2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) | 979-8-3503-0037-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-SEIP58684.2023.00041

Hybrid Cloudification of Legacy Software for Efficient Simulation of Gas Turbine Designs

Fozail Ahmad*, Maruthi Rangappa*, Neeraj Katiyar*, Martin Staniszewski† and Dániel Varró*‡
*Department of Electrical and Computer Engineering, McGill University, Montreal, Canada
Email: {fozail.ahmad, maruthi.rangappa, neeraj.katiyar}@mail.mcgill.ca, daniel.varro@mcgill.ca
†Aeroderivative Gas Turbines, Siemens Energy, Montreal, Canada
Email: martin.staniszewski@siemens-energy.com
‡Department of Computer and Information Science, Linköping University, Sweden

***Abstract*—When developing aeroderivative gas turbines at Siemens Energy, engine models are subject to complex simulation campaigns for finite element analysis carried out by a legacy simulation tool. This paper presents results of a multi-year software modernization project to provide a software-as-a-service (SaaS) framework that enables the distributed and automated execution of simulation jobs over a hybrid cloud platform containing both private cloud and public cloud nodes. Our framework allows to significantly reduce the net time required for completing complex simulation campaigns, thus increasing the effectiveness of engineers. The performance of our framework is evaluated in various cloud configurations with complex simulation campaigns performed in the context of a real simulation task.**

***Index Terms*—cloud computing, software modernization, automation, simulation as a service**

I. Introduction

Industrial relevance: The design of aeroderivative gas turbines (AGTs) is a complex multi-disciplinary engineering process, which involves various domain experts. Improving the performance of AGTs is an iterative process spread over many years, where new AGT models are repeatedly designed and analyzed to determine if their performance has improved.

Since building physical prototypes is very costly, a variety of software tools are used to develop AGT engine models as digital twins which are then analyzed by simulation to assess and optimize mechanical performance [1], [2], [3], [4], [5], [6], [7], [8]. As a critical design step, *finite element analysis* (FEA) is performed by *large-scale simulation* campaigns on thermomechanical AGT models to validate new designs against performance requirements and objectives [9], [10], [11].

Existing industrial practice at Siemens Energy uses a legacy software tool for FEA deployed on desktop computers of engineers. For a complex simulation campaign (covering hundreds of analysis points), engineers need to manually invoke the tool to run the simulation of each analysis point. The analysis runtime for a single analysis point takes over 2-12 hours depending on model granularity and the type of FEA.

Objectives: This paper summarizes results of a multi-year software modernization project that enables the automated execution of entire simulation campaigns over a hybrid cloud platform while using the existing legacy FEA tool. *Simulation Software as a Service* (SaaS) is a hybrid cloud based framework which can flexibly provision computing nodes with automated job scheduling from on-premise servers (private cloud) as well as from Amazon Web Services (AWS) resources (public cloud). *Simulation SaaS* is available for front-end design tools and analysis scripts via RESTful APIs.

Contributions: Compared to past engineering practice and tools at at Siemens Energy for FEA in AGT design, the specific innovations reported in the paper include the following:

- We migrate a legacy FEA simulation software into a container architecture to enable machine-agnostic execution.
- We integrate cloud-based object storage to reduce data transmission and handle input/output artifacts for simulation.
- We provide a software-as-a-service solution to access the cloud-based simulation software over the internet from various frontends without installation needs.
- We offer a hybrid cloud architecture and software service for automated distributed execution of simulation campaigns.
- We carry out an extensive experimental evaluation of the performance of the proposed architecture in the context of a complex simulation task provided by Siemens engineers.

Benefits: The main business added value that the *Simulation SaaS* framework provides is significant time reductions on multiple levels. As the primary effect, the speed-up gained by distributed execution enables the completion of complex simulation campaigns in significantly less amount of time, thus flaws are identified and corrected earlier by engineers. As a secondary effect of our SaaS solution, no time is spent on installing the legacy FEA simulation tool on individual desktop computers of engineers. Finally, our SaaS solution also enables the seamless integration of FEA simulation into complex engineering tool chains used by various design teams.

More efficient FEA by the *Simulation SaaS* enables better utilization of engineers' time and workload. Since the hourly rate of engineers is significantly higher than the hourly rate of cloud computing resources, the increased costs required for provisioning AWS resources are counterbalanced by efficiency gains. Moreover, with automated execution of FEA tasks, engineers can investigate more analysis points, resulting in higher performing and sustainable AGT products. Thanks to its flexible deployment capabilities, the use of *Simulation SaaS* can be limited to certain geographical regions or user groups in order to comply with strict export control regulations.

2832-7659/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-SEIP58684.2023.00041

II. Motivation and Industrial Context

Aeroderivative gas turbines (AGTs) are mechanical power generation products built using aircraft turbine engines. Aircraft turbine engines are designed to ramp up and down very quickly for effective flight control which enables AGTs to dynamically change their power output based on demand variations in the power grid. The ability of AGTs to rapidly adapt to grid conditions makes them well suited to efficiently fill demand peaks in the electrical grid as traditional power generation technologies, such as hydroelectric dams, cannot be brought online or offline as quickly. Siemens Energy manufacturers AGTs of various capacity to offer a flexible option alongside their traditional offerings of large gas turbines.

A. Finite Element Analysis for AGTs

To avoid building costly physical prototypes whenever possible, an AGT engine model is built as a digital twin of a real AGT engine. It contains various components such as blades and shafts which are then combined into configurations to create sections of the AGT such as the compressor, combustion chamber and turbine. AGTs have specifications for operating condition ranges that need to be met such as intake air pressure and ambient temperature. In order to evaluate and improve the performance of AGTs, *finite element analysis* (FEA) has to be performed under varying operating conditions on different engine models with unique and experimental component designs and placement.

The iterative process of improving an AGT necessitates performing FEA multiple times in complex *simulation campaigns* as the performance of an AGT has to be evaluated for various engine models and operating conditions to ensure robust design. A *simulation job* is defined as one configuration of an AGT engine model under a specific operating condition for which FEA has to be performed using a simulation tool to evaluate its performance. When analyzing a real AGT engine at Siemens Energy, hundreds of different operating conditions need to be evaluated by engineers via simulation.

B. Legacy Simulation Tool

Mechanical engineers at Siemens Energy have been using a legacy (in-house) simulation software tool for FEA of AGT designs for several decades (see Figure 1). This legacy simulation tool is preferred by engineers as they have a clear understanding of the capabilities and limitations of the tool, which helps them provide faithful and reliable estimates of real engine parameters by extensive simulation campaigns.

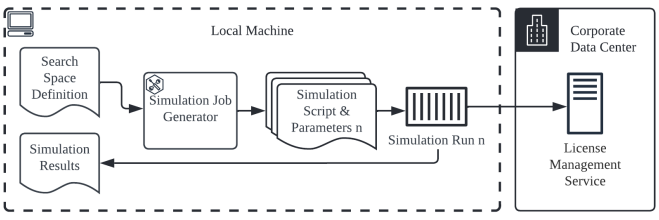

Fig. 1: Existing engineering practice for simulation based FEA

In existing practice, the legacy simulation tool has been used by engineers in accordance with the following workflow:

1) Engineers provide the *search space definition* in a local file by listing the various AGT engine models and operating conditions for FEA to be carried out by simulation. Engineers need to ensure that (a) all engine models are available locally on their desktop computer, and (b) the legacy simulation tool is properly installed.
2) Then the search space definition file, engine model locations, and legacy software location are passed as inputs to the *simulation tool launcher*. The simulation tool launcher will then create all the possible combinations of the engine models and operating conditions (i.e. analysis points) and programmatically generate a *simulation campaign*.
3) For executing each *simulation job* in the campaign (i.e. for a particular engine model and operating condition), the engineer needs to manually invoke the simulation tool to carry out FEA for this analysis point. While some engineers may partially automate this step by developing ad hoc scripts, no robust and fully automated solution has been available, and constant supervision of the simulation is required by the engineers.
4) The simulation runtime takes approximately 2-12 hours depending on the complexity of the model (e.g. 2D vs 3D) and the analysis (e.g. temperature vs. vibrations). If the execution of a simulation job has successfully completed, the results are available locally on the machine of the engineer. If the simulation job fails, the engineer needs to manually re-run the simulation tool for the designated analysis point. For certification and quality assurance of key engine models, the engineer needs to manually upload the simulation results to the product lifecycle management tool used by Siemens Energy.

