the batches, to include more than eight scenarios, to reduce idle CPUs at the end of a batch's run. The amount of work needed for this change ranged between one and two weeks, resulting in an increase to 80-300 scenarios evaluated per day.

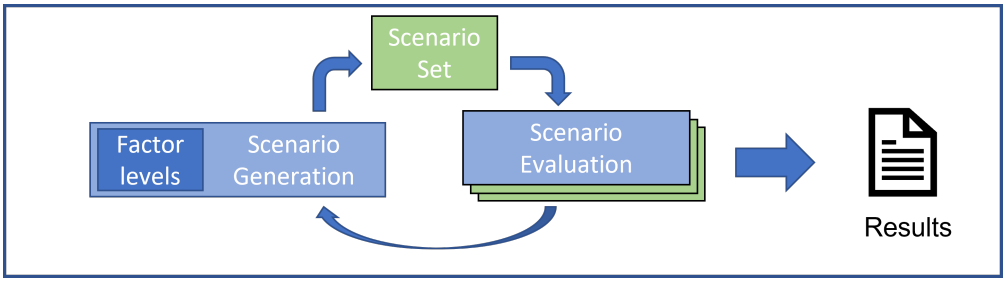

Figure 4: Version of data farming as a monolith on a virtual machine.

3.3 Distributed Runners on Multiple VMs

To harness more computing power, and therefore be able to parallelize more runs, as a next step we moved to several VMs. Leaving the realm of a single VM required not only changes to the system running the scenarios but also to scenario generation and data storage. While up to this point factors and the resulting scenarios could be generated on the fly during evaluation, this is difficult to handle when running on several VMs in parallel. The goal of this step was to have several identical versions similar to the single-VM version

run in parallel, with one centralized system handling the distribution of scenarios to different VMs and collecting result data of all runners for further processing (see Figure 5). The main change was therefore to move from hard-coded factors and results in text files to a system where both are stored within a database. As discussed in Pappert and Rose (2022), we changed the way we look at factors. To allow for additional changes, we implemented a very flexible database scheme, which is automatically generated based on the factors used and therefore can adapt to future changes. It took about three weeks for this implementation and allowed us to run about 200-500 scenarios per day.

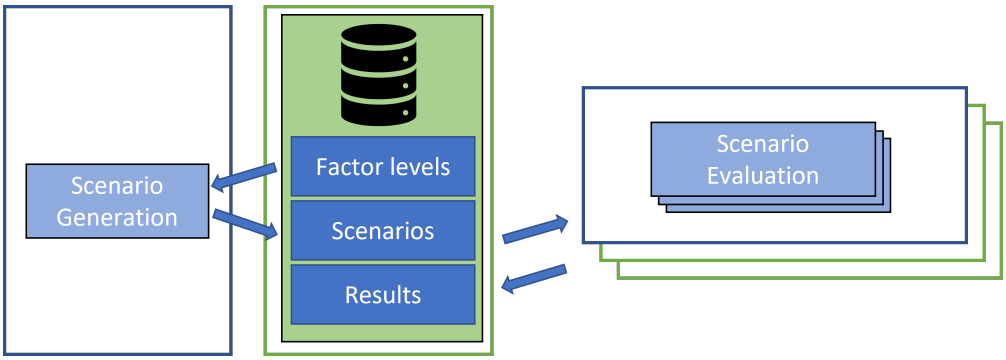

Figure 5: Version of data farming as a group of monoliths on several virtual machines.

3.4 Distributed Runners Inside Pods

An issue with the aforementioned approach is the need to calibrate batch sizes for each VM to be used individually. This makes a larger rollout tedious, especially when the availability of computing resources for the used VMs change based on the load caused by other projects. Therefore, we considered another significant redesign, moving from VMs to containerized simulation runs in a container orchestration environment, as shown in Figure 6. Moving the responsibility of load allocation away from our software to a container orchestration tool has proven a good step towards scaling our data farming flexibly. With the simulation software being an in-house Java development, we could containerize all of our software within about two to three days time. For off-the-shelf simulation packages, the required time will depend heavily on the software used. Two main concerns when containerizing software are first, the required operating system, where software naively running in Linux is usually easier to package, and second, licensing. With our own software or common open-source software, there is usually no additional effort or artificial constraint due to the number of parallel instances running or even to access hardware keys needed for running an instance of the simulator.

