EXPPO: EXecution Performance Profiling and Optimization for CPS Co-simulation-as-a-Service

Yogesh D. Barve*, Himanshu Neema*, Zhuangwei Kang*, Hongyang Sun*, Aniruddha Gokhale*, Thomas Roth†
*Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37212, USA
{yogesh.d.barve, himanshu.neema, zhuangwei.kang, hongyang.sun, a.gokhale}@vanderbilt.edu
†Smart Grid and Cyber-Physical Systems Program Office, Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
thomas.roth@nist.gov

***Abstract*—A co-simulation may comprise several heterogeneous federates with diverse spatial and temporal execution characteristics. In an iterative time-stepped simulation, a federation exhibits the Bulk Synchronous Parallel (BSP) computation paradigm in which all federates perform local operations and synchronize with their peers before proceeding to the next round of computation. In this context, the lowest performing (i.e., slowest) federate dictates the progression of the federation logical time. One challenge in co-simulation is performance profiling for individual federates and entire federations. The computational resource assignment to the federates can have a large impact on federation performance. Furthermore, a federation may comprise federates located on different physical machines as is the case for cloud and edge computing environments. As such, distributed profiling and resource assignment to the federation is a major challenge for operationalizing the co-simulation execution at scale. This paper presents the Execution Performance Profiling and Optimization (EXPPO) methodology, which addresses these challenges by using execution performance profiling at each simulation execution step and for every federate in a federation. EXPPO uses profiling to learn performance models for each federate, and uses these models in its federation resource recommendation tool to solve an optimization problem that improves the execution performance of the co-simulation. Using an experimental testbed, the efficacy of EXPPO is validated to show the benefits of performance profiling and resource assignment in improving the execution runtimes of co-simulations while also minimizing the execution cost.**

***Index Terms*—cyber-physical systems, distributed simulation, cloud computing, latency, performance, resource management, gang scheduling**

I. Introduction

Cyber-physical systems (CPS) such as smart city, smart manufacturing, and transactive energy systems must make time-sensitive control decisions to ensure their safe operations. However, such CPS are an amalgamation of multiple dynamic systems such as transportation systems, vehicle dynamics, control systems, and power systems with different interconnected networks of different scales and properties. The assurance that such complex systems are safe and trustworthy requires simulation capabilities that can rapidly integrate tools from multiple domains in different configurations. Co-simulation is an attractive option for interlinking such multiple simulators to simulate higher-level, complex system behaviors.

The IEEE 1516-2010 High Level Architecture (HLA) defines the standardized set of services offered to a process in a distributed co-simulation [1]. A co-simulation in HLA comprises individual simulators called federates that are grouped into a logical entity called a federation. Each federate can have diverse computation and networking resource requirements, which must be considered when assigning resources to the simulators when the federation is deployed to cloud-fog-edge computation environments. The wall-clock execution time required for each federate's computation step varies based on this resource allocation. The variation in execution times can result in some federates waiting on others before they can proceed to the next stage of computation. Federates that require more wall-clock time for their computation steps (the low performing federates) increase the overall completion time (or *makespan*) of the entire simulation. This can potentially violate the simulation completion deadline. Thus, resource allocation and resource configuration selections are very important for the overall performance of distributed simulations.

Although co-simulations have traditionally been hosted in high-performance computing (HPC) clusters, there has been an increasing trend towards the adoption of cloud computing for simulation jobs. It is in this context that the Docker container run-time platform [2] provides a solution for running federates across different computation platforms by providing a unified packaging of simulation code with its software dependencies. But recent research [3] has shown that there are operational challenges, such as performance aware resource assignments, that need to be considered for running simulations in cloud environments. Furthermore, there are several infrastructure-related complexities that must be considered to run these distributed simulations in a cloud environment.

This work presents a performance profiling, simulation run-time optimization and resource configuration platform called EXPPO - (***EX****ecution* ***P****erformance* ***P****rofiling and* ***O****ptimization*). EXPPO uses distributed tracing [4] to assess federate level performance for each computational step. These performance characteristics are used at runtime to determine whether changes to the resource allocation of the federation

could enable shorter makespan for the simulation run. To enable distributed tracing, EXPPO utilizes the *opentracing* [5] specification for measuring the wall-clock time spent in each computational step of the simulation. To shield the developers from complexities when embedding tracing code in the simulation application logic, EXPPO leverages generative aspect of Model Driven Engineering (MDE) to auto-generate source-code snippets and configuration files for the simulation run. To provide resource recommendations for the individual federates, EXPPO uses the tracing information to build a resource-performance model and solves an optimization problem to find the resource allocation with the lowest makespan and cost for the co-simulation.

The main contributions of EXPPO are:

1) Demonstration that the default resource configuration for a federation has scenarios where a federate with a longer wall-clock execution time for its computational step increases the makespan of the co-simulation;
2) Development of an approach to generate tracing probes inside the source code of the simulation logic, which is required to perform distributed simulation profiling;
3) Development of a new resource recommendation engine which uses an optimization algorithm for finding the resource configuration for a federation that minimizes its overall makespan and cost; and
4) Validation of EXPPO showing its benefits on the performance of distributed simulation execution.

The rest of the paper is organized as follows. Section II provides a motivating use-case for EXPPO and lists its key requirements. Section III provides an overview of the EXPPO framework. Section IV describes the key EXPPO components and its resource configuration and optimization algorithms. EXPPO 's cloud architecture and co-simulation framework are described in Section V. The experiment evaluation results are described in Section VI, related works are described in Section VII and Section VIII concludes the paper.