Technological constraints:

The legacy simulation tool was designed for a *single computer deployment*, and requires *direct access to a license management server*. The simulation tool is an executable that can be invoked on any computer running the Windows operating system, and requires certain (3rd party) libraries to be installed locally. However, such local deployments do not natively provide any mechanisms for scaling horizontally with a dynamic pool of host computers.

Moreover, on startup, the simulation software needs to access a proprietary license management server as required by the original FEA tool vendor. The license management service is deployed on premise at Siemens Energy in a secure network to restrict its usage to authorized users. Therefore, any computer running the software must also have access to the secure network where the license server is available.

Finally, the code base of the legacy simulation tool is not managed by Siemens Energy, and no changes can be requested or made to the original software.

Process and business constraints: Using a legacy simulation tool that is no longer upgraded in the context of ongoing initiatives at Siemens Energy such as digitalization and increased use of machine learning initiatives has its own

drawbacks. The high simulation runtimes and the high degree of manual supervision required currently limits FEA to stable engine designs. As a consequence, engineers simply lack time to run FEA simulations for novel, unconventional engine designs. Moreover, the existing engineering practice is unable to generate sufficient amount of FEA simulation data to train surrogate machine learning models (unlike in case of [12]).

Finally, AGT design needs to *comply with export control regulations* imposed by various countries. Thus one needs to guarantee that (i) only restricted personnel has access to certain engine designs, and (ii) data required for FEA simulations continuously stays withing a certain geographic region.

C. Objectives

Our research collaboration with Siemens Energy aimed to address the following objectives as part of a software modernization work package in a 5-year multi-disciplinary project on design optimization.

O1 *Simulation software as a service*. Provide seamless access to the legacy FEA tool in the form of a software as a service (SaaS) solution from various front-ends (GUI, Python programs) via standard interfaces without any further local installation needs.

O2 *Automated execution of simulation campaigns*. Enable the automated execution of large simulation campaigns without any invocation or supervision of individual simulation jobs by human experts.

O3 *Decrease total simulation time* required for complex simulation campaigns by allowing the parallel computation of simulation jobs (horizontal scaling).

O4 *Distributed execution over hybrid cloud platform*. Enable the seamless distributed execution of simulation jobs over a hybrid cloud platform with a configurable mix of both on-premise servers and public cloud servers on AWS.

O5 *Geographically restricted simulation data and execution*. Ensure compliance with export control regulations of intellectual property of designated countries while supporting collaboration of engineering units. In such a case, FEA simulation data and task execution must not leave the servers of a specific region.

Constraints: The main technical constraint for this software modernization project is that the legacy FEA simulation tool cannot be modified. Therefore our approach needs to transparently adhere to the usage requirements of the legacy simulation software; namely (1) it must have network access to the license management server (even outside of the Siemens network) and (2) all input/output files must be available locally.

III. Approach

A. High-level Software Architecture

Our *Simulation SaaS* (see Figure 2) provides engineers the ability to perform FEA as simulation jobs on a hybrid cloud platform without the need to install or run the simulation software on their local computer. The service implements a standardized way for users to structure, package and submit simulation jobs which can then be executed in an automated way. The *Simulation SaaS* is built using a distributed microservice architecture that allows the service to be deployed flexibly across different computers, networks and geographic regions. The service uses a configurable pool of heterogeneous (public and private) computation nodes to concurrently execute simulations jobs in order to speed up the computation time for users running large complex simulation campaigns.

1) Servicification strategies: On a high-level, our simulation as a service challenge can be formalized as a function $analyze(M, Func, Params)$ where M represents a hierarchical engine design model (with components and their dependencies) to be analyzed by using a designated functionality $Func$ along a set of operating conditions $Params$.

For related analysis challenges, cloud-based servificiation can possibly be carried out on different levels of granularity.

1) **Entire model, entire tool, all params**: As the lowest granularity solution, one can execute an entire simulation campaign on the cloud by invoking the tool on the entire model for all the operating conditions as a single service.
2) **Entire model, entire tool, specific params**: Alternatively, a simulation campaign can be divided into analysis points (one for each operating condition) and the tools is invoked on the entire model for each point separately.
3) **Entire model, specific function, specific params**: At a next level of granularity, a simulation campaign can be performed over an entire model by invoking a designated analysis function (instead of invoking the entire tool).
4) **Partial model, specific function, specific params**: Finally, the most fine-grained solution could carry out analysis on selected components of the engine model.

As the legacy simulation software cannot be modified we could not provide a centralized cloud repository for managing hierarchical engine models. Neither could the individual FEA functions within the tool (such as thermal, structural and fluid analysis) be exposed as standalone services.

In our servicification strategy of the legacy FEA simulation tool, we decided to (1) define the boundaries of a transparent simulation job to use the *entire tool on entire engine models for specific analysis points* and (2) use a cloud based file storage to transfer input and output data for each simulation job.

The main benefit of our servicification strategy is that it can be adapted to incorporate other simulation tools used at Siemens Energy with only minor modifications. As a limitation, there is a likely performance loss compared to a more fine-grained servicification option, which exploits inter-model dependencies or individual tool functions.

2) Key components: The *Simulation SaaS* contains three core components; the *Manager*, the *Worker* and the *Executor*.

- The *Manager* is responsible for serving the service to users and essentially operates as its primary endpoint.
- The *Worker* is an agent responsible for pulling *Simulation Jobs* from the *Job Repository* and launching *Executors*.
- The *Executor* is an ephemeral container which will actually performs the FEA for a *simulation job*.

There are also external third party services that the core components of the *Simulation SaaS* depend upon, such as the

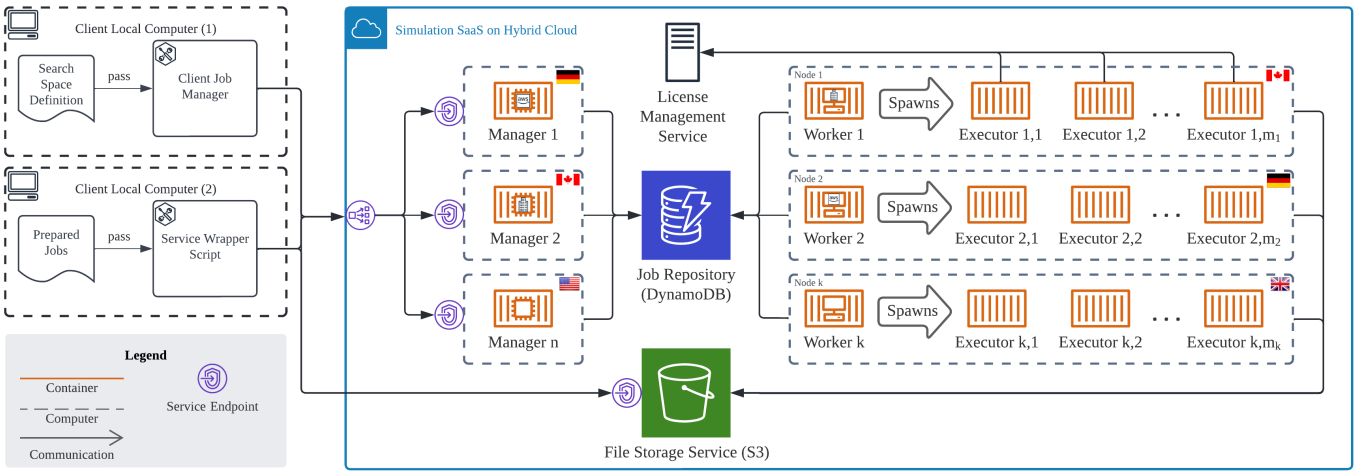

Fig. 2: Architecture for Simulation SaaS over a hybrid cloud platform

Job Repository (as a service), the *File Storage Service* and the *License Management Service*. Each component is built and packaged in such a way that it can be deployed independently of each other and across different environments, enabling the service to be fully distributed in a hybrid deployment.