When running these pods on our old VMs, in a Test-Cluster, we already had a little bit of speed gain with 250-550 scenarios per day, most likely due to avoided idle time at the end of batches. In our current Mini-Cluster environment using four hosts as worker nodes, we are running 30 pods resulting in about 800-2000 scenarios being evaluated per day. Scaling the number of pods would now only be a matter of a single command, to utilize most of the 40 hosts available. With respect to some hosts being used for administrative purposes, we expect the final version to run with about 275 parallel pods. Our current estimate, for this configuration, places the number of possible scenarios evaluated per day at about 7500-18500 per day.

4 CONTAINERIZATION OF THE MODEL

With our change in architecture towards using a container-based approach, we can run our data-farming experiment inside our Kubernetes cluster. Therefore, we first have to containerize our simulation model. Hence, we need to answer the following questions:

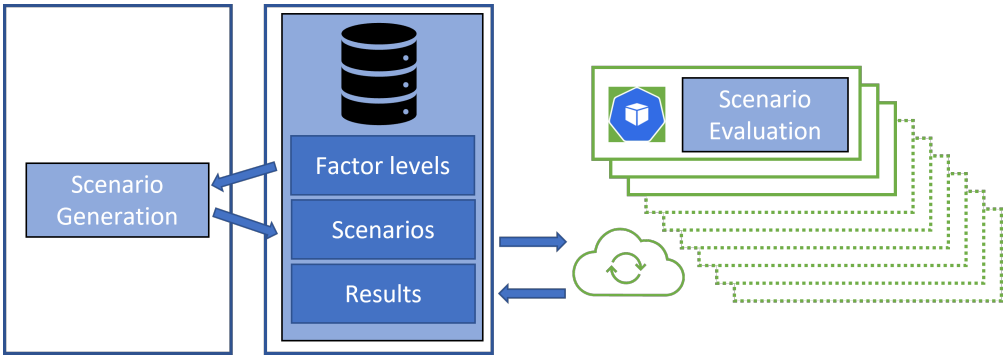

Figure 6: Containerized simulation runs in a container orchestration environment.

- What type of container engine do we want to use?
- How do we want to store our container images?
- How do we want to provision the images to our Kubernetes cluster?

The container engine delivers a runtime for container execution and command line tools for pulling existing images from an image store, creating new images, executing images on our PC, etc. In addition to enabling us to create a simulation container, the decision for one container engine also influences the cluster creation process. Not every container created with one container engine runs on a different container engine. As every Kubernetes cluster needs a Container Runtime Interface (CRI) to be able to execute containers, the CRI of our cluster must be compatible with the container engine used for creating our simulation containers. After the creation of a container image, it gets stored locally by default. As Kubernetes needs access to the container images, it would be necessary to enable communication between the cluster and the local container development machine. This is feasible, but comes with an unnecessary configuration overhead, as there are different applications for securely storing container images with native support for Kubernetes.

4.1 Choosing a Container Engine

There are many different container engines available to choose from. For this first simulation experiment, we examined two often used container engines: *Docker* and *Apptainer*. The Docker engine is one of the most popular container engines. It was released in 2013 by Docker Inc. and can be considered the first real container engine, as stated by Hitchcock (2022). The standalone engine can be installed on every mainstream Linux distribution. For development on Windows and macOS, Docker Inc. offers a desktop application that delivers all functionalities of the engine and a lightweight Linux VM, providing the Kernel needed to create Linux containers. This desktop version of the Docker engine can be installed in a matter of minutes. The second containerization tool we looked at is Apptainer. Initially developed as Singularity at Lawrence Berkely National Laboratory, the project was moved to the Linux Foundation in November 2021 and was renamed Apptainer (Apptainer Project 2023). Providing containerization technology that supports existing and traditional high-performance computing resources is one main focus of the Apptainer project as stated in Kurtzer et al. (2017). In Table 1 we compare Docker engine and Apptainer, to help decide on one container engine. One drawback of Apptainer is in the larger image sizes, due to packaging the complete Operating Systems (OSs) in the container. This makes the containers created with Apptainer not only less portable but adds to the complexity of creating containers. Furthermore, Apptainer does not provide an application that makes containerization possible on Windows or macOS. To use Apptainer on those OSs, other tools have to be installed and set up manually. We, therefore, decided on using Docker engine for this project, keeping Apptainer in mind, if we have workloads that benefit from Graphics Processing Unit (GPU) support.

Table 1: Comparison between Apptainer and Docker containers.