II. Motivation and Solution Requirements

The computation pattern of many time-stepped co-simulations follows the Bulk Synchronous Parallel (BSP) model [6]. In this model, every participating simulation completes its computation for a given time step, waits for the other participating simulations to complete their computations, exchanges new state information with its peers, and only then proceeds to the next time step. Thus, if one simulation takes more time to execute, the other simulations are forced to wait for it and remain idle. This is illustrated in Figure 1 which shows an HLA federation with three federates. Each federate executes one computation task repeated each time step. Federate 1 takes the longest to execute, and the other two federates are waiting for Federate 1 to complete before moving to the next computation step. This increases the makespan of the entire simulation, thereby decreasing the performance of the simulation execution.

Recently, co-simulations have been deployed using container management solutions [7] [8], such as Docker Swarm

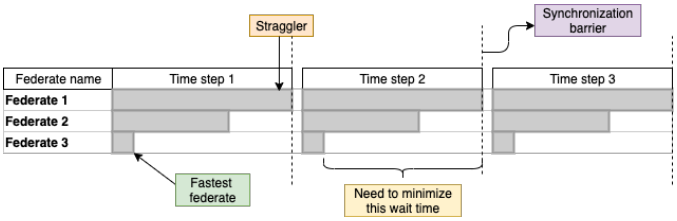

Fig. 1: Performance visualization of an example federation.

[9], Kubernetes [10], Mesos [11]. However, these solutions use queue-based scheduling, wherein the containers are allocated to machines one at a time. Hence, there could be instances where there may not be enough resources available in the cluster to schedule all the federates of the federation. However, a few federates may still get deployed while the remaining federates are stuck in the scheduler queue until more resources are available. This will cause the entire simulation to stall, since all the federates are required to participate in the simulation to progress. For instance, from Figure 1, if there are resources to deploy Federate 1 and Federate 3, and not sufficient resources to deploy Federate 2, the job scheduler should not deploy any federates and should wait until more resources are available. Hence, there is a need for bag-of-tasks scheduling mechanism for deploying these federates on the cloud computing environments.

The resource configuration also plays an important role in the execution time of the federates. Figure 2(a) depicts the performance of a federate when assigned different numbers of cores for executing a computation step. This federate is running a Freqmine application from the Princeton Application Repository for Shared-Memory Computers (PARSEC) benchmark [12] as its computation task. This figure shows that the execution time generally decreases with the increasing amount of resources assigned to the federate. Thus, there is a potential for minimizing the wait time by appropriately configuring the resources assigned to the federates. Minimizing the wait time can be done either by providing the highest resource configuration for all the federates, or finding a resource configuration that considers the cost of assigning the resources to the federates. However, deciding what resource configuration to select for a given computation is a non-trivial task for a simulation developer who may not have the domain expertise of configuring and running applications in cloud computing environments. Performance profiling of the federates can help in understanding the relation between the resource assignment and the execution performance. However, these simulations might be deployed across different physical and/or virtual host environments when running in cloud computing platforms; performance profiling for such distributed simulations can be very challenging.

Building on the above use-case, below are the four key requirements EXPPO aims to satisfy:

Requirement R1. *Conduct distributed performance tracing of the federates:* To understand the bottlenecks in federation performance, there is a need for logging and

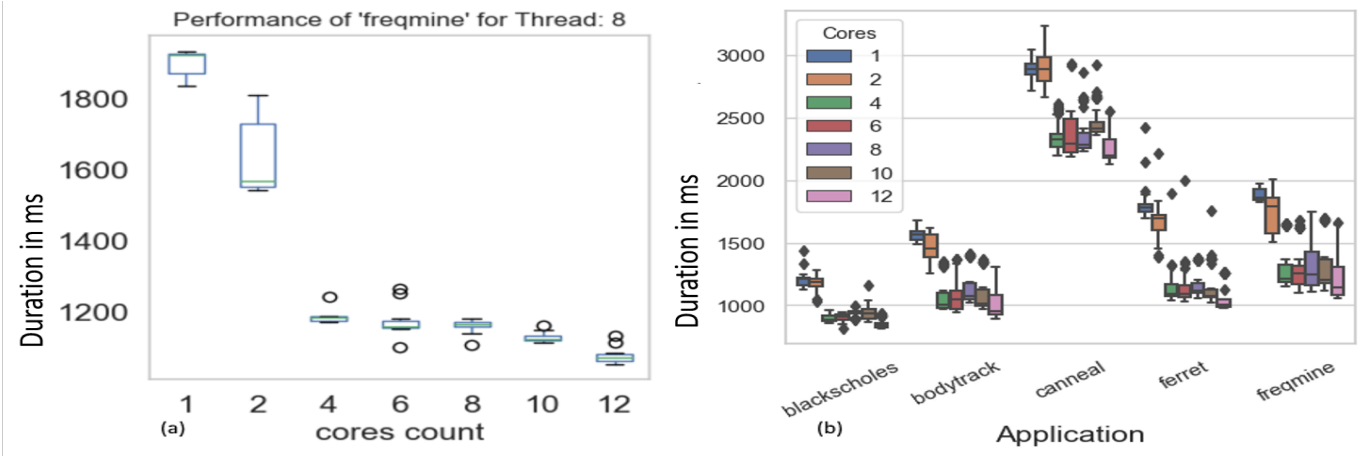

Fig. 2: (a) Performance of the federate running the PARSEC Freqmine application using 8 threads for different resource configuration selections. (b) Performance of five PARSEC benchmark applications for different resource configuration selections.

gathering execution performance traces of the federates. The tracing infrastructure needs to handle federates which are distributed across multiple physical hosts. Hence, the tracing infrastructure should be able to correlate traces from different federates of a federation.