B. Simulation Job

A *simulation job* is the primary resource of the *Simulation SaaS* that users interact with. In order to perform FEA using our service users must create and manage *simulation jobs*. A *simulation job* contains all the data and functionality required for a single self-contained simulation execution. A high-level overview of its detailed contents is provided in Figure 3, while its lifecycle is detailed in Figure 4.

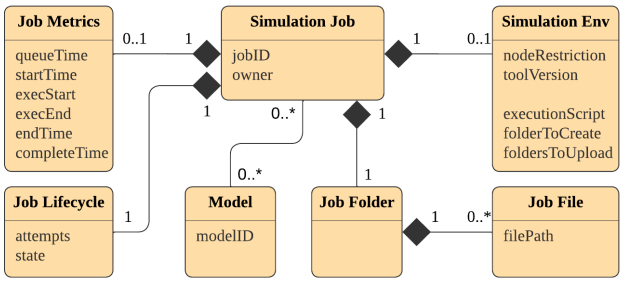

Fig. 3: Simulation job definition

1) Taxonomy: Every simulation job has a unique identifier (*jobID*) which enables the users and the service to monitor and manipulate a task from creation to completion. Moreover, to ensure that the *Simulation SaaS* is being used effectively by users within Siemens Energy, every simulation job is tied to an employee (*owner*). A simulation job has various metadata (*job metrics*) which consists of the number of execution attempts, the queue time, the start time, the execution start & end time, the end time and the complete time.

Every simulation job is ultimately executed in a unique *simulation environment*, which is isolated and can be configured to the needs of the simulation job. In order to meet geographic and technical requirements, a simulation job can specify which type of node they want their simulation environment to exist within (*nodeRestriction*).

Each simulation job has a *job folder* to store all the *job files* necessary for their simulation. Simulation jobs should only rely on files within their *job folder* except for standard libraries which are available at predefined paths within the *simulation environment*. All references between files in a *job folder* must be relative as the *simulation environment* that ultimately executes the job is not transparent to the user. Technically, a *job folder* cannot contain empty sub folders, however the FEA simulation software may need to write to such a location. Thus, the simulation environment can be configured to create empty folders within its local job folder clone (*foldersToCreate*). Before a simulation job executes, its job folder is copied into its *simulation environment* and it becomes the working directory during the execution.

Each simulation job has an an entry point specified as an execution script (*executionScript*). It should be located in the job folder and its location is passed to the FEA simulation software during execution. Moreover, the FEA simulation software version to use for the job execution can also be configured within the *simulation environment* (*toolVersion*). At the end of the execution the *simulation environment* can be configured to only synchronize back a subset of the folders within the local job folder clone instead of everything (*foldersToUpload*).

During job execution, the FEA simulation software may load engine *models* from the local execution environment as specified by the execution script. *Models* are generally large files that contain the mechanical mesh data on which FEA is performed. As such, the same engine *model* can be reused by multiple simulation jobs (with each performing a specific FEA function at a particular analysis point). Simulation jobs can specify which *models* need to be accessed in their local execution environment by a *modelID*. Due to the large size and repetitive use of models, the *Simulation SaaS* maintains a file

repository for *models*, which makes them locally available for individual simulation jobs. This approach enables to reduce overall data transfer and storage costs and time required for using the *Simulation SaaS*, especially for hybrid deployments where data transfer costs within the cloud are much lower.

2) Lifecycle: A simulation job has a total of six possible states which each representing a different point in the lifecycle of a simulation as shown in Figure 4. Each state transition can be triggered either by the user or by various components in the *Simulation SaaS* based on certain events and rules.

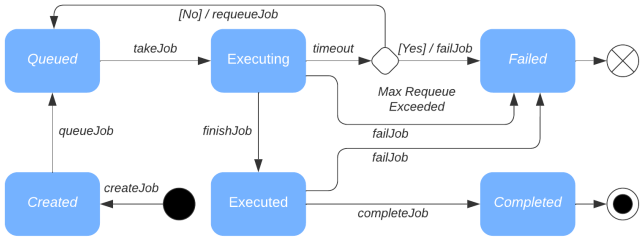

Fig. 4: Simulation job lifecycle

- **Created**: This state indicates that the simulation job has been created within the *Job Repository* along with all its configuration parameters, and is ready to accept *job files* in its *job folder* at the *File Storage Service*.
- **Queued**: This state indicates that the simulation job is ready to be taken by a *Worker* for execution, and it expects that all the *job files* have been uploaded to its *job folder* at the *File Storage Service*.
- **Executing**: In this state, the simulation job has been taken by a *Worker* and it is being executed by an *Executor*.
- **Executed**: This state shows that the execution script of the simulation job has successfully exited the FEA simulation software and the resulting files are available in the *job folder* at the *File Storage Service*. The *Executor* has terminated, and it is awaiting cleanup by the *Worker*.
- **Completed**: This state indicates that the *Executor* of the simulation job was successfully cleaned up by the *Worker* and the job is completed; all job data is ready for retrieval.
- **Failed**: The failed state indicates that the simulation job was unable to complete for some reason.

C. Manager

The *Manager* allows to create simulation jobs, queue those jobs, and get job information when needed. The *Manager* focuses only on handling the simulation jobs without actually executing them. The *Manager* exposes its service via a RESTful API which users can access using HTTP requests using the simulation job as the primary resource (see Table I).

TABLE I: REST API for Simulation SaaS Manager

Action	HTTP Method	URI
Create Job	POST	`/jobs`
Queue Job	PUT	`/jobs/<job-id>`
Get Job	GET	`/jobs/<job-id>`

- To create a job, a POST request is submitted for a simulation job together with its configuration details as query parameters. The *Manager* verifies each creation request and returns a *job ID* which is then used to track and manipulate the job throughout its lifecycle.
- When a simulation job is ready to queued, a PUT request needs to be issued together with the designated queued state.
- A simulation job can be retrieved from the task collection along a GET request by specifying the *job ID* in the URI.

As the primary function of the *Manager* is a service access point of the Simulation SaaS for end users, it does not directly interact with other core components. It only connects to the *Job Repository* to create, queue and retrieve simulation jobs. Furthermore, the *Manager* is packaged in a Docker container so that it can be deployed on any computer and built such that multiple instances of it can be deployed simultaneously for horizontal scaling and hybrid deployments.

D. Worker

The *Worker* is an agent that runs on a node in the *Simulation SaaS*. It is responsible for pulling queued jobs from the *Job Repository* and launching and managing *Executors* on the node to execute the simulation jobs. The *Worker* acts as a local job manager for a node, ensuring that jobs are properly processed. The *Worker* is packaged in a Docker container and hence can be deployed on any computer to make it into a node in the *Simulation SaaS*. The number of concurrent simulation jobs (*Executors*) allowed by the *Worker* on its node can be configured (*capacity*). Moreover, a *node type* can be set for each *Worker* such that jobs with a matching *node restriction* can be executed by that specific *Worker*.

Once deployed, launching *Executors* for simulation jobs is the primary function of the *Worker*. It periodically polls the *Job Repository* for queued jobs and reserves a job for execution based on a FIFO order. To allow for the distributed execution of simulation jobs, there is no limit on the number *Workers* that can be deployed. Consequently, the job reservation process is designed in a way such that only one *Worker* can secure any given job from the *Job Repository*. For every job secured, a corresponding *Executor* is launched, the start time and attempt counter is updated for the job in the *Job Repository*.

Further responsibilities of the *Worker* includes the management of *Executors* such as simulation job timeout and cleanup. The *Worker* periodically check points each *Executor*, and if the run time exceeds a maximum threshold, the *Executor* is terminated, and the corresponding job is requeued for another worker to eventually execute. Once a certain requeue limit is reached, the job will transition to a fail state. Upon termination, log files generated by the *Executor* are uploaded to the corresponding *job folder* at *File Storage Service* and the job state and complete time are updated in the *Job Repository*.

E. Executor

The *Executor* is a temporal container that provides an *simulation environment* where the actual FEA is performed using the legacy FEA software. The *Executor* provides an

isolated environment for the execution of each simulation job, and it is responsible for their successful execution. An *Executor* is launched by a *Worker* for every simulation job that it secures by passing the simulation job ID and other relevant configuration data. The lifecycle of an *Executor* has three distinct phases: initialization, execution and wrap-up.