Feature	Apptainer	Docker
Use Case	HPC workloads	general-purpose
Image size	large (entire OS)	small
Image Building	complex	simple
Native GPU support	yes	no

4.2 Storing Container Images

As mentioned above, container images are getting stored locally by default, either on the VM (when using a setup with Windows or macOS) or in the local file system of the Linux distribution in use. Storing container images only locally has considerable drawbacks, i.e., reduced portability of containers, challenging data recovery, etc. The common approach to storing them is therefore using an online repository. This can either be hosted on self-owned hardware or be supplied by a service provider. For us, the decision between one of those two concepts ultimately came down to the two following points: data security and cost. As simulation models, especially high-fidelity models used as digital twins, contain proprietary logic or data that is considered confidential, storing them in some online repository is usually not an option. Additionally, most hosted repositories come with recurring license/subscription fees making them less appealing within academia. Therefore we decided on using a self-hosted container image repository for storing our simulation containers.

Similarly to the container engines, there is a wide amount of self-hosted container repository providers available to choose from. The Cloud Native Landscape (2023) gives a good overview of the current container repository projects available. Choosing one of the available container repository projects is mostly a question of personal preference. At the time of writing, the only container repository project in the Cloud Native Landscape that is considered graduated, and recommended for industrial use, is Harbor, hence we decided on using it for our self-hosted container repository. As a starting point, an instance of Harbor as a container inside a VM on Docker can be set up in about three days.

4.3 Creating the Simulation Container

To create a container image with Docker, a text file, commonly referred to as Dockerfile, is needed. It defines all steps that have to be executed, to create a working container image. For our particular development environment, where we use Maven as an automatic build tool, a plugin called fabric8 is available that lets us integrate the containerization process into the build process. This is done by defining the containerization steps directly in Maven. In addition to this, fabric8 also allows us to integrate our self-hosted repository in the build process, completely automating the containerization process. The steps we used for containerizing our model can be boiled down to the following three:

- defining the base image our simulation containers is using,
- copying the compiled model in the container file system, and
- providing the command to start the simulation model.

As our model is written in Java, we use the OpenJDK-image, which provides the Java Runtime Environment (JRE) necessary for executing our model. Setting up our automated containerization pipeline took approximately two days.

5 CREATING A KUBERNETES CLUSTER

There are multiple ways for creating Kubernetes clusters, which can be put into two categories:

1. Creating unmanaged clusters.

2. Creating managed clusters with tools like Tanzu or Rancher.

The creation of unmanaged clusters is feasible for small cluster sizes, that connects up to five PCs, as the process is predominantly manual. Creating these smaller unmanaged clusters can be done quickly, with an initial set-up time of around three days. Furthermore, production-grade features like High-Availability (HA), are tied to complex manual configurations, adding to an increase in set-up time. Therefore, for larger clusters with production-grade features, managed clusters are better suited. They require more initial set-up time, due to the complexity of the tools needed, but can easily be scaled up and down. Moreover, the creation of additional clusters is quick, hence we decided to use managed clusters created with Rancher. Rancher is a common open-source choice for creating managed Kubernetes clusters and is distributed as a containerized application (Rancher Labs 2023).

A quick way to use Rancher is running it as a container on a PC or a VM. This can be done in a matter of minutes with one command in Docker engine. The running Rancher container hosts a web application, that guides you through the cluster creation process via a graphical interface. As we create our Kubernetes cluster on top of VMs, using vSphere as our virtualization software, we had to configure Rancher so it can automatically create VMs and register them to Kubernetes. This took us quite some time due to complex documentation. Accordingly, we needed four weeks to create the first cluster with Rancher. After becoming familiar with Ranchers GUI-driven cluster creation process, we created our simulation cluster, which consists of three VMs for managing workloads and four VMs for simulation execution. This cluster runs on a flavor of Kubernetes called RKE2, also developed by Rancher. RKE2 comes with "containerd" as a CRI, which is compatible with our simulation container created with Docker engine. The four worker VMs are consuming 90% of the physical hosts CPU resources they are running on. In Figure 7 the structure of our simulation cluster, with detailed information regarding allocated resources, is visualized.