Requirement R2. *Reduce complexity in provisioning software probes:* Requiring the developer to manually write source code for performance tracing for the federation can result in complexities and errors which need to be minimized. The developer needs to understand the tracing software and write code which adheres to the tracing software requirements. This is tedious for the developer, who now apart from writing the simulation logic must also setup and configure the tracing infrastructure. EXPPO should reduce the manual configuration of the tracing information, thereby reducing repeated effort by the developer.

Requirement R3. *Recommend resource configuration to minimize the co-simulation makespan and cost:* It can be challenging for an end user to determine the resource requirements for a federation because each federate can be configured differently. A bad resource configuration can have inadvertent effect on the completion time of the simulation. However, the choice of resource configuration also has an associated cost. Hence, the configuration must be chosen such that it satisfies both quality of service (QoS) and cost factors.

Requirement R4. *Provide a gang-scheduling algorithm for executing simulations:* The runtime platform should support deployment of multiple federate using bag-of-tasks scheduling (or gang scheduling) algorithm.

III. Overview of EXPPO

Figure 3(a) shows the workflow and components of EXPPO. The *design phase* requires the developer to model the federation using the federation development toolkit. This toolkit is based on the Web-based Generic Modeling Environment (WebGME) [13]. To measure the execution time of a federate in a time step, the simulator needs to embed a tracing code initializer and logger to record the execution time completion of the computation step. The tracing initialization code and the tracer configuration files are auto-generated by the custom WebGME model interpreters. Once the required code is generated and the user has implemented the necessary simulation logic, the federate is compiled into an executable image, which is then packaged inside a Docker container.

During the *profiling phase*, each federate is executed and profiled under different resource configurations. The execution time for each federate is logged using the tracing information, and this tracing information is stored in a centralized database. The resource configuration tuner uses the recorded logs together with user provided objectives to optimize the resource configuration for each federate. Finally, the optimized resource configuration is used to configure the co-simulation deployment accordingly.

During the *runtime phase*, the simulation job information for the federates in the co-simulation is submitted to the deployment manager. The job information includes the name of the federate Docker image, resource configuration requirements, etc. The deployment manager submits the scheduling information to the job scheduler, which handles the execution of the jobs on the co-simulation runtime execution platform.

IV. Design Elements of EXPPO

EXPPO allows users to design and deploy co-simulations on distributed compute infrastructures supported by Docker-based virtualization. Its generative capabilities simplify the auto-generation of performance monitoring instrumentation, configuration and probing for the different federates. Its resource configuration tuner optimizes resource allocations to federates to lower the makespan and execution cost for the federation execution. Its runtime platform supports parallel execution of different federations on its shared compute infrastructure. This section details each of the EXPPO components.

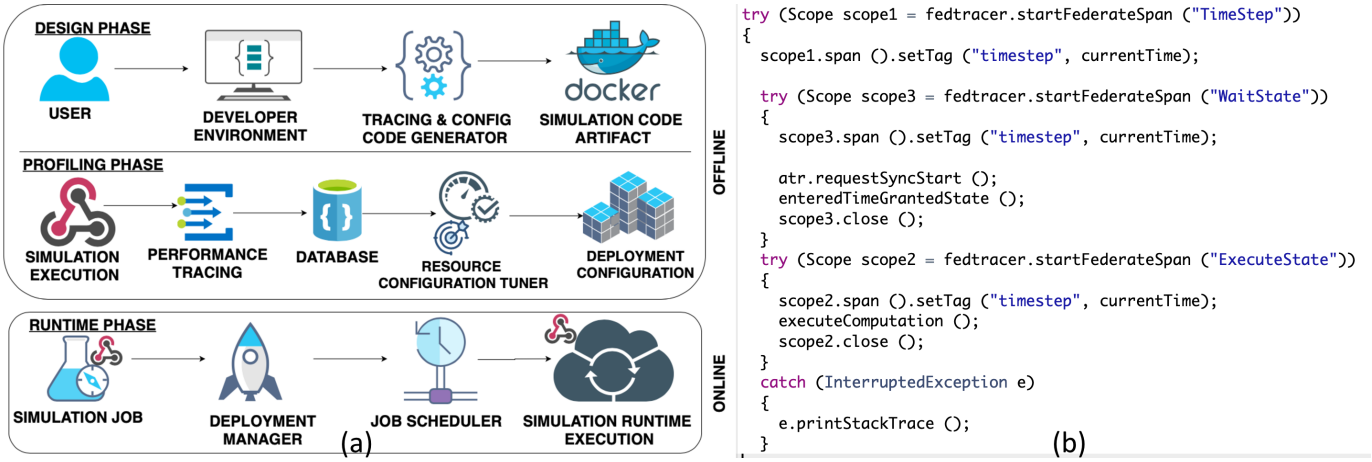

Fig. 3: (a) Workflow of EXPPO illustrating the connections between different components of the system. (b) Profiling code snippet in Java language generated leveraging the MDE techniques.