- *Initialization*: A local copy of the *job folder* and its *job files* is made in the *simulation environment* which is then initialized by creating the required empty folders.
- *Execution*: The requested version of the FEA simulation software (*toolVersion*) is launched with the execution script as its input. The execution start time is updated in the *Job Repository*. This phase continues until the FEA simulation software terminates at which point the execution end time is updated in the *Job Repository*.
- *Wrap-up*: Analysis results and log files generated during execution are synchronized back to the *job folder* and the end time is updated in the *Job Repository*.

F. Client-side Components

The *Simulation SaaS* is made available via a RESTful API which can be accessed by end users and client-side (front-end) tools by HTTP requests. To assist those mechanical engineers who are not skilled in API programming, we developed two front-end components that hide the technicalities of API calls.

- *Service Wrapper* helps engineers execute simulation campaigns on the SaaS. Engineers prepare jobs on their local machine and then use scripts of the *Service Wrapper* to create the simulation jobs, upload the job files, queue their campaign jobs and monitor the progress of a campaign.
- *Job Manager* enables engineers to define a search space (e.g. temperature, air pressure, etc.) to perform FEA on the engine models. Given such search space definitions as input, the *Job Manager* generates the required simulation jobs and automatically executes them using the *Simulation SaaS*.

The *Simulation SaaS* combined with client-side components enables the end users to perform complex simulation campaigns for FEA powered by an existing (trusted) legacy simulation software as automated jobs instead of manually invoking the software. Providing a transparent and simple service to use the legacy simulation software and manage its input/output data empowers the practitioners to produce better mechanical designs by more extensive simulations.

G. Export Control Considerations

As the *Simulation SaaS* needs to process simulation jobs which may involve engine models and data under export control regulations, we have also developed a mechanism to restrict where simulation jobs can be executed. This is achieved by a tagging strategy where *Workers* deployed on a certain node type (i.e. the region, OP/AWS) can be tagged with a unique identifier. Consequently, jobs which must execute on certain node type can be tagged with the appropriate identifier (*nodeRestriction* - represented symbolically by flag icons in Figure 2) in their configuration, ensuring that they will be executed in compliance with export control.

Export control restrictions for data to reside in a particular region can be fulfilled with the public cloud as they offer various regions in which *Workers* can be deployed to meet geographic requirements. Moreover, access to public cloud resources where confidential data may reside can be strictly controlled and secured using the comprehensive set of identity and access management tools provided by the cloud platforms.

IV. Evaluation

In order to evaluate the performance of our *Simulation SaaS*, we have conducted an extensive set of experiments to address the following research questions:

- **RQ1**: What is the runtime overhead of executing a simulation job using the *Simulation SaaS*?
- **RQ2**: What is the wait time of simulation jobs in a simulation campaign on AWS vs. on-premise servers?
- **RQ3**: What is the performance of a realistic hybrid configuration for a complex simulation campaign compared to existing engineering practice at the industrial partner?

A. General Measurement Setup

Motivation: We carry out an extensive performance evaluation of our *Simulation SaaS* by running a specific *real simulation job* for FEA of a real world gas turbine component developed by Siemens Energy. In our measurements, campaigns of simulation jobs will be executed on different configurations of the *Simulation SaaS* consisting of on-premise (OP) and/or Amazon Web Services (AWS) public cloud nodes.

Measured performance parameters: In order to evaluate the performance and effectiveness of the *Simulation SaaS*, we measure various execution times related to a simulation job in accordance with its lifecycle Figure 4 defined as follows:

- *Wait time* t_w: The time period from the moment a simulation Job is queued by the user to the moment a *Worker* secures the simulation job from the job repository.
- *Setup time* t_s: The time period from the moment a *Worker* has secured a simulation job, launched an *Executor* until the moment the *Executor* is up and running has finished preparing the simulation environment for the execution.
- *Runtime* t_r: The time period from the moment the *Executor* launches the FEA simulation software until it has completed execution and exits.
- *Wrap-up time* t_u: The time period from the moment the FEA simulation software terminates until the *Executor* wraps up the simulation environment and exits itself.
- *Cleanup time* t_c: The time period from the moment the *Executor* exits until the *Worker* cleans up the *Executor*.
- *Execution time* t_e: The execution time for a simulation job is the sum of all four previous items except for the wait time, i.e. $t_e = t_s + t_r + t_u + t_c$.
- *Total time* t_t: The total completion time for a simulation job includes wait time and execution time: $t_t = t_w + t_e$.

Each of these time periods are computed from timestamps that are captured by the different components of the *Simulation SaaS* as they interact with any simulation job, and were previously defined in Figure 3 (see *Job Metrics*).

Configurations of Simulation SaaS: The *Simulation SaaS* can be configured in various ways to meet different needs. The type of node(s) that the *Worker(s)* are deployed on is part of the configuration of the SaaS, as adjusting the number of workers and the type of the their underlying node can affect the overall performance of the SaaS. To ensure the underlying node hardware is not overloaded, we need to set the the maximum number of concurrent simulation jobs (*capacity*) for each *Worker*. In our performance evaluation, various *Simulation SaaS* configurations are investigated (with varied number of *Workers* and different simulation job capacity):

- **AWS-SEQ**: single worker running on a AWS type node, with a capacity of one *simulation job*.
- **AWS-PAR**: two workers running on a AWS type node, each with a capacity of two *simulation jobs*.
- **OP-SEQ**: single worker running on a OP type node, with a capacity of one *simulation job*.
- **OP-PAR**: single worker running on a OP type node, with a capacity of four *simulation jobs*.
- **HYB**: three workers running, two on a AWS type node and one on a OP type node each with a capacity of two.

Since limited available resources may influence the runtime of simulation jobs, every *Worker* can be configured to allocate certain amount of RAM to each *Executor* it launches for the simulation jobs it secures. In our performance evaluation, the *RAM allocated* by all *Workers* in the SaaS to *Executors* can be either **1GB** or **8GB**.

B. RQ1: Runtime Overhead

Rationale: In addition to the runtime of the FEA simulation software, the *Simulation SaaS* introduces a measurable overhead during the lifecycle of simulation jobs. By exploring the typical overhead of a simulation job, we evaluate the efficiency of the *Simulation SaaS* with respect to the computational capacity being deployed.

1) Measurement setup: The *overhead of a single job* t_o is defined as the sum of the setup, wrap-up and cleanup time: $t_o = t_s + t_u + t_c$. We measure these time periods and compare it against the actual runtime t_r of the FEA simulation software to assess the overhead of executing a simulation job when using the *Simulation SaaS*. As wait time t_w depends on the actual SaaS configuration, it will be assessed as part of **RQ2**.

To accurately determine these time periods we execute a campaign with 30 identical *simulation jobs* on eight different *Simulation SaaS* configurations: (1) **AWS-SEQ** with **1GB**; (2) **AWS-PAR** with **1GB**; (3) **AWS-SEQ** with **8GB**; (4) **AWS-PAR** with **8GB**; (5)**OP-SEQ** with **1GB**; (6) **OP-PAR** with **1GB**; (7) **OP-SEQ** with **8GB**; (8) **OP-PAR** with **8GB**.

Results: The medians of overhead times in various configurations of *Simulation SaaS* are plotted in Figure 5 (centre). We also present the simulation runtime t_r for different configurations (right), and depict the percentage breakdown of the task execution time in respective configurations (left). Based on the measurement results, we make the following observations on certain characteristics of the *Simulation SaaS*.

O1.1: When the SaaS is configured as **AWS-PAR** or **AWS-SEQ**, the overhead of the jobs is significantly smaller than **OP-PAR** or **OP-SEQ**, both in time and percentage. Part of this can be explained by the fact that AWS based *File Storage Service* offers better network performance on AWS type nodes in the **AWS-PAR/AWS-SEQ** SaaS configurations.

O1.2: Given same RAM allocation, the parallel execution of jobs results in increased overhead time t_o and simulation runtime t_r, which is attributed to the overhead of resource constrained multitasking in the underlying hardware.