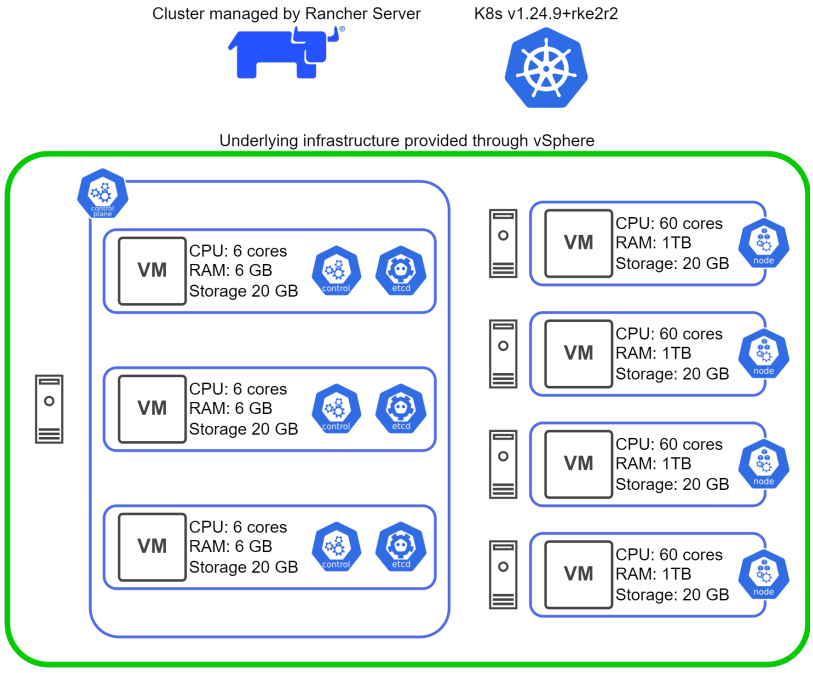

Figure 7: Schematic structure of our simulation cluster.

6 RUNNING AN EXPERIMENT

So far, we put our application into a container using Docker engine, pushed it to our self-hosted Harbor container repository, and created our Kubernetes cluster. To run our experiment on the cluster, we need

to package the simulation container in a scalable Kubernetes API object, as Kubernetes does not run containers directly. Instead, Kubernetes provides the pod API object, that wraps around the container and adds features (labeling, port selection, etc.) to it. Since pods themselves have no scaling functionalities, Kubernetes provides an additional API object called deployment. A deployment works like a controller, and manages the number of pods executed on the cluster. This enables scaling and ensures pod-health. The setup of our experiment is visualized in Figure 8.

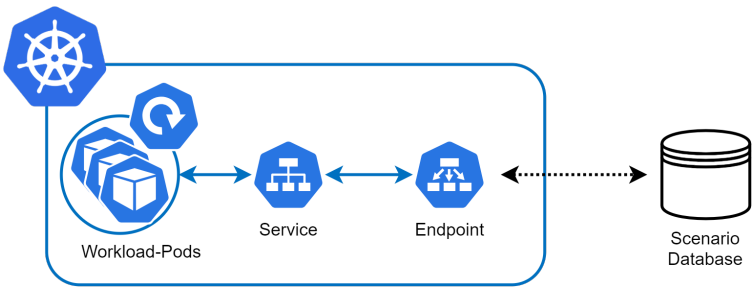

Figure 8: Schematic setup of our data farming experiment running inside of our cluster.

7 CONCLUSIONS AND FURTHER RESEARCH

As we stated in the sections above, installing all necessary parts for containerizing simulation models and running them in Kubernetes takes different amounts of time, depending on the maturity of the setup. We summarize the time we dedicated to the different steps in Table 2, with the optional steps for creating a containerization pipeline and container image repository on the bottom of the table.

Table 2: Approximate time required for each parallelization step.

Step	description	time required
PC	simple monolith	2d
Single VM	monolith on large VM	2w
Multi VM	distributed runners on VMs	3w
Test-Cluster	initial setup	4w
Mini-Cluster	current setup described	1d
Full-Cluster estimate	adding more resources and production-grade features	2w
Optional steps		
Containerization pipeline	installing Docker engine and setting up Maven	3d
Container repository	running Harbor as a container	3d

Our research shows that Kubernetes benefits the execution of data-farming experiments significantly, as visualized in Figure 9. Nonetheless, setting up a containerization pipeline and a production-grade managed cluster is not the go-to solution for every use case. Depending on the hardware available and the number of models running in parallel, smaller unmanaged clusters can be an alternative that can be set up quickly in less than two weeks. Furthermore, the cluster created for one simulation experiment can also be used to execute another containerized model, provided the container engine is supported by one of the CRIs of the cluster.