A. Performance Profiling of Federates

Understanding the performance characteristics of the co-simulation execution is of paramount importance when deciding how to run federates in runtime execution environments. Individual federates have different resource needs, and their execution performance will vary depending on how the resources are assigned. Thus, understanding the performance profiles of each federate is critical to the problem of optimizing the resource allocation and thereby lowering the makespan and execution cost of the federation execution. EXPPO leverages distributed tracing to assist in the logging of time stamps of distributed events generated in the federation (Requirement **R1**). It leverages Opentracing instrumentation [5] to track execution time spent during each computation time step of the federate. The execution time is then logged into a timeseries database for conducting performance analysis. It leverages MDE technologies [14] such as Domain Specific Modeling Language (DSML), code generators and model interpreters to synthesize performance profiling software artifacts which can be used for performance profiling of federates (Requirement **R2**). An example code snippet is shown in Figure 3(b).

B. Federation Resource Configuration Optimization

When running a federate in a Docker container, cloud providers usually have multiple resource configurations from which the user can select for their application. However, without analyzing the resource dependency of the application, it may be challenging for the user to select the resource configuration that meets the application's quality of service (QoS) requirement while at the same time minimizing the execution cost in terms of the cost of resources utilized. Figure 2(b) shows how the different resource configuration impacts the execution time of the federate which is running applications from the PARSEC benchmark. Furthermore, in a co-simulation, the resource selection becomes critical as every federate's execution time will contribute to the co-simulation's performance. To address this issue (Requirement **R3**), EXPPO provides a resource configuration recommendation system that selects the resource assignment which optimizes the application's QoS performance and user's budget requirements.

Consider the following optimization problem: Suppose the system has a set of homogeneous machines, each with k processing cores. Let $F = \{f_1, f_2, \ldots, f_n\}$ denote a federation (co-simulation) that consists of a set of n federates. Given a resource assignment $R = [r_1, r_2, \ldots, r_n]$ to each federate, the execution time of federate $f_j \in F$ can be expressed as $t_j(r_j)$ when assigned r_j cores. For performance reasons, assume a federate cannot be split among two or more machines, so we have $1 \leq r_j \leq k$. The makespan M for every computation step for the entire federation F is dictated by the slowest running federate (i.e., the straggler), and is defined as $M = \max_j t_j(r_j)$. The execution cost C is given by the total resource used by all the federates over the makespan duration. Since the cores will be reserved for the federation until the slowest federate is done executing, the cost is defined as $C = M \cdot \sum_j r_j$. The resource configuration recommender needs to find a resource assignment R that minimizes $G = M + C = M(+ \sum_j r_j)$, where and denote the user-defined weights to the application's QoS and the execution cost, respectively.

Two additional assumptions are made to solve this optimization problem: (1) federate execution time does not increase with the amount of resources (number of cores) assigned, i.e., $r_j \leq r_j^0$ implies $t_j(r_j) \geq t_j(r_j^0)$; (2) federate execution cost does not decrease with the amount of resources assigned, i.e., $r_j \leq r_j^0$ implies $c_j(r_j) \leq c_j(r_j^0)$, where $c_j(r) = r \cdot t_j(r)$. These are realistic assumptions as many practical applications have *monotonically increasing and sublinear* speedup functions [15], [16], such as those that follow Amdahl's law [17]. As such, the optimization problem can be solved by examining all possible makespan values while guaranteeing the minimum cost. Algorithm 1 presents the pseudocode of this solution with a time complexity of $O(n \log n)$ by maintaining a priority queue for all jobs. Similar approaches can be applied to

Algorithm 1: Resource Configuration Tuner

Input : Execution time $t_j(r)$ for each federate f_j in federation F when allocated different amounts of resources r, where $1 \leq r \leq k$.

Output: A resource assignment $R^* = [r_1, r_2, \ldots, r_n]$ for each federate in the federation that minimizes a linear combination of makespan and cost.

Initialize $r_j \leftarrow 1$ for all $1 \leq j \leq n$;
Compute $G \leftarrow (\max_j t_j(r_j)) \cdot (\alpha + \beta \sum_j r_j)$;
$R^* \leftarrow [r_1, r_2, \ldots, r_n]$ and $G^* \leftarrow G$;
while $\sum_j r_j < nk$ **do**
    $j \leftarrow$ Index of a federate with longest execution time;
    if $r_j = k$ **then**
        break;
    else
        Increment r_j to the next higher profiled resource amount;
        Update $G \leftarrow (\max_j t_j(r_j)) \cdot (\alpha + \beta \sum_j r_j)$;
        if $G < G^*$ **then**
            $R^* \leftarrow [r_1, r_2, \ldots, r_n]$ and $G^* \leftarrow G$;

find the minimum makespan subject to a cost budget or the minimum cost for a target makespan.

C. Federation Machine Scheduling Heuristics

A custom scheduler is required to deploy all the federates in the federation to their respective distributed computing environments. Since the federates cannot run independently of the federation, the scheduling scheme must simultaneously run all of the federates of the federation (Requirement **R4**). This is referred to as *Gang scheduling* or *Bag-of-tasks scheduling* in the literature. To achieve this, EXPPO supports two heuristics to simultaneously schedule the federates on a fixed number m of available machines while utilizing the resource configuration results obtained from Section IV-B. These approaches handle the case where some of the machines are loaded with other compute tasks unrelated to the federation.