O1.3: We observe that increased RAM allocation significantly decreases overhead time t_o, and runtime t_r with on-premise configurations (**OP-SEQ/OP-PAR**). Moreover, with **1GB** we experienced a major increase in runtime likely attributed to thrashing. However, RAM allocation did not really influence runtime on AWS configurations.

> **RQ1**: *The overhead imposed by the* Simulation SaaS *takes 5-12 minutes, which is less than 8% of the overall execution time of a typical simulation job. For simulation jobs executed within AWS, the overhead is less than 4%.*

With comparable network setup, the absolute overhead time (required for instantiating and managing a simulation job) is expected to stay within the same range for other simulation jobs. However, the overhead percentage with respect to execution time may change for longer simulation jobs.

C. RQ2: Job Wait times

1) Rationale: The *Simulation SaaS* is designed to process simulation jobs in parallel to enable lower overall (end-to-end) execution time for simulation campaigns. By exploring the *wait times* of simulation jobs in campaigns launched on different *Simulation SaaS* configurations, we can assess the performance and potential speedup of using the SaaS.

Measurement setup: The wait time of single task t_w is defined as the time period between the moment a job gets queued by a user to when it is selected for execution by a *Worker*. Wait time t_w depends on the configuration of the *Simulation SaaS*; mainly the number of *Workers* deployed within the SaaS, the node type the *Workers* are deployed on, and the combined capacity of all the *Workers* (i.e. the sum of concurrent simulation jobs executable by each *Worker*).

To accurately assess the performance of the *Simulation SaaS*, we execute a campaign with 30 identical simulation jobs on four different configurations with the following parameters: (1) **AWS-PAR** with **1GB**; (2) **AWS-PAR** with **8GB**; (3) **OP-PAR** with **1GB**; (4) **OP-PAR** with **8GB**. All SaaS configurations have the same *combined capacity of four jobs* (i.e. at most four jobs can be executed at a time across the SaaS).

Tracking the wait time t_w of every job in a campaign allow us to assess the performance of the *Simulation SaaS* compared to a theoretical minimum wait time t_w^{min}. The formula to calculate the minimum wait time t_w^{min} of a simulation campaign of size k is defined as follows:

$$t_w^{min}(k) = \left(\left\lceil \frac{k}{\sum_{i=1}^{N} C_i} \right\rceil - 1 \right) \times t^{min}$$

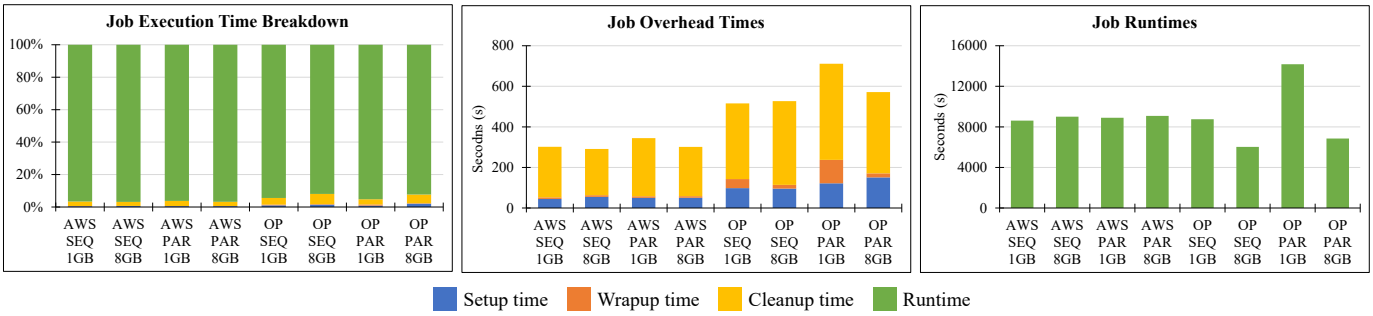

Fig. 5: *Left*: Percentage breakdown of total times (t_t) *Centre*: Median overhead time for simulation jobs for various *Simulation SaaS* configurations (t_o) *Right*: Median runtime of simulation jobs (t_r)

- N : The number of nodes in the SaaS configuration
- C_i : Capacity, i.e. the maximum number of concurrent simulation jobs on a node
- k : The size of the simulation campaign
- t^{min} : The (reference) execution runtime of a simulation job, i.e. the median job execution time t_e of the best SaaS configuration observed in **RQ1** (6,555.5 seconds).

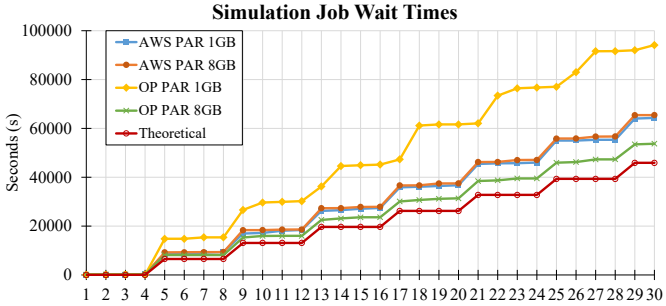

Fig. 6: Simulation job wait times t_w for various configurations

Results: The wait time t_w for each simulation job within each *Simulation SaaS* configuration is plotted in Figure 6, and the theoretical $t_w^{min}(k)$ is also plotted for reference. Based on the measurement results, we make the following observations on the simulation job wait times and of the *Simulation SaaS*.

O2.1: In all *Simulation SaaS* configurations, the wait time for all simulation jobs follows a step function: four (identical) jobs execute almost simultaneously at a time until all jobs within the campaign are finished.

O2.2: When the SaaS is configured as **AWS-PAR**, increasing the RAM allocation from **1GB** to **8GB** does not provide substantial reduction in job wait times, hence, the wait times under both configurations is nearly identical. Real job wait times for **AWS-PAR-1GB** deviate 29%-44% and **AWS-PAR-8GB** deviate 39%-44% from the theoretical t_w^{min}.

O2.3: When the SaaS is configured as **OP-PAR**, increasing the RAM allocation from **1GB** to **8GB** results in significant reduction in job wait times. With **1GB**, the job wait times are the largest observed (due to thrashing issues explored in **RQ1**), deviating between 80%-135% from the theoretical t_w^{min}. For **8GB** configurations, the job wait times are the smallest observed deviating only 15%-24% from the theoretical t_w^{min}.

> **RQ2**: *With sufficient resources, the job wait times in a simulation campaign vary from the theoretical minimum up to 44% and 24% in AWS and OP configurations, respectively. Moreover, in resource constrained environments, the wait time can deviate by as much as 135%.*

D. RQ3: Speed-up of Hybrid Configuration

1) Rationale: The *Simulation SaaS* is designed to be deployed in a heterogeneous configuration with different node types. Besides enabling the parallel execution of simulation jobs, the hybrid configuration of the *Simulation SaaS* also enables Siemens Energy to leverage existing on-premise resources while taking advantage of the flexibility offered by public cloud providers such as AWS. By measuring the total execution time of a realistic simulation campaign on a typical hybrid SaaS configuration, we can gain a better understanding of the performance of our framework, and compare it to the existing engineering practice using the legacy setup.

Measurement setup: For this RQ, we measure the total (end-to-end) completion time t_t of each job from the moment it is queued until it has been cleaned up after completing its execution. Consequently, the total execution time $t_t(k)$ of a simulation campaign (with k jobs) is the time period from the moment the first simulation job is queued until the last job has been cleaned up. To effectively measure the performance of the *Simulation SaaS*, we run a campaign with $k = 30$ identical simulation jobs and we execute this measurement three times.

The simulation campaign is executed on a hybrid cloud configuration (**HYB** with **8GB**) which represents a typical SaaS configuration to be used at Siemens Energy. This SaaS configuration has a combined job capacity of *six*, which allows the concurrent execution of six simulation jobs.

We compare the total execution time of each simulation campaign $t_t^{hyb}(k)$ to three baselines to evaluate the performance of hybrid deployment of the *Simulation SaaS*.

- *Theoretical minimum*: Given the theoretical minimum execution time of a single job t^{min} (as defined in **RQ2** as 6,555.5 seconds), we define the theoretical minimum total execution time for a simulation campaign of size k with a

SaaS capacity of *6* (like our **HYB** configuration) as follows: $t_t^{min}(k) = \lceil \frac{k}{6} \rceil \times t^{min}$.