The next step for us is developing a web application with a GUI, that functions as the central entry point to our simulation cluster. In this way we hope to reduce the barrier of entrance for colleagues to use our architecture for their experiments without deeper knowledge of container orchestration. We also aim to make both the cluster as well as the future web application reproducible, so that anyone interested in running

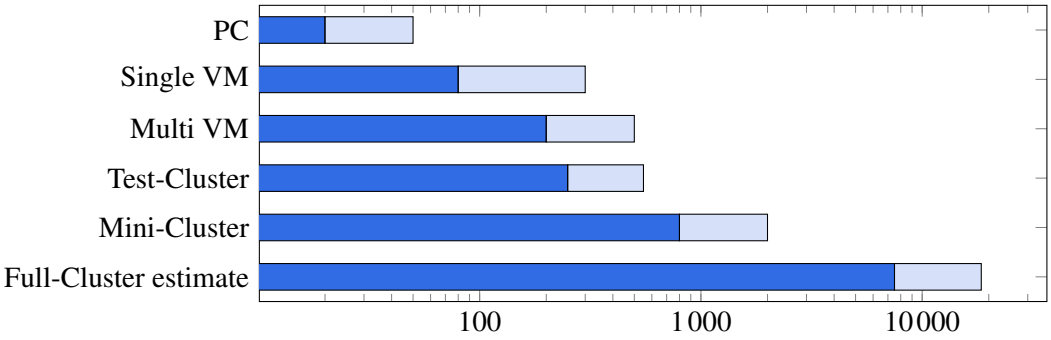

Figure 9: Number of evaluated scenarios per parallization step.

a similar setup can use our solution. Although we currently are very fortunate to have gotten new powerful server hardware, this is not necessarily an absolute requirement of setting up a similar infrastructure. With office and computing pool PCs usually idling during nights and weekends there is much computing power unharnessed in many organizations, which could be combined to form a flexible and powerful computing infrastructure.

ACKNOWLEDGMENTS

We want to thank Uwe Langer and Alexandros Karagkasidis for their continuous support on the hardware infrastructure. This research is funded by dtec.bw - Center for Digitization and Technology Research of the Bundeswehr. dtec.bw is funded by the European Union -NextGenerationEU.

REFERENCES

Amouzgar, K., S. Bandaru, T. Andersson, and A. H. C. Ng. 2018. "A Framework for Simulation-Based Multi-Objective Optimization and Knowledge Discovery of Machining Process". *The International Journal of Advanced Manufacturing Technology* 98(9-12):2469–2486.

Anagnostou, A., S. J. E. Taylor, N. T. Abubakar, T. Kiss, J. DesLauriers, G. Gesmier, G. Terstyanszky, P. Kacsuk, and J. Kovacs. 2019. "Towards a Deadline-Based Simulation Experimentation Framework Using Micro-Services Auto-Scaling Approach". In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K. G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, 2749–2758. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Apptainer Project 2023. "Documentation | Apptainer". https://apptainer.org/docs/, accessed 11th May 2023.

Aurand, S. S., and P. J. Miller. 1997. "The Operating Curve: A Method to Measure and Benchmark Manufacturing Line Productivity". In *Proceedings of the 1997 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, edited by D. T. Fletcher, S. R. McClure, and J. Goodman, 391–397. Piscataway, New Jersey: Institue of Electrical and Electronics Engineers, Inc.

Bandaru, S., and A. H. Ng. 2015. "On the Scalability of Meta-Models in Simulation-Based Optimization of Production Systems". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, V. W. K. Chan, I.-C. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 3644–3655. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Cloud Native Landscape 2023. "Cloud Native Landscape - Container-Registry". https://landscape.cncf.io/card-mode?category=container-registry&grouping=category, accessed 27th March 2023.

Hitchcock, K. 2022. *Linux System Administration for the 2020s*. 1st ed. Berkely: Apress.

Hopp, W. J., and M. L. Spearman. 1996. *Factory Physics*. 3rd ed. New York: McGraw-Hill.

Hunt, A., and D. Thomas. 2003. *Der Pragmatische Programmierer*. München: Hanser.

Król, D., M. Wrzeszcz, B. Kryza, Ł. Dutka, and J. Kitowski. 2013. "Massively Scalable Platform for Data Farming Supporting Heterogeneous Infrastructure". In *The 4th International Conference on Cloud Computing, Grids, and Virtualization*, edited by W. Zimmermann, 144–149. Wilmington: IARIA.

Kubernetes. "Kubernetes Documentation". https://kubernetes.io/docs/home/, accessed 15th June 2023.