The first heuristic is inspired by the First-Fit Decreasing (FFD) algorithm for bin packing and is described in Algorithm 2. The heuristic first sorts all the federates in decreasing order of resource assignment and then tentatively allocates each one of them in order onto the first available machine. If all federates in the federation can be successfully allocated, then the schedule is finalized; otherwise, the entire federation will be temporarily put in a waiting queue to be scheduled later. The time complexity of the heuristic is $O(n(\log n + m))$. The other heuristic is based on the Best-Fit Decreasing (BFD) algorithm that works similarly to FFD, except that it finds, for each federate, a best-fitting machine (i.e., with the least remaining resource after hosting the federate). Note that since finding the optimal schedule (or bin packing) is an NP-complete problem, these heuristics may not always find a feasible allocation for a federation even if one exists. However, once an allocation has been found, it is guaranteed to produce the optimal makespan and cost for the federation by using the resource configuration from Section IV-B.

Algorithm 2: First Fit Decreasing (FFD)

Input : Resource assignment $R = [r_1, r_2, ..., r_n]$ for all federates in a federation. Current available resource $A = [a_1, a_2, ..., a_m]$ of all m machines in the system.

Output: Machine allocation $L = [\ell_1, \ell_2, \ldots, \ell_n]$ of all federates in the system.

Sort all resource assignments in decreasing order, i.e., $r_1 \geq r_2 \geq \cdots \geq r_n$;
Initialize $\ell_j \leftarrow 0, \quad \forall 1 \leq j \leq n$;
for $j = 1, 2, \ldots, n$ **do**
    $fit \leftarrow false$;
    for $i = 1, 2, \ldots, m$ **do**
        if $r_j \leq a_i$ **then**
            Update $a_i \leftarrow a_i - r_j$;
            Set $\ell_j \leftarrow i$;
            $fit \leftarrow true$;
            break;
    if $fit = false$ **then**
        // revert allocations done so far
        for $k = 1, 2, \ldots, j-1$ **do**
            $i \leftarrow \ell_k$;
            $a_i \leftarrow a_i + r_k$;
            $\ell_k \leftarrow 0$;
        break;

V. Co-simulation as a Service Middleware

Figure 4 shows the different components and workflow of the EXPPO co-simulation framework. It is called the Co-simulation-as-a-Service (CaaS) middleware because it enables users to automatically deploy groups of service-based applications to a cloud environment without any concern of resource allocation, application lifecycle monitoring, and cluster management. The functionality of each component is described below.

In ①, the FrontEnd component allows a user to submit a simulation job descriptor in JavaScript Object Notation (JSON) using a Representational State Transfer (REST) Application Programming Interface (API). Each simulation job contains a list of federates, and each federate is run on an individual Docker container. The simulation job descriptor also includes meta-information for each federate, for example, resources required, running status, container image details, etc. The FrontEnd creates a record in a database ④ for each incoming job, then relays the job identifier to JobManager ② which handles resource management. Instead of deploying jobs immediately, the JobManager stores received jobs in a local queue and consults the database about the latest status of cluster resources. It then periodically transfers the information of pending jobs and available resources to JobScheduler ③, which is a pluggable component that implements multiple scheduling algorithms. The JobScheduler either replies with a scheduling decision if the submitted jobs are deployable, or returns a KeepWaiting signal to notify the JobManager that resources are insufficient. The JobManager then forwards deployable jobs to GlobalManager ⑤, synchronizes job status