- *Sequential execution*: The estimated time t_t^{seq} of executing a campaign of size k sequentially in the *Simulation SaaS* is calculated as follows: $t_t^{seq}(k) = t^{min} \times k$.
- *Legacy practice*: We measured the execution time t_r^{leg} of one simulation job in the legacy setup (i.e. on a desktop computer). We carried out thirty measurements with a median execution time of 5,502 seconds. As a further reference value, we compute the *estimated total execution time* t_t^{leg} for a simulation campaign of size k in the legacy setting as $t_t^{leg}(k) = t_r^{leg} \times k$. Note that this value is likely a lower bound for the real total execution time of a campaign in a legacy setting as the actual creation, setup and wrap-up time of a simulation job is excluded.

Results: The complete execution time for each simulation job within each campaign for a **HYB** SaaS configuration is plotted in Figure 7, and the theoretical $t_t^{min}(k)$ and legacy $t_t^{leg}(k)$ is also plotted for reference. Based on the measurement results, we make the following observations on the total execution time of the simulation campaigns.

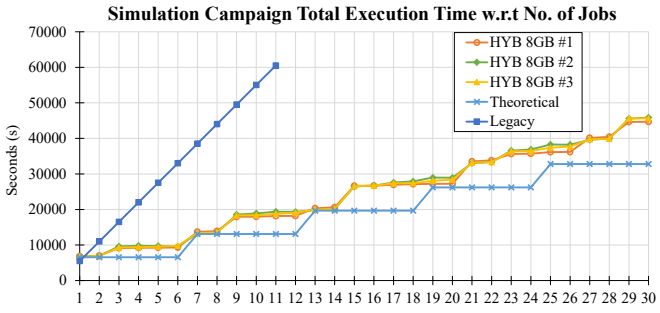

Fig. 7: Total execution time of simulation campaign execution times with an increasing number of simulation jobs

O3.1: The total execution time of some jobs gets very close to the theoretical minimum as they are executed on OP nodes which generally have faster execution times. However, jobs executed on AWS nodes have longer execution times and made the total execution time of the simulation campaigns 39% higher on average when compared to the theoretical minimum.

For our simulation campaigns with $k = 30$ jobs each, the theoretical minimum total execution time of a campaign on our SaaS configuration is $t_t^{min}(k)$. Moreover, the estimated time of executing such a campaign sequentially on the SaaS is $t_t^{seq}(k)$. Therefore, the theoretical maximum speedup offered by our **HYB** SaaS configuration is $s^{max} = \frac{t_t^{seq}(k)}{t_t^{min}(k)}$. Finally, the expected total execution time of our our campaign in a legacy setting would be $t_t^{leg}(k)$.

Table II summarizes speed-up results for all three simulation campaigns executed on our **HYB** SaaS configuration;

1) The speedup with respect to sequential execution on the SaaS: $s^{seq} = \frac{t_t^{seq}(k)}{t_t^{hyb}(k)}$.
2) The percentage of speedup achieved w.r.t. the theoretical maximum speedup s^{max} is: $p^{seq} = \frac{s^{seq}}{s^{max}} = \frac{t_t^{min}(k)}{t_t^{hyb}(k)}$.
3) The speedup with respect to legacy setting (with sequential execution): $s^{leg} = \frac{t_t^{leg}(k)}{t_t^{hyb}(k)}$.
4) The percentage reduction of the total execution time of the simulation campaign compared to a legacy setting: $p^{leg} = 1 - \frac{t_t^{hyb}(k)}{t_t^{leg}(k)}$.

TABLE II: Speedup of **HYB** SaaS campaigns compared to sequential execution and legacy practice

HYB 8GB Run #	1	2	3
Speedup w.r.t sequential	4.40	4.29	4.31
Percentage w.r.t theoretical (%)	73.35	71.47	71.78
Speedup w.r.t. to legacy	3.73	3.60	3.61
Percentage reduction w.r.t. legacy (%)	72.93	72.22	72.33

O3.2: The **HYB** configuration of the SaaS achieves a speedup of at least 4.29 when compared to sequential execution on the SaaS whilst also being very close to the theoretical maximum speedup (71%). The **HYB** speedup is lower than the theoretical maximum which is calculated using t^{min}, whereas the average job execution time t_e used for the **HYB** speedup is generally higher. As such, t^{min} was achieved exclusively with OP type nodes in **RQ1** while the **HYB** configuration of the SaaS also contains AWS type nodes as well.

O3.3: The **HYB** configuration of the SaaS achieves a speedup of at least 3.30 when compared to sequential execution in a legacy setting, and the total execution time of simulation campaigns on our **HYB** configured SaaS are at least 72% faster. Although the concurrent capacity is 6, our speedup is less since the SaaS measurements include all overhead while in a legacy setting only the bare metal execution time of the FEA simulation software is considered with no overheads.

O3.4: The median runtime t_r of a single job in a legacy setting 5,502 sec is considerably better then the best median t_r recorded in **RQ2** 6,028 sec. This can be explained by the fact that all simulation jobs in the SaaS run within an isolated container environment which increases the overhead, while in the legacy setting the FEA simulation software runs directly on a desktop computer (i.e. bare metal execution).

RQ3: *The Simulation SaaS deployed over a hybrid cloud platform with a 6 job capacity offers 3-4 times speedup for complex simulation campaigns compared to legacy settings while introducing negligible service overhead.*

E. Threats to Validity

Internal validity: We applied several techniques in order to increase the internal validity of our experimental results. (1) All timestamps within the SaaS relied upon the Network Time Protocol (NTP) to generate universally accurate time data. (2) Moreover, large simulations campaigns have been executed to identify and mitigate any transient effects of the underlying infrastructure. (3) Finally, all SaaS configurations have been chosen in a way to ensure that sufficient amount of resources (CPU, memory) are available (expect for one configuration which highlights the effects when simulations are run with insufficient amount of memory).

Our experimental evaluation is carried out on sample configurations that are expected to be used at Siemens Energy to maximize the use of existing resources. However, we did not investigate how to obtain the best configuration for a given simulation campaign, which may reduce simulation job wait times compared to results reported in Figure 6.

There are also certain limitations concerning the internal validity of the reported results. (i) The accuracy of timestamp generation was $\sim$1 second while overall accuracy was $\pm$5 seconds. (ii) Our measurements exclusively report execution times for successful simulation runs, hence no component failures have been incorporated. The execution time of a failing simulation run has not been reported. Since the overall failure rate was well below 3%, the overall effect is quite negligible.

External validity: We have applied various steps to increase the generalizability of our results. (1) Performance measurements of the SaaS were taken on different configurations including exclusively on-premise and public cloud based infrastructure, and a hybrid setup. (2) Our *Simulation SaaS* has been evaluated using a real engineering simulation job to determine real world performance.

Nevertheless, the generalizability of our results is also limited by various factors. (i) A specific type of public cloud and private cloud node instance has been used exclusively, thus one might experience a different simulation runtime when using other types of nodes. (ii) The network performance of any node may vary on the external or the cloud environment, which may influence the absolute overhead by increasing or decreasing file transfer time. (iii) The same (real) simulation job has been used for all performance evaluations while engineers may use more varied and complex jobs for their analysis. (iv) We have not evaluated the scale out of SaaS by using hundreds of nodes. While such scaling options are offered by AWS, they are not explored by Siemens Energy at the moment as their current focus is to maximize the usage of existing resources.

Construct validity: We have been measuring and evaluating standard performance metrics such as execution time (end-to-end as well as for individual phases), and speed-up with respect to runtime of reference configurations.

V. Related Work

Existing research related to our *Simulation SaaS* focuses on three main areas: simulation as a service, legacy software modernisation and hybrid cloud computing.

A. Simulation as a Service

Digital simulation is a computationally intensive process that requires high performance computers. Traditional simulation software is installed and run on the users local computer, which are configured to handle such workloads [13]. The major drawback of such an approach would be a fragmented deployment of the simulation software in any organization, with the computational resources being split unevenly amongst the users and not being fully utilized. Most importantly it would limit the execution of the simulation software to a single computer, making it difficult to execute large complex simulations or simulation campaigns. In order to address these challenges there has been research in the area of modelling and simulation as a service (MSaaS) which have two major components; the service and the infrastructure.