Kurtzer, G. M., V. Sochat, and M. W. Bauer. 2017. "Singularity: Scientific Containers for Mobility of Compute". *Plos One* 12(5):e0177459.

Kutzner, C., C. Kniep, A. Cherian, L. Nordstrom, H. Grubmüller, B. L. de Groot, and V. Gapsys. 2022. "GROMACS in the Cloud: A Global Supercomputer to Speed Up Alchemical Drug Design". *Journal of Chemical Information and Modeling* 62(7):1691–1711.

Law, A. M., and W. D. Kelton. 2004. *Simulation Modeling and Analysis*. 3. internat. ed. New York: McGraw-Hill.

Lechler, T., M. Sjarov, and J. Franke. 2021. "Data Farming in Production Systems - A Review on Potentials, Challenges and Exemplary Applications". *Procedia CIRP* 96:230–235.

Pappert, F. S., and O. Rose. 2021. "Reducing Response Time with Data Farming and Machine Learning". *SNE Simulation Notes Europe* 31(2):87–94.

Pappert, F. S., and O. Rose. 2022. "Using Data Farming and Machine Learning to Reduce Response Time for the User". In *Proceedings of the 2022 Winter Simulation Conference*, edited by B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C. G. Corlu, L. H. Lee, E. P. Chew, T. Roeder, and P. Lendermann, 1707–1718. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pappert, F. S., O. Rose, F. Suhrke, and J. Mager. 2017. "Simulation Based Approach to Calculate Utilization Limits in Opto Semiconductor Frontends". In *Proceedings of the 2017 Winter Simulation Conference*, edited by V. W. K. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. H. Page, 3888–3898. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Poulton, N., and P. Joglekar. 2022. *The Kubernetes Book*. 2022 ed. self published: Nigel Poulton.

Rancher Labs 2023. "Rancher Homepage". https://www.rancher.com/, accessed 27th May 2023.

Robinson, J., J. Fowler, and E. Neacy. 2003. "Capacity Loss Factors in Semiconductor Manufacturing". https://fabtime.com/files/CapPlan.pdf, accessed 27th June 2023.

Sanchez, S. M., P. J. Sanchez, and H. Wan. 2021. "Work Smarter, Not Harder: A Tutorial On Designing and Conducting Simulation Experiments". In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, 1–15. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Tsuruta, S., K. Kabata, H. Kawakami, M. Kishimura, S. Yamaguchi, and T. Ogawa. 2006. "Study of the Relation Between Loading Ratio and X-Factor". In *Proceedings of the 2006 IEEE International Symposium on Semiconductor Manufacturing*, 296–298. Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

FALK STEFAN PAPPERT is a research assistant and PhD student at the Chair of Modeling and Simulation of the University of the Bundeswehr Munich. His focus is on using data farming to support the development of proxy models for production systems to be used in planning, and short term decision making. He is a member of GI. He has received his M.S. degree in Computer Science from Dresden University of Technology. His email address is falk.pappert@unibw.de. ORCID: 0009-0006-6541-3924

DANIEL SEUFFERTH is research assistant and PhD student at Universität der Bundeswehr München as a member of the scientific staff of Prof. Dr. Oliver Rose at the Chair of Modeling and Simulation. His focus is on the usage of virtualization and containerization for parallizing simulation execution. He has received his M.Eng. degree in Mechanical Engineering from Coburg University of Applied Sciences and is a member of VDI. His email address is daniel.seufferth@unibw.de. ORCID: 0009-0000-2831-7761

HEIDEROSE STEIN is a research assistant and PhD student at the Chair of Modeling and Simulation of the University of the Bundeswehr Munich.Her research interests include how to use simulation to solve decision problems, in techniques to efficiently search or reduce the solution space or how to speed up data farming experimentation, and transferring knowledge of different domains. She holds a M.S. degree in Chemical Engineering from the Technical University of Munich and is a member of ASIM. Her email address is heiderose.stein@unibw.de.

OLIVER ROSE holds the Chair for Modeling and Simulation at the Department of Computer Science of the University of the Bundeswehr Munich, Germany. He received a M.S. degree in applied mathematics and a Ph.D. degree in computer science from Wurzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities and supply chains. He is an experienced teacher of modeling and simulation, as well as the mathematical and practical background, and shares his experiences for example as a member of INFORMS Simulation Society, ASIM, and GI. His email address is oliver.rose@unibw.de. ORCID: 0000-0003-3365-1611