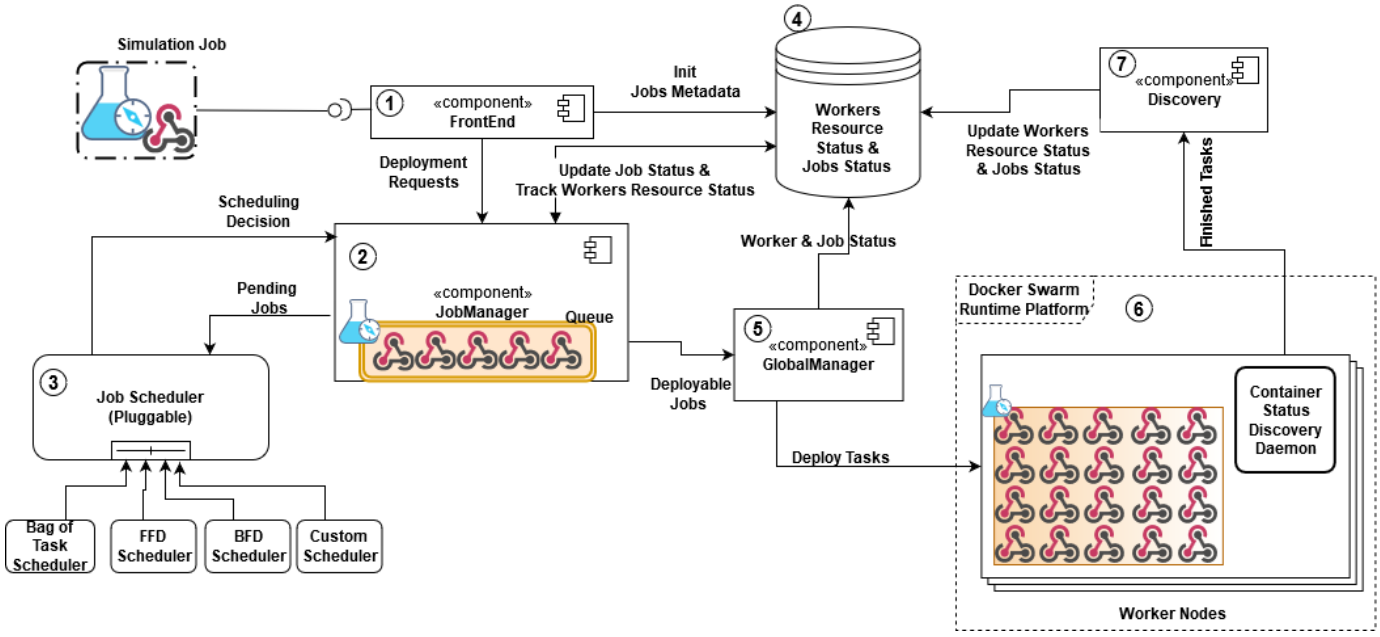

Fig. 4: Co-simulation-as-a-Service.

with the database, and deletes the deployed jobs from its queue. The GlobalManager is responsible for managing participants of the Docker Swarm Runtime Platform ⑥. It launches the master node of the Docker Swarm cluster and accepts registration requests sent from worker nodes. Every joined Worker Node runs a CaaS-Worker daemon that is used to receive and perform commands sent from the GlobalManager. The GlobalManager parses received deployment requests and spawns containers in specific Worker Nodes. Additionally, the Worker Nodes employ a CaaS-Discovery daemon to track the status of containers, and reports a StatusChanged signal to the Discovery component ⑦ when a task is completed.

VI. Experimental Evaluation

A. Experimental Setup

EXPPO was validated using seven homogeneous compute servers with a configuration of 12-core 2.1 GHz AMD Opteron central processing units, 32 GB memory, 500 GB disk space, and the Ubuntu 16.04 operating system. The runtime platform was based on Docker engine version 19.0.5 with swarm mode enabled. There was one client machine that submitted simulation job requests. The front end, job manager, job scheduler and the global manager components were running on a single shared compute server, and five compute worker servers were deployed for running the simulation jobs.

The simulation job consisted of three federates each running a unique application from the PARSEC benchmark: freqmine, blackscholes and ferret. These applications were chosen as they are realistic representations of real-world simulation tasks [12]. During the federation execution, each federate executed its application at every logical time step, for a total number of 100 logical time steps. The implementation of the co-simulation federation was done using Portico HLA [18]. The BFD scheduler was used in the experiments (as it was found to have better performance than FFD). The weights for the QoS and the cost were set to = 1 and = 0:5. These weights are chosen so that they correspond to a specific brand of QoS that prioritizes low makespan over resource usage, which is representative of CPS applications.

B. Experimental Results

EXPPO recommended a resource configuration of 4 cores for freqmine federate, 4 cores for ferret federate and 1 core for blackscholes federate. Two baseline approaches were used to compare the performance of resource configuration selection of EXPPO. In the first approach (*least configuration*), all the federates were assigned the lowest possible configuration of 1 core each. In the second approach (*max configuration*), all the federates were assigned the highest possible configuration of 10 cores each. The performance data was collected over 10 simulation jobs.

Figure 5(a) shows the cumulative distribution function (CDF) of the execution time of EXPPO compared to the other two approaches. The resource configurations selected by EXPPO for the federation had a 90*th* percentile execution time of around 230 seconds, which was significantly better than the 320 seconds for the *least configuration*. Its performance was close to that of the *max configuration* (90*th* percentile execution time of around 200 seconds), with the difference due to its lower resource allocation to conserve the cost.

Figure 5(b) shows the cost analysis of the three strategies using the cost function defined in Section IV-B. As can be seen, resource configuration selected by EXPPO incurred a larger cost than the *min configuration* due to the higher resource allocation to reduce execution time, but it had a substantially lower cost compared to the *max configuration*.

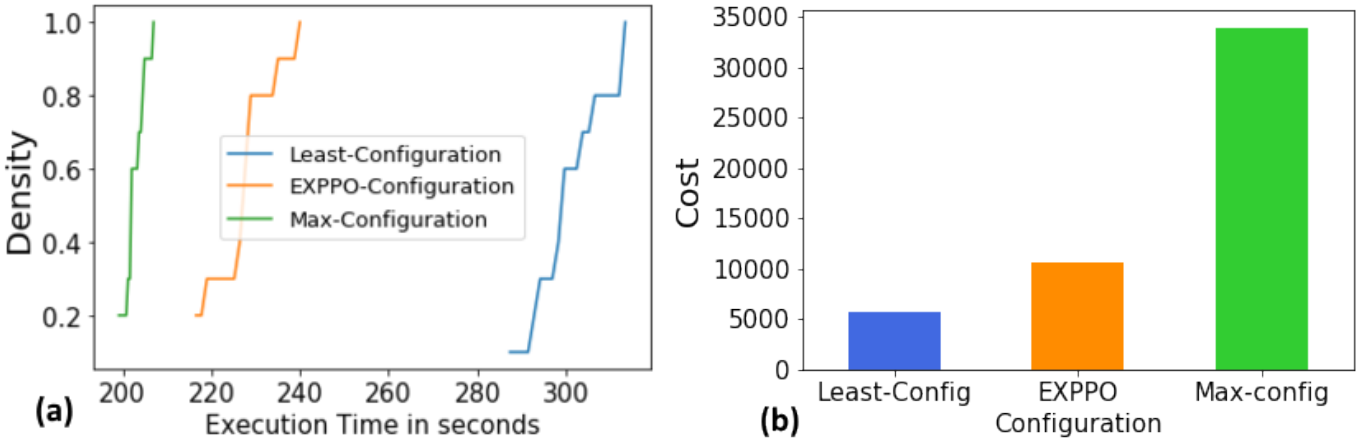

Fig. 5: (a) Execution time for the EXPPO resource configuration compared to other strategies. (b) Cost for the EXPPO resource configuration compared to other strategies.