1) Service: Existing MSaaS approaches [14], [15], [16], [17], [18] expose modelling and simulation functions, frameworks, data and other related resources as services in a uniform and domain agnostic way. MSaaS is typically accessible via web-based communication standard such as HTTP using a RESTful API interface [16], [14], [18], [15] or web service protocols such as SOAP [19], [20].

While there are several frameworks that address some security aspects (e.g. privacy, trust, accountability) related to handling restricted data (see [21] for a survey), our *Simulation SaaS* is the first framework explicitly designed for handling export control regulations.

2) Infrastructure: Concerning underlying infrastructure, few recent frameworks [16] use a monolith tightly coupled architecture. Most existing approaches [15], [14], [17] offer a layered middleware architecture which decouples the service from the infrastructure (computation platform), thus simulation services can be executed on heterogeneous computing units. Such a middleware enables scaling the infrastructure based on actual demand in a flexible way.

Our *Simulation SaaS* further increases flexibility by decouple the service from the operating system by using Docker containers (instead of virtual machines used by others).

B. Modernizing Legacy Software

While legacy software has been causing engineering challenges for decades, deploying legacy software to public cloud, or turning it into a service is a typical path to take in software modernization research.

1) Migrating to public cloud: Migrating legacy software deployments to the public cloud requires a thorough and systematic analysis of the application and the purpose of the migration [22], [23], [24], [25]. It can be summarized in a three step process; (1) the migration is planned by modelling the application functionality and its requirements, (2) the migration is designed and implemented based on stakeholder goals and cloud capability, (3) the cloud deployment is tested for validation. The goal of most cloud migrations is to improve reliability and scalability while controlling costs [23].

2) Migrating to Service Oriented Architecture: The service oriented architecture (SOA) model is a modern approach for software design while legacy software generally has a tightly coupled monolith design. Migrating to a SOA is the process of adapting and mapping the functionality provided by the legacy software into service units and resources [26], [27], [28], [29]. Furthermore, the internal application architecture is redesigned into smaller units (microservices) to improve scalability and maintainability [30], [31], [32], [33], [34].

The *Simulation SaaS* creates a simulation job as a service resource which users can then use to perform analysis. All the functionality of the FEA software is mapped to the configuration of the simulation job resource. Moreover, the service is

designed using a microservice architecture for scalability when deployed on the cloud instead of exclusively on-premise.

C. Hybrid Cloud Computing

Hybrid cloud computing consists of using cloud infrastructure composed of multiple distinct clouds (such as as private and public) which are then connected so that data can flow between them in a secure and seamless manner [35]. Research in the area indicates that potential cost savings and scalability as the two primary motivators for such deployments.

1) Cost Savings: Hybrid cloud computing can help users avoid vendor lock by making them deploy their applications in a platform agnostic manner [36]. More significant cost savings are possible as hybrid deployments can leverage existing on premise resources which are already payed for while also taking advantage of the flexible usage based pricing offered by public cloud providers [37]. Furthermore, the spare computational capacity of public clouds that is offered at significant discounts enables non critical applications to execute for a fraction of the cost anywhere else [38], [39], [40].

2) Scaling: Scaling possibilities with hybrid cloud computing are unique. Users can design and benefit from their private cloud to meet baseline computational demand. However during peak demand events when they rapidly require more resources they can *cloud burst* into the public cloud and provision more resources as needed. Cloud bursting is the main scaling advantage offered by hybrid cloud setups and allows users to meet spikes in demand in a timely manner [41], [42] or to manage resource intensive workloads [43].

As shown in Section IV, a unique aspect of cost reduction in *Simulation SaaS* (which is not mentioned in existing papers) is to minimize the net amount of time an engineer at Siemens Energy spends on a complex simulation campaign.

VI. Conclusion & Future Work

This paper reported on our software modernization initiative in a multi-disciplinary project aiming to run a legacy simulation tool used for finite element analysis in gas turbine engine design in a cloud-based setting. To assist engineers at Siemens Energy, we developed a *Simulation Software-as-a-Service* framework for automating FEA in a standards-compliant and transparent way, deployed over a hybrid cloud execution platform using a combination of public and on-premise nodes while complying to export control regulations.

An extensive performance evaluation highlights that our *Simulation SaaS* framework introduces negligible runtime overhead (in the range of 5-12 minutes attributing only to 4-8% of total execution time). Moreover, a hybrid SaaS configuration with a combined job capacity of 6 already provides 3-4 times speed-up when executing complex simulation campaigns compared to the legacy environment used at Siemens Energy.

The hybrid deployment of the *Simulation SaaS* enables FEA as a service for Siemens Energy without large upfront investments in high-performance computing hardware, as such using (i) existing on-premise computers which have minimal operating costs and (ii) AWS servers with large discounts.

The main *industrial benefit* of our *Simulation SaaS* is *increased productivity*. Compared to legacy practice, Siemens engineers need to spend less time with installing and configuring the simulation tool, and can spend more time with actually using the tool for FEA. Decreased execution time of simulation campaigns also increases *product quality*, as engineers can now investigate more analysis points with the same amount of (net) engineering time. Furthermore, engineers are now able to run campaigns with hundreds of jobs in a matter of days which used to take weeks, which will *decrease time-to-market* for new (upcoming) aeroderivative gas turbine engines.

Our future work will aim to provide support for *automated scaling*. Based on engineering priorities (cost model) and the needs of the simulation campaign, the *Simulation SaaS* will be able to determine the optimal number of computing nodes, and provision those nodes in a dynamic manner. Moreover, the existing fault tolerance mechanisms for simulation jobs are planned to be complemented by providing improved fault tolerance for *Worker* nodes. This further enhances robustness when running complex large-scale simulation campaigns potentially involving multiple design candidates. We also plan to design a more intelligent job scheduler which can optimize for cost, priority and security when assigning simulation jobs across the nodes in the service. Finally, we aim to further increase the robustness of our export control functionality by incorporating comprehensive control over data storage regions and user access to simulation job data.

Acknowledgements: This work was partially supported by the Digital Multidisciplinary Analysis and Design Optimization Platform for Aeroderivative Gas Turbines project (Siemens Ca CRDPJ 513922-17 X-247371 and NSERC CRDPJ 513922-17 X-247323 funds). The last author was supported in part by the Wallenberg AI, Autonomous Systems and Software Program (WASP), Sweden.

The authors would like to express their gratitude to numerous engineers at Siemens Energy and graduate students at McGill University and École de technologie supérieure (ÉTS) who provided assistance during the collaboration. Specifically, I would like to extend my appreciation my colleagues Faiq Khalid, Ali Mortada, Omotayo Bankole, Dirk Zimmermann and Stefan Hertel who supported me through this research. Finally, the authors are also grateful for the insightful comments of the anonymous reviewers of the paper.

REFERENCES

[1] I. Ibrahem, O. Akhrif, H. Moustapha, and M. Staniszewski, "An ensemble of recurrent neural networks for real time performance modeling of three-spool aero-derivative gas turbine engine," *Journal of Engineering for Gas Turbines and Power*, vol. 143, no. 10, 2021.

[2] H. Hanachi, C. Mechefske, J. Liu, A. Banerjee, and Y. Chen, "Performance-based gas turbine health monitoring, diagnostics, and prognostics: A survey," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1340–1363, 2018.

[3] E. Tsoutsanis, N. Meskin, M. Benammar, and K. Khorasani, "Dynamic performance simulation of an aeroderivative gas turbine using the matlab simulink environment," in *ASME International Mechanical Engineering Congress and Exposition*, vol. 56246. American Society of Mechanical Engineers, 2013, p. V04AT04A050.

[4] A. Benato, A. Stoppato, and A. Mirandola, "Dynamic behaviour analysis of a three pressure level heat recovery steam generator during transient operation," *Energy*, vol. 90, pp. 1595–1605, 2015.

[5] S. K. Yee, J. V. Milanović, and F. M. Hughes, "Validated models for gas turbines based on thermodynamic relationships," *IEEE transactions on power systems*, vol. 26, no. 1, pp. 270–281, 2010.