Overall, the results show that EXPPO is able to select resource configurations and schedule the federates in such a way that minimizes the combination of execution time and cost of the simulation successfully.

VII. Related Work

In [7], the authors presented a Kubernetes co-simulation execution platform for cloud computing environments. Similarly, [8] presented a Docker swarm co-simulation platform for running mixed electrical energy systems simulations. However, these platforms do not use a gang-scheduling based simulation deployment strategy.

For resource recommendation, [19] presented a data-driven approach for selecting the best resource configuration for a virtual machine from a set of different configuration options. [20] explored the cost-sensitive allocation of independent tasks to cloud computing environments. However, these approaches are different from EXPPO as they focus on a single task rather than a pool of BSP tasks.

In [21], the authors proposed a scientific workflows scheduling algorithm to minimize the execution time under budget constraints for deploying to cloud computing environments. [22] proposed an advance-reservation scheduling strategy for message passing interface (MPI) applications. Similarly, [23] presented an approach for gang-scheduling of jobs with different resource needs, such as a compute intensive task paired with a network or I/O intensive task. In [24], the authors present scheduling of multiple container workloads on shared cluster as a minimum cost flow problem (MCFP) constraint satisfaction problem. In [25], the authors proposed a locality-based process placements for parallel and distributed simulation. The evaluation of the proposed framework was carried out using the OMNET++ network simulator.

Compared to related work, EXPPO provides a resource recommendation engine which tries to minimize the makespan and cost for the entire federation (BSP tasks) rather than a single task. EXPPO uses a gang scheduling scheme based on heuristic bin packing techniques to deploy the entire federation on Docker container platform as one batch job. Furthermore, EXPPO provides automatic code generation of distributed tracing probes for performance profiling of the federates which maybe deployed on a distributed infrastructure, thereby relieving the developers from incurring complexities in writing code for the profiling of federates [26].

VIII. Conclusion & Future Works

Resource allocation plays a critical role in co-simulation performance. However, the end user is not necessarily well-equipped to determine what resource allocations work best for their co-simulation jobs given the various resource configuration options (and associated costs) available from the cloud provider. To address these challenges, this paper presents EXPPO, which is a Co-simulation-as-a-Service (CaaS) platform for executing distributed co-simulations in cloud computing environments. EXPPO provides performance profiling capabilities for federates which help in the understanding of the relationship between resource allocations and the simulation performance. Similarly, it addresses performance profiling challenges for a simulation job comprising heterogeneous computation tasks or federates deployed across distributed systems. Furthermore, EXPPO selects the resource configurations for these federates in a way that not only minimizes the makespan of the co-simulation, but also satisfies the cost budget of the user.

Future work will address the following:

The assumption that federate computations have identical wall-clock execution times across all iterations restricts the generality of the approach to a subset of application use cases. For hybrid simulations or simulations with non-linear dynamics, the execution times of the involved simulation steps will vary at each iteration. Future work must explore dynamic resource allocation for federates which have different work loads for different computation steps. Reinforcement learning approaches which can monitor

the federates and dynamically adjust resource allocations based on the varying demand of a federate over time offer a promising approach that remains to be explored.

EXXPO only considers workloads that are CPU bound. It assumes constant communication costs during each computation step for all the federates. Future work must consider the different communication costs for simulations which may need to be constantly updating states or sending messages for triggering discrete events.

When other applications are co-located in the cluster environments, the effects of noisy-neighbors [27], [28] can also affect federation performance. Thus, future work could consider the effect of noisy-neighbors for resource scheduling and simulation placement in the cluster.

EXPPO assumes the use of homogeneous servers for running simulations to simplify its algorithms, and could be extended to include heterogeneous systems. Also, with the growing relevance of edge computing and digital twin techniques, efficient resource allocation and scheduling of the simulations at the edge will be necessary.

IX. Acknowledgments

This work was supported in part by the National Institute of Standards and Technology under award 70NANB18H269; and AFOSR DDDAS FA9550-18-1-0126. Official contribution of the National Institute of Standards and Technology; not subject to copyright in the United States. Certain commercial products are identified in order to adequately specify the procedure; this does not imply endorsement or recommendation by NIST, nor does it imply that such products are necessarily the best available for the purpose.

References

[1] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules." Institute of Electrical and Electronic Engineers New York, 2010, *doi*:10.1109/ieeestd.2000.92296.

[2] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[3] S. J. Taylor, A. Khan, K. L. Morse, A. Tolk, L. Yilmaz, J. Zander, and P. J. Mosterman, "Grand challenges for modeling and simulation: simulation everywhere—from cyberinfrastructure to clouds to citizens," *Simulation*, vol. 91, no. 7, pp. 648–665, 2015, *doi*:10.1177/0037549715590594.