[6] F. Haglind and B. Elmegaard, "Methodologies for predicting the part-load performance of aero-derivative gas turbines," *Energy*, vol. 34, no. 10, pp. 1484–1492, 2009.

[7] J. Martinsson, M. Panarotto, M. Kokkolaras, and O. Isaksson, "Exploring the potential of digital twin-driven design of aero-engine structures," *Proceedings of the Design Society*, vol. 1, p. 1521–1528, 2021.

[8] P. Timothé, B. Ahmed, S. Martín, Moustapha, Hany, K. Michael, and G. L. Francois, "Integration of secondary air system for multidisciplinary design optimization of gas turbines," 2019.

[9] G.-x. Chen and C.-y. Liu, "Cyclic plastic deformation behavior of tc4 titanium alloy under different microstructures and load conditions using finite element method," *Journal of Failure Analysis and Prevention*, vol. 21, no. 2, pp. 678–688, 2021.

[10] S. Sanaye and S. Hosseini, "Prediction of blade life cycle for an industrial gas turbine at off-design conditions by applying thermodynamics, turbo-machinery and artificial neural network models," *Energy Reports*, vol. 6, pp. 1268–1285, 2020.

[11] H. Taplak and M. Parlak, "Evaluation of gas turbine rotor dynamic analysis using the finite element method," *Measurement*, vol. 45, no. 5, pp. 1089–1097, 2012.

[12] S. Pilarski, M. Staniszewski, M. Bryan, F. Villeneuve, and D. Varró, "Predictions-on-chip: model-based training and automated deployment of machine learning models at runtime," *Softw. Syst. Model.*, vol. 20, no. 3, pp. 685–709, 2021.

[13] S. Guo, F. Bai, and X. Hu, "Simulation software as a service and service-oriented simulation experiment," in *2011 IEEE International Conference on Information Reuse & Integration*, 2011, pp. 113–116.

[14] S. Wang and G. Wainer, "Modeling and simulation as a service architecture for deploying resources in the cloud," *Int. Journal of Modeling, Simulation, and Scientific Computing*, vol. 07, no. 01, p. 1641002, 2016.

[15] K. Al-Zoubi and G. Wainer, "RISE: A general simulation interoperability middleware container," *Journal of Parallel and Distributed Computing*, vol. 73, no. 5, pp. 580–594, 2013.

[16] T. Treichel, P. O. Antonino, F. S. Santos, and L. S. Rosa, "Simulation-as-a-Service: a simulation platform for cyber-physical systems," in *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, 2021, pp. 155–161.

[17] W.-T. Tsai, W. Li, H. Sarjoughian, and Q. Shao, "SimSaaS: Simulation software-as-a-service," in *Proc. of the 44th Annual Simulation Symposium*. Society for Computer Simulation International, 2011, p. 77–86.

[18] S. Wang and G. Wainer, "A simulation as a service methodology with application for crowd modeling, simulation and visualization," *SIMULATION*, vol. 91, no. 1, pp. 71–95, 2015.

[19] C. Seo and B. P. Zeigler, "Automating the DEVS modeling and simulation interface to web services," in *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, 2009.

[20] J. B. Fitzgibbons, R. M. Fujimoto, D. Fellig, S. D. Kleban, and A. J. Scholand, "IDSim: an extensible framework for interoperable distributed simulation," in *Proceedings of the 2004 IEEE International Conference on Web Services*. IEEE Computer Society, 2004, pp. 532–539.

[21] E. Cayirci, "Modeling and simulation as a cloud service: A survey," in *2013 Winter Simulations Conference (WSC)*, 2013, pp. 389–400.

[22] A. Bergmayr, H. Brunelière, J. L. C. Izquierdo, J. Gorroñogoitia, G. Kousiouris, D. Kyriazis, P. Langer, A. Menychtas, L. Orue-Echevarria, C. Pezuela, and M. Wimmer, "Migrating legacy software to the cloud with ARTIST," in *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 465–468.

[23] M. F. Gholami, F. Daneshgar, G. Beydoun, and F. Rabhi, "Challenges in migrating legacy software systems to the cloud — an empirical study," *Information Systems*, vol. 67, pp. 100–113, 2017.

[24] O. Bushehrian and S. Y. Nabavi, "A new method for migrating legacy applications to the cloud: a finite state process approach," in *2017 International Symposium on Computer Science and Software Engineering Conference (CSSE)*, 2017, pp. 86–91.

[25] A. L. Shastry, D. S. Nair, B. Prathima, C. P. Ramya, and P. Hallymysore, "Approaches for migrating non cloud-native applications to the cloud," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0632–0638.

[26] H. M. Sneed, "Integrating legacy software into a service oriented architecture," in *10th European Conf. on Software Maintenance and Reengineering (CSMR 2006)*. IEEE Computer Society, 2006, pp. 3–14.

[27] H. M. Sneed and H. M. Sneed, "Wrapping legacy software for reuse in a SOA," 2005.

[28] S. Rochimah and A. Sheku, "Migration of existing or legacy software systems into web service-based architectures (reengineering process): A systematic literature review," *International Journal of Computer Applications*, vol. 133, pp. 43–54, 01 2016.

[29] K. Garcés, R. Casallas, C. Álvarez, E. Sandoval, A. Salamanca, F. Viera, F. Melo, and J. M. Soto, "White-box modernization of legacy applications: The oracle forms case study," *Computer Standards & Interfaces*, vol. 57, pp. 110–122, 2018.

[30] S. Frey and W. Hasselbring, "Model-based migration of legacy software systems to scalable and resource-efficient cloud-based applications: The cloudmig approach," in *Proc. of the First Int. Conf. on Cloud Computing, GRIDs, and Virtualization*, Lisbon, Portugal, 2010, pp. 155–158.

[31] M. A. A. Sheikh, H. A. Aboalsamh, and A. Albarrak, "Migration of legacy applications and services to service-oriented architecture (SOA)," in *The 2011 International Conference and Workshop on Current Trends in Information Technology (CTIT 11)*, 2011, pp. 137–142.

[32] J. Kazanavičius and D. Mažeika, "Migrating legacy software to microservices architecture," in *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, 2019, pp. 1–5.

[33] H. Knoche and W. Hasselbring, "Using microservices for legacy software modernization," *IEEE Software*, vol. 35, no. 3, pp. 44–49, 2018.

[34] S. G. Haugeland, P. H. Nguyen, H. Song, and F. Chauvel, "Migrating monoliths to microservices-based customizable multi-tenant cloud-native apps," in *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2021, pp. 170–177.

[35] P. Mell, T. Grance *et al.*, "The NIST definition of cloud computing," 2011.

[36] K. Kritikos, P. Skrzypek, A. Moga, and O. Matei, "Towards the modelling of hybrid cloud applications," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 291–295.

[37] T. Guo, U. Sharma, P. Shenoy, T. Wood, and S. Sahu, "Cost-aware cloud bursting for enterprise applications," *ACM Transactions on Internet Technology*, vol. 13, no. 3, May 2014.

[38] I. Menache, O. Shamir, and N. Jain, "On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud," in *11th International Conference on Autonomic Computing (ICAC 14)*. USENIX Association, June 2014, pp. 177–187.

[39] K. Tamrakar, A. Yazidi, and H. Haugerud, "Cost efficient batch processing in amazon cloud with deadline awareness," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, 2017, pp. 963–971.

[40] S. G. Domanal and G. R. M. Reddy, "An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment," *Future Generation Computer Systems*, vol. 84, pp. 11–21, 2018.

[41] M. Mattess, C. Vecchiola, S. Garg, and R. Buyya, "Cloud bursting: Managing peak loads by leasing public cloud services," *Cloud Computing: Methodology, Systems, and Applications*, 01 2017.

[42] H. Zhang, G. Jiang, K. Yoshihira, and H. Chen, "Proactive workload management in hybrid cloud computing," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 90–100, 2014.

[43] T. Bicer, D. Chiu, and G. Agrawal, "A framework for data-intensive computing with cloud bursting," in *2011 IEEE International Conference on Cluster Computing*, 2011, pp. 169–177.