[4] J. Mace and R. Fonseca, "Universal context propagation for distributed system instrumentation," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 8, *doi*:10.1145/3190508.3190526.

[5] Opentracing, "Opentracing specification," https://opentracing.io/specification/, 2020.

[6] T. L. Williams and R. J. Parsons, "The heterogeneous bulk synchronous parallel model," in *International Parallel and Distributed Processing Symposium*. Springer, 2000, pp. 102–108, *doi*:10.1007/3-540-45591-4_12.

[7] K. Rehman, O. Kipouridis, S. Karnouskos, O. Frendo, H. Dickel, J. Lipps, and N. Verzano, "A cloud-based development environment using hla and kubernetes for the co-simulation of a corporate electric vehicle fleet," in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2019, pp. 47–54, *doi*:10.1109/sii.2019.8700423.

[8] Y. Barve, H. Neema, S. Rees, and J. Sztipanovits, "Towards a design studio for collaborative modeling and co-simulations of mixed electrical energy systems," in *2018 IEEE International Science of Smart City Operations and Platforms Engineering in Partnership with Global City Teams Challenge (SCOPE-GCTC)*. IEEE, 2018, pp. 24–29, *doi*:10.1109/scope-gctc.2018.00011.

[9] Docker, "Docker swarm," https://docs.docker.com/engine/swarm/, 2020.

[10] Kubernetes, "Kubernetes :production-grade container orchestration," https://kubernetes.io/, 2020.

[11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[12] C. Bienia, S. Kumar, and K. Li, "Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*. IEEE, 2008, pp. 47–56, *doi*:10.1109/iiswc.2008.4636090.

[13] M. Maróti, R. Kereskényi, T. Kecskés, P. Völgyesi, and A. Lédeczi, "Online collaborative environment for designing complex computational systems," *Procedia Computer Science*, vol. 29, pp. 2432–2441, 2014, *doi*:10.1016/j.procs.2014.05.227.

[14] Y. Barve, S. Shekhar, S. Khare, A. Bhattacharjee, and A. Gokhale, "UPSARA: A Model-driven Approach for Performance Analysis of Cloud-hosted Applications," in *11th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, Zurich, Switzerland, Dec. 2018, pp. 1–10, *doi*:10.1109/ucc.2018.00009.

[15] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008, *doi*:10.1109/hpca.2008.4658638.

[16] B. Berg, J. L. Dorsman, and M. Harchol-Balter, "Towards optimality in parallel scheduling," *POMACS*, vol. 1, no. 2, pp. 40:1–40:30, 2017, *doi*:10.1145/3154499.

[17] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring), 1967, pp. 483–485, *doi*:10.1145/1465482.1465560.

[18] Portico, "Portico," https://github.com/openlvc/portico, 2016, *doi*:10.1093/acrefore/9780199381135.013.5262.

[19] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best vm across multiple public clouds: A data-driven performance modeling approach," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 452–465, *doi*:10.1145/3127479.3131614.

[20] S. Selvarani and G. S. Sadhasivam, "Improved cost-based algorithm for task scheduling in cloud computing," in *2010 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, 2010, pp. 1–5, *doi*:10.1109/iccic.2010.5705847.

[21] X. Lin and C. Q. Wu, "On scientific workflow scheduling in clouds under budget constraint," in *2013 42nd International Conference on Parallel Processing*. IEEE, 2013, pp. 90–99, *doi*:10.1109/icpp.2013.18.

[22] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *1999 Seventh International Workshop on Quality of Service. IWQoS'99.(Cat. No. 98EX354)*. IEEE, 1999, pp. 27–36, *doi*:10.1109/iwqos.1999.766475.

[23] Y. Wiseman and D. G. Feitelson, "Paired gang scheduling," *IEEE transactions on parallel and distributed systems*, vol. 14, no. 6, pp. 581–592, 2003, *doi*:10.1109/tpds.2003.1206505.

[24] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "Concurrent container scheduling on heterogeneous clusters with multi-resource constraints," *Future Generation Computer Systems*, vol. 102, pp. 562–573, 2020, *doi*:10.1016/j.future.2019.08.025.

[25] S. Zaheer, A. W. Malik, A. U. Rahman, and S. A. Khan, "Locality-aware process placement for parallel and distributed simulation in cloud data centers," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7723–7745, 2019, *doi*:10.1007/s11227-019-02973-9.

[26] A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda, "(wip) cloudcamp: Automating the deployment and management of cloud services," in *2018 IEEE International Conference on Services Computing (SCC)*. IEEE, 2018, pp. 237–240, *doi*:10.1109/scc.2018.00038.

[27] Y. D. Barve, S. Shekhar, A. Chhokra, S. Khare, A. Bhattacharjee, Z. Kang, H. Sun, and A. Gokhale, "Fecbench: A holistic interference-aware approach for application performance modeling," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, June 2019, pp. 211–221, *doi*:10.1109/IC2E.2019.00035.

[28] S. Shekhar, Y. Barve, and A. Gokhale, "Understanding performance interference benchmarking and application profiling techniques for cloud-hosted latency-sensitive applications," in *Proceedings of the10th International Conference on Utility and Cloud Computing*. ACM, 2017, pp. 187–188, *doi*:10.1145/3147213.3149